THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour l'obtention du titre de

DOCTEUR SPECIALITE AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

par

CANTEGRIT ERIC

MODELISATION ET SIMULATION ORIENTEES OBJETS DES PROCESSUS PAR LOT

Soutenue le 23 février 1988 devant la commission d'Examen:

Mr J.M. TOULOTTE	Président et Directeur de recherches
Mr C. MELIN	Rapporteur
Mr J.P. FRACHET	Rapporteur
Mr C. VASSEUR	Examinateur
Mr J. DEFRENNE	Examinateur
Mme M. FRANCILLARD	Invitée
MR P. FRECHET	Invité

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour l'obtention du titre de

DOCTEUR SPECIALITE AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

par

CANTEGRIT ERIC

MODELISATION ET SIMULATION ORIENTEES OBJETS DES PROCESSUS PAR LOT

Soutenue le 23 février 1988 devant la commission d'Examen:

Mr J.M. TOULOTTE	Président et Directeur de recherches
Mr C. MELIN	Rapporteur
Mr J.P. FRACHET	Rapporteur
Mr C. VASSEUR	Examinateur
Mr J. DEFRENNE	Examinateur
Mme M. FRANCILLARD	Invitée
MR P. FRECHET	Invité

SOMMAIRE

INTRODUCTION

Le travail, présenté dans ce mémoire, a été réalisé au Centre d'Automatique de l'Université des Sciences et Techniques de Lille-Flandres-Artois au sein de l'équipe du Professeur J.M.TOULOTTE, dans le cadre d'un contrat de recherche avec Rhône Poulenc Industrialisation. L'objectif est de construire un outil de modélisation des processus par lot en vue de la simulation de leur comportement.

Les processus par lot sont des systèmes de production, dans lesquels, les matières d'oeuvre passent de poste en poste où elles subissent, à chaque fois, une certaine transformation.

L'investissement humain dans l'industrie coûte de plus en plus cher. L'expérience acquise au cours d'une carrière professionnelle est toujours difficile à communiquer à un débutant. L'apprentissage d'un pilote de processus et les essais de validation, de conception de la commande, par risque d'accident et de dégâts matériels, sont délicats à effectuer sur le site. Un poste de simulation, tel que nous le concevons, permet l'intégration de la résolution de ces problèmes dans le milieu industriel.

Ce mémoire comprend six parties. Dans un premier chapitre, nous donnons la notion d'objet, tels qu'ils sont perçus dans une installation industrielle. Ces derniers sont classés fonctionnellement dans une bibliothèque à usage multiple.

Le deuxième chapitre introduit la philosophie de décomposition en objets d'un système industriel automatisé. Nous réalisons cette approche autant d'un point de vue partie commande que d'un point de vue partie opérative. Nous exposons également les problèmes émanants de cette décomposition.

Dans le troisième chapitre, nous appliquons la notion de programmation objet, en vue de l'obtention des objectifs de la simulation. Cet outil doit principalement permettre la vérification de l'adéquation de la commande vis à vis du cahier des charges, le contrôle de l'efficacité des verrouillages sécuritaires, introduits dans la commande, face aux anomalies, et enfin l'apprentissage de pilotes de conduite à partir des connaissances que les experts ont pû formaliser.

La quatrième partie place le rôle de l'opérateur dans son contexte, afin de modéliser son comportement. Pour cela nous adjoignons à la description des objets, une base de connaissances rendant compte de l'influence sur les objets d'un opérateur expert. Lors de difficultés, que la partie commande n'est plus capable de résoudre.

Dans le cinquième chapitre, nous donnons l'environnement nécessaire à la création de notre outil de simulation. Nous utilisons pleinement l'interactivité et les outils graphiques disponibles dans la programmation objet du type Smalltalk.

Le sixième chapitre, à l'aide d'un exemple, permet au lecteur de se faire une opinion

quant à la facilité d'exploitation de notre outil et la rapidité de développement d'une application.

CHAPITRE 1

L'objet dans une installation industrielle

INTRODUCTION

L'objectif d'une installation industrielle est d'obtenir un produit final de qualité définie, tel que pièces mécaniques dans un atelier flexible, principes actifs dans l'industrie pharmaceutique, additifs aromatiques dans l'industrie chimique..., en partant de matières premières et en tenant compte de critères économiques et énergétiques.

Ce passage de la matière brute au produit souhaité s'obtient grâce à une valeur ajoutée telle qu'un assemblage, un usinage, une réaction chimique, une intervention humaine...

Cette valeur ajoutée est l'objectif global pour lequel a été défini le système de production. Ce dernier est fait d'un ensemble d'objets capable d'assurer l'élaboration progressive de la valeur ajoutée sur les matières d'oeuvre. La connaissance de ces objets, du procédé chimique ou processus de fabrication, permet d'affronter les divers problèmes d'automatisation du système.

Il convient donc d'identifier et de classifier les objets du système à automatiser.

1.1 SCHEMATHEQUE D'OBJETS [FRA 87] [VAL 87] [RP 85] [GRE 85] [TOU 82] [IGL 82]

Dans le schéma d'une installation industrielle, comme celui de la figure 1.1, nous pouvons distinguer les objets naturellement différenciables suivants:

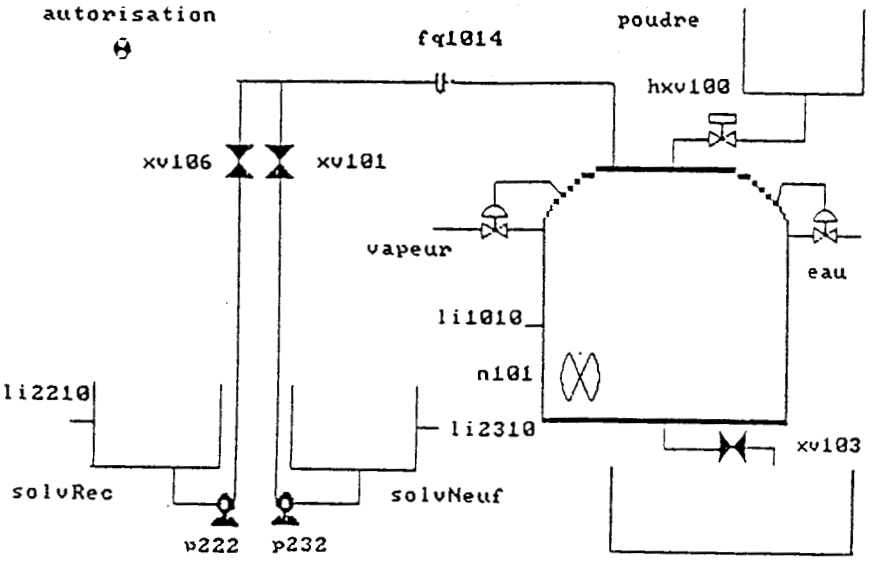


figure 1.1

Pompes
Vanne manuelle
Vannes régulées
Vannes fermeture par manque d'air
Capteurs
Débitmètre
Agitateur
Réservoirs
Réacteur
Tuyaux

Pour l'utilisateur, un objet assure une fonction, un réservoir contient des produits, une vanne laisse passer des matières... Le choix de ces éléments est fait en fonction d'une

mission à accomplir et compte tenu des contraintes physiques caractéristiques, des objectifs de commande, des moyens financiers, etc...

Ces objets constituent une schématèque, celle-ci est un moyen partiel de représentation de l'activité de production. Elle apporte la définition du code des objets et sert d'élément fondamental pour l'édition de la documentation.

D'un point de vue externe, un système de production est constitué d'une partie commande et d'une partie commandée dite partie opérative. La partie commande du système regroupe les missions assignées au système relativement au contrôle d'une part de la valeur ajoutée afin d'élaborer des produits finis, d'autre part maîtriser l'énergie ou l'information de façon rentable.

Les diverses missions d'un système sont assurées par des objets comme ceux répertoriés à la figure 1.1, des hommes, des logiciels...

Une installation industrielle n'est pas une entité isolée, elle s'intègre dans un environnement, avec lequel elle maintient un dialogue.

Celui-ci est caractérisé par l'existence d'un certain nombre de flux:

Le flux de matière porte toutes les informations caractéristiques des matières d'oeuvre absorbées en amont ou évacuées en aval avec les volumes de production... La définition de ces flux de matière est liée aux délimitations du système étudié et permet de modéliser une installation en tant qu'élément d'un système de production plus vaste.

Le flux d'énergie qui peut apparaître à divers niveaux, tout d'abord sous la forme du flux d'énergie de fabrication dont l'importance est considérable sur le plan de l'optimisation de la production. Le flux entrant correspond à l'énergie nécessaire et le flux sortant donne l'énergie produite ou perdue. Un des critères essentiels est actuellement l'économie d'énergie. Le flux d'énergie peut également apparaître aux niveaux des actionneurs d'un système de commande. Son influence est surtout examinées lors de son absence et doit être prise en compte dans les causes d'anomalies. Il en va de même pour l'énergie d'alimentation des systèmes de commande proprement dit.

le flux d'environnement comprend des grandeurs telles que température, humidité, poussière...Il permet de replacer l'installation modélisée dans son contexte physique et intervient à la conception comme contrainte dans le choix des objets ou en exploitation pour introduire des conditions d'alarme ou d'anomalies relativement à une modélisation de l'environnement. Le flux sortant peut correspondre aux déchets et pollutions diverses.

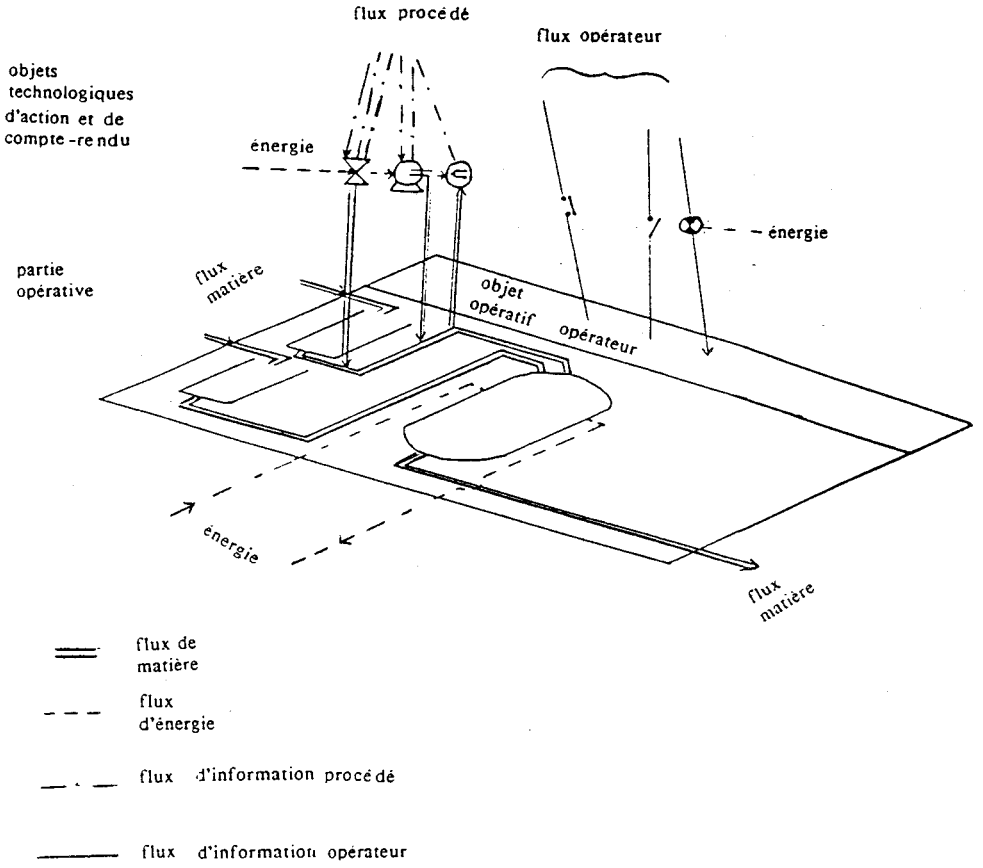


figure 1.2

Le flux informations procédés véhicule les données relatives au processus de fabrication. On y trouve les recettes, les paramètres de fabrication, les bases de données nécessaires à certains postes de travail...

Le flux des informations opérateurs et de contrôle logiciel peut se situer à divers niveaux (opératif, commande, conduite, organisation, décision).

Nous présentons figure 1.2 une structure d'installation et les divers types d'informations qui y circulent.

Dans le cadre de l'automatisation, il est important de bien identifier physiquement et fonctionnellement les objets d'action et les objets de compte-rendu. Les objets d'action ont une influence sur les flux de matières et contribuent à la transformation des produits.

Les objets de compte-rendu précisent les conséquences des actions et caractérisent généralement les grandeurs de la matière d'oeuvre relativement à une mission donnée.

Seuls les objets d'action et de compte-rendu sont contrôlés par le système de commande. Mais, la seule connaissance de ces objets ne suffit pas à la modélisation d'un système de production.Il est intéressant de prendre en compte et de modéliser les objets dits de transfert ou les objets dits passifs, qui interviennent dans le cheminement ou le stockage de la matière d'oeuvre.

Chaque objet peut être décrit suivant différent point de vue. On peut tout d'abord faire apparaître les caractéristiques physiques externes: forme, poids, volume, présentation, connectique... Ceci est important lors de la définition de l'installation pour bien positionner l'objet et l'intégrer à l'ensemble. On peut également s'intéresser à ses caractéristiques fonctionnelles, énergétiques (type d'alimentation, quantité d'énergie) de fiabilité (mtbf selon l'environnement), de coût...

Si la connaissance matérielle du système entraîne une étude fonctionnelle de chacun des constituants, cela ne suffit pas à la modélisation d'une installation industrielle. En effet, ceci exige une connaissance des relations, des interactions entre éléments.

A ce stade, nous pouvons représenter chacun des objets par une boîte (figure 1.3) à laquelle est associée une messagerie correspondant aux différents flux.

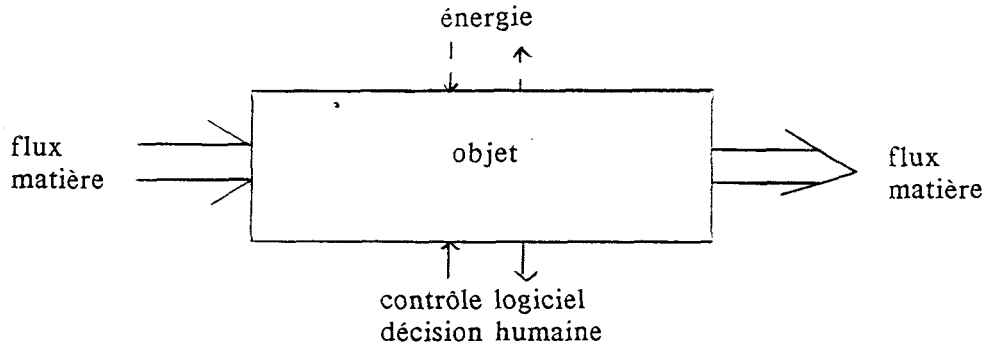


figure 1.3

Nous pouvons faire le rapprochement de notre représentation d'un système avec celle obtenue par la méthode SADT.

SADT (Structured Analysis and Design Technique) es' une méthode de spécification et de conception. Elle est constituée d'un outil graphiqu et d'une méthodologie d'analyse et de description des besoins généraux d'un système. Cette technique utilisent un

ensemble structuré et hiérarchisé de diagrammes dont la construction suit des règles garantissant une approche systématique des spécifications.

SADT différencie le modèle fonctionnel présentant les objectifs du système, du modèle de conception présentant les moyens à mettre en oeuvre. Ces deux modèles sont représentés respectivement par des diagrammes d'activité ou actigrammes et des diagrammes de données ou datagrammes. La représentation SADT est une vision globale du système décomposée progressivement, jusqu'à obtenir les détails jugés suffisants.

Les boîtes (figure 1.4) ainsi obtenues d'un diagramme sont interconnectées par des flèches, dont l'une représente les données d'entrée ou les activités d'entrée, une autre donne le flux des données de sortie ou le flux des activités de sortie, une troisième permet le contrôle sur la boîte et la dernière précise, si nécessaire, le support physique de l'activité ou de la donnée.

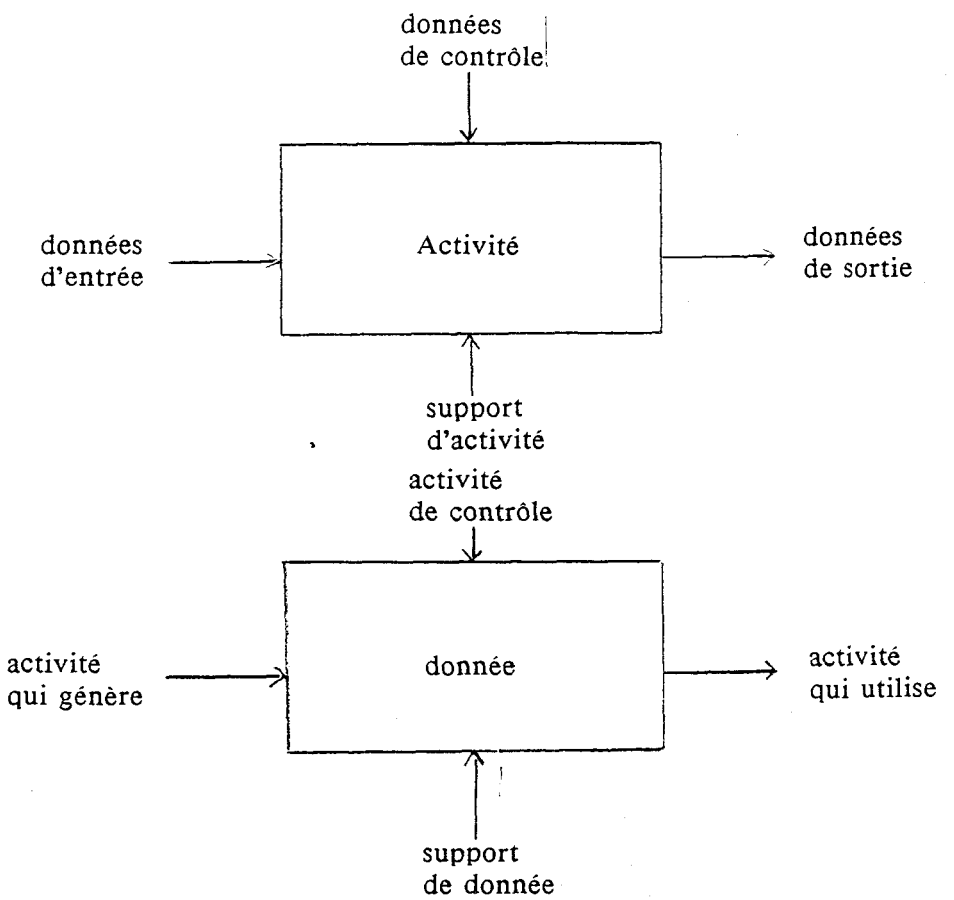


figure 1.4

La méthode SADT est intéressante, si les personnes concernées par la conception sont de formations diverses. La compréhension est facilité par une décomposition

accompagnée de commentaires. Grâce au cycle auteur-lecteur, la qualité de représentation se trouve améliorée.

1.2 LES OBJETS D'ACTION ET DE COMPTE-RENDU

Le lien entre la partie commande et la partie opérative d'un système est assuré par des objets d'action et de compte-rendu.

Les objets d'action présentent plusieurs états. Un état est une configuration fonctionnelle de l'objet. Certains de ces états sont dits intermédiaires. Souvent mal définis, ils permettent d'accéder aux états finaux bien connus. Par exemple, pour une vanne nous considérons un état "ouvert", et un état "fermé". Le passage d'un de ces états final à un autre est obtenu par une multitude d'états intermédiaires correspondant aux différentes configurations de l'objet. Bien que ces états intermédiaires ne semblent pas d'un attrait primordial, dans un premier temps, il est bon de les modéliser dans le cas d'anomalies. Le changement d'état d'un ou plusieurs objets provoque souvent une modification du comportement de la matière d'oeuvre sur le processus.

Le choix de ces objets d'action va permettre la description de la définition d'une commande en précisant les effets à obtenir sur la partie opérative du système. Cela revient à décrire un ensemble d'éléments appelés ordres et à indiquer quand et comment il faut les appliquer. Un ordre envoyé à un objet d'action entraîne un changement d'état de cet objet.

Les objets d'action peuvent travailler de deux façons différentes. En boucle ouverte, la commande ne vérifie pas si l'action s'effectue, ni comment elle s'effectue. En boucle fermée, l'objet communique son état à la commande, elle peut donc contrôler l'effet engendré sur cet objet (figure 1.5).

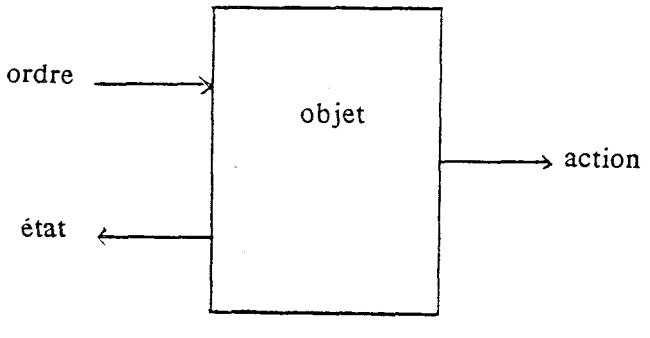


figure 1.5

Un objet d'action contribue à la réalisation d'une mission.

Seuls, ces objets ne suffisent pas à l'automatisation du processus. On y adjoint les objets de compte-rendu qui communiquent à la commande des informations sur l'évolution du système.

L'objet de compte-rendu permet à la partie commande ou à une sous partie commande de contrôler l'aboutissement ou la bonne exécution d'une mission. Il possède divers états. Le passage d'un état à l'autre est provoqué par l'évolution des caractéristiques de la matière d'oeuvre, par exemple l'évolution d'une température prise en compte par une thermo-sonde.

1.3 LES OBJETS DE TRANSFERT

Les objets de transfert ont une existence physique mais ils échappent au contrôle des automatismes. Ils permettent le cheminement de la matière d'oeuvre.

Bien que ces objets n'influent pas sur l'étude de l'automatisation, le concepteur les considère comme des objets à part entière. Prenons par exemple l'étude du transfert d'un produit d'une cuve à une autre. Le concepteur étudie le type de canalisation à utiliser, le diamètre du tuyau selon le débit désiré, le matériau à employer en fonction du fluide...

En simulation, ces objets présentent l'intérêt de modéliser certains comportements de la matière d'oeuvre. Ils permettent, d'une part de faire apparaître les défaillances telles que l'encrassement, l'obturation, une fuite dans le cas d'une canalisation ou la saturation, le blocage de pièces pour un convoyeur... et d'autre part de modéliser des phénomènes physiques plus particuliers pouvant influencer l'évolution des matières d'oeuvre tels que des turbulences naissant à l'intersection de deux entrées d'une canalisation.

1.4 LES OBJETS PASSIFS

Comme les objets de transfert, les objets passifs ne subissent pas le contrôle direct des automatismes. Ils ne créent pas d'action sur la matière d'oeuvre, mais ils modifient les qualificatifs de ses caractéristiques. Le débit de produit entrant d ns un réservoir est transformé en élévation de niveau.

En simulation, l'étude de ces objets ne présentent aucun intérêt, seules les transformations qui s'y produisent sont modélisées.

1.5 NOMENCLATURE DES OBJETS [BAU 86]

Malgré la grande diversité des installations industrielles, le concepteur réutilise souvent les mêmes objets ou des groupes d'objets habituels, en modulant toutefois les paramètres caractéristiques. Il est extrêmement rare à un certain niveau de créer spécialement un objet pour une application. C'est pourquoi, il est utile de construire pour chacun de ces objets un modèle générique, valable pour tous les éléments d'une catégorie, dont le comportement statique ou dynamique est identique.

Tout objet ou groupe pertinent d'objets peut être vu comme une instance d'un modèle. Dans ce cas ils possèdent un ensemble de propriétés accessibles à l'instanciation. Celle-ci, personnalisation du modèle, consiste à affecter des valeurs à certains attributs et permet les accès au comportement statique et dynamique du modèle .

Compte-tenu de la réutilisabilité, l'intérêt est de regrouper les modèles dans une base de données appelée nomenclature s'enrichissant avec l'expérience.

Chacun des éléments de cette base est un modèle de connaissance technologique, graphique, fonctionnelle, relationnelle de chaque objet... Une des principales caractéristiques de ces éléments est la généricité, permettant ainsi la réutilisation du même modèle dans une ou plusieurs applications.

Cette nomenclature constitue une documentation complète, dans laquelle les consultations sont aisées et les transferts d'informations significatives, automatiques. Il est donc indispensable de rendre lisible la description des éléments pour différents utilisateurs. La base de données établie peut être utilisée, comme base de choix technologique, de CAO d'atelier ou d'automatisation. Elle permet d'assurer un contrôle de cohérence des différents documents techniques et schémas d'installations. Elle sert également de déclarations aux programmes rédigés dans le langage de description de la commande et dans celui de la partie opérative pour la simulation.

La définition d'une application consiste dans un premier temps, à extraire de la nomenclature les modèles utiles et de constituer de cette façon la base des modèles de l'application. Celle-ci peut d'ailleurs être éventuellement complétée par quelques modèles spécifiques. Il suffit ensuite d'instancier les objets dont les modèles sont décrits dans la nomenclature de l'application.

1.6 NOTION ET ROLE DES CLASSES D'OBJET

En observant la liste des objets de la figure 1.1 donnée précédemment, on peut remarquer une classification assez naturelle de ces éléments. En effet, certains objets peuvent avoir un comportement statique et dynamique général semblable, mais particularisés par un comportement qui leur est propre. C'est la raison pour laquelle, il nous semble judicieux de classer les différents modèles d'objets selon une hiérarchie. Ces modèles s'appellent désormais des classes.

Une classe est un type abstrait générique. Elle possède des paramètres dont les valeurs sont propres à chaque instanciation et décrit les comportements de ces objets. L'ensemble de ces classes, modules fonctionnels, constituent dès lors une nomenclature.

La notion de hiérarchie de classes facilite l'organisation de la base de données. En effet, cette notion permet non seulement de structurer la base de données, mais de plus elle simplifie la description des classes. Celles-ci disposent des attributs et du comportement décrit dans les classes hiérarchiquement supérieures; elles héritent donc des attributs et du comportement de leurs surclasses.

CONCLUSION

Dans ce chapitre, nous avons identifié les différents types d'objets intervenant dans une installation industrielle. En mettant en évidence les liaisons indispensables à la modélisation, afin de restituer l'environnement dans lequel elle se trouve immergée. Nous rappelons les bases de la méthode SADT, qui présente des analogies avec la décomposition que nous envisageons.

Une bibliothèque regroupe en différentes classes, les objets répertoriés dont nous avons décrit les comportements fonctionnels, technologiques...

Toutefois ces objets ne suffisent pas à la description d'un système. Il est nécessaire de définir la commande et le comportement du système lui-même.

INTRODUCTION

Les objets d'une installation industrielle, éléments de base de la nomenclature, ont fait l'objet du chapitre précédent. L'instanciation des classes ne suffit pas à la description de l'application. Il faut de plus représenter le comportement des interactions pouvant exister entre ces différents objets, le traitement ou les réactions sur la matière d'oeuvre, la commande pilotant certains objets.

C'est pourquoi, dans ce chapitre, nous donnons les principes, que nous avons adoptés pour la décomposition de tout système de production.

Nous trouvons tout d'abord les objets opératifs représentant le comportement de la matière d'oeuvre en fonction de l'état des objets d'action et des objets compte-rendu. Ces objets opératifs prennent en compte le comportement des objets de transfert et des objets passifs. Ils intègrent également les réactions de l'environnement et réagissent au comportement de l'opérateur de conduite.

Il existe aussi des objets missions et éventuellement des objets de commande, qui assurent la coordination des dispositifs opératifs du processus automatisé.

Les interactions entre les parties opératives et les missions s'obtiennent par le biais des objets d'action et de compte-rendu, qui dialoguent par message comme tout objet du système.

2.1 L'OBJET OPERATIF

Un ou plusieurs objets technologiques d'action agissent sur la matière d'oeuvre. Une réaction, obtenue via des objets de transfert ou des objets passifs, est traduite par un ou plusieurs objets de compte-rendu. Ces objets opératifs servent à donner le comportement de la matière d'oeuvre en fonction des états des objets d'action de manière à générer les valeurs pour les objets de compte-rendu.

Des variables sont utilisées en entrée ou sortie des modèles. Les objets d'action forcent la valeur d'une ou plusieurs variables. Les objets comptes-rendus peuvent lire certaines variables caractéristiques de la matière d'oeuvre. Celles-ci satisfont aux lois physiques, chimiques, biologiques... ou ont des valeurs définies à partir d'un modèle expert.

Le modèle lui même peut être:

-Evénementiel, en utilisant des temporisations et des variables binaires.
-Physique, en utilisant des équations ou des tables.
-Stochastique, soit en jouant sur le temps d'exécution soit en introduisant des variations sur les grandeurs propres à la matière d'oeuvre.
-Expert, en faisant référence à l'expérience, si aucune loi n'est connue.

L'introduction de ces différents types de représentation à ce niveau permet de considérer toutes anomalies intervenant sur le processus, soit comme causes premières physiques, chimiques, biologiques...,ou technologiques, soit comme causes secondaires (conséquences des précédentes).

En effet, les objets technologiques définis ultérieurement sont supposés avoir un comportement idéal.

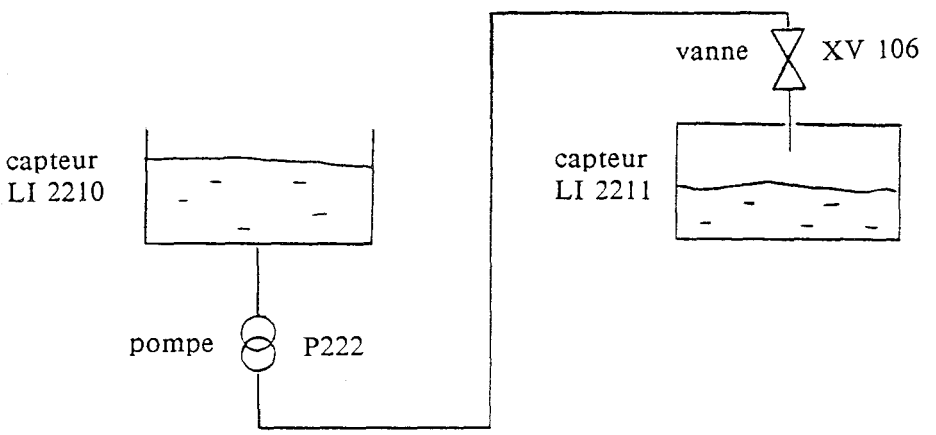
Pour avoir une meilleure vue des objets opératifs, nous considérons l'exemple de remplissage d'une cuve. Nous connaissons le débit de la vanne d'arrivée et de la canalisation et nous mesurons le niveau de produit dans la cuve.

Dans le modèle événementiel, l'ouverture de la vanne lance une temporisation, présélectionnée à une ertaine valeur. Lorsque le temps est écoulé la variable binaire niveau est mise à 1. Les seuls défauts possibles sont des modifications de temporisation ou de la variable binaire (mise à 1, mise à 0, modification de la présélection).

Le deuxième modèle correspond à l'usage de lois physiques. Nous introduisons une variable "flux", une variable "niveau" lisible par l'opérateur par l'intermédiaire d'un objet de compte-rendu. Celui-ci peut transformer les valeurs analogiques en valeurs binaires pour le dispositif de commande. Il est possible d'introduire des défauts plus précis que les variations de temporisation. Pour l'opérateur de conduite, l'influence de

cette modélisation est importante, car ceci peut lui permettre un meilleur contrôle lors de défaillances. Par exemple, si la cuve est à moitié pleine après une défaillance, l'opérateur peut décider de démarrer la phase suivante avec les produits en quantité proportionnelle au niveau obtenu. La loi physique peut être représentée par une équation de recurrence dont le pas d'échantillonnage doit être judicieusement choisi.

L'utilisation de lois stochastiques peut être intéressante, dans le premier modèle pour représenter la dispersion du temps de remplissage, dans le deuxième pour moduler le débit. Ces lois stochastiques permettent de représenter les variations aléatoires d'événements. En effet, une même séquence d'événements, lancée plusieurs fois, peut se dérouler différemment. Ces lois introduisent également l'occurrence d'anomalies



objets technologiques:	XV106	actions
	P222	
	LI2210	compte-rendu
	LI2211	

figure 2.1

Le dernier modèle fait appel aux systèmes experts, utilisés par la modélisation des événements de lois inconnues ou complexes. En effet, le comportement de certains traitements ou certaines réactions chimiques ou biologiques n'est pas complètement maîtrisé. Il est possible de les représenter, si un utilisateur expert est capable, grâce à

son expérience ou à son observation, d'établir une base de connaissances. Celle-ci doit permettre de modéliser les réactions à l'occurrence de différents événements, à introduire les défaillances causées par l'usure des objets.

Généralement la concordance des comportements de plusieurs objets technologiques est nécessaire pour obtenir une réaction sur la matière d'oeuvre.

Considérons par exemple le dispositif de la figure 2.1:
le débit à la sortie de la vanne ne dépend pas uniquement de celle-ci, mais aussi de la pompe, du diamètre du tuyau, du niveau dans le réservoir d'alimentation et du niveau dans la cuve réceptrice, si celle-ci est en vase clos.

Le modèle doit gérer ou utiliser:

le niveau de la cuve de stockage.
le niveau de la cuve réceptrice.
l'ouverture de la vanne.
le débit de la pompe.
et éventuellement le débit du tuyau, si on souhaite introduire des anomalies d'encrassement et de bouchage.

Le transfert du produit peut par ailleurs être compromis par des défauts des objets technologiques.

La décomposition éventuellement hiérarchisable de la partie opérative est souvent statique, c'est à dire qu'elle n'évolue pas au cours du temps.

Néanmoins, il peut exister des ressources partageables, en effet dans une industrie chimique, un même réacteur peut être utilisé à plusieurs fins, auquel cas, il est nécessaire d'avoir recours à un arbitre, dont le rôle est de contrôler les conflits d'accès à cet objet au cours du temps.

L'organisation de la partie opérative peut être dynamique. Pendant la production cette décomposition évolue, afin de rentabiliser au mieux l'activité d'un atelier, flexible par exemple. Un centre de ressources gère l'activité des différents objets, il leur envoie des ordres d'attente, de préparation, d'exécution... Ce centre d'organisation fait face également aux problèmes de reconfiguration en cas d'anomalie.

2.2 L'OBJET TECHNOLOGIQUE

La définition du comportement des objets technologiques est identique quelque soit l'implémentation. Il suffit donc de les décrire une seule fois, ce qui permet d'utiliser une base de données d'objets technologiques. Celle-ci donnera aussi bien les caractéristiques statiques, morphologiques, de connectiques,..., que les variables d'état et les procédures permettant de les faire évoluer.

Nous distinguons deux types d'objets technologiques, ceux d'action agissant sur la matière d'oeuvre et ceux de compte-rendu, traduisant les réactions de la matière d'oeuvre.

Les objets technologiques jouent le rôle d'interface entre les objets opératifs et les objets missions, définis au paragraphe suivant.

Un objet technologique d'action reçoit les ordres d'un objet mission dont il dépend. En retour il donne accès à son état.

Les actions, sur la matière d'oeuvre, d'un objet technologique sont prises en compte au niveau de l'objet opératif. Leurs incidences sont transmises par ce dernier à un objet de compte-rendu.

A la demande de l'objet mission, l'objet de compte-rendu lui communique le résultat conséquent aux actions effectuées sur la matière d'oeuvre.

Un objet technologique est le plus petit sous ensemble d'objets du processus, dont nous acceptons de décrire le comportement. Tout élément, éventuellement unique, de cet ensemble joue un rôle dans l'action de l'objet technologique. Les interactions existant au sein de l'objet technologique dépendent uniquement des éléments de l'ensemble. Un objet technologique dialogue uniquement avec un objet mission, un objet opératif, éventuellement l'objet opérateur pour la commande manuelle, à l'exclusion de tout autre objet technologique.

Une vanne associée à ses capteurs de fin de course est donc considérée comme un objet technologique (figure 2.2). Le débitmètre doit être séparé de la vanne, puisque le couplage débitmètre vanne est obtenu par le flux matière, et que le dialogue entre chacun de ces objets technologiques est établi par le biais d'un objet opératif et d'un objet mission.

Dans le cas d'une vanne régulée, il est impératif de placer le régulateur, le débitmètre et la vanne dans le même objet, puisque la pression de commande de la vanne issue du régulateur n'est pas directement élaborée par la commande et que la sortie du

débitmètre n'est pas communiquée à la commande mais à l'entrée du régulateur (figure 2.3).

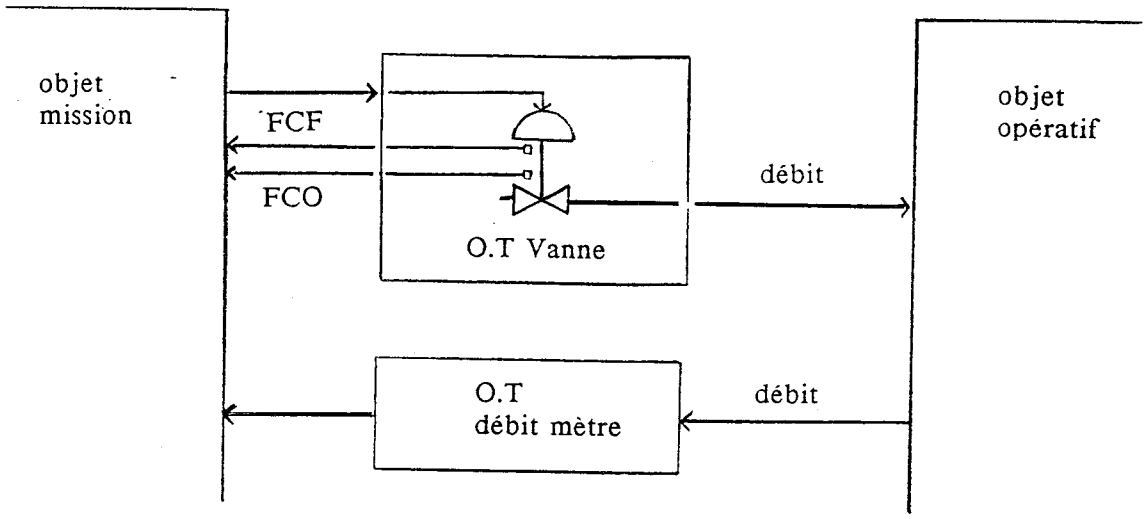


figure 2.2

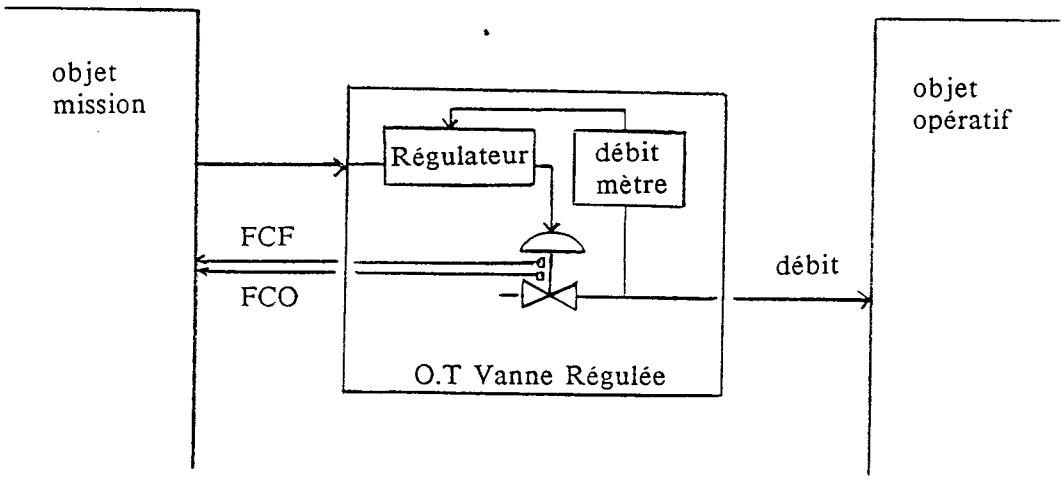


figure 2.3

Un objet technologique est supposé avoir un comportement idéal. Si par exemple le débit à la sortie de la vanne de l'exemple précédent (figure 2.1) ne donne pas de valeur correcte du fait d'un encrassement, cette variation est introduite par la variable correspondante au niveau de l'objet opératif.

2.3 L'OBJET MISSION

Un système de production industriel est fait pour assurer une ou plusieurs missions, décomposables en sous missions... jusqu'à définir des missions élémentaires exécutant l'activation des objets technologiques tels que moteurs, vannes, pompes... Le contrôle de la bonne exécution de ces missions est fait via des objets technologiques de compte-rendu.

L'objet mission a pour but d'exécuter une tâche du système de production. Pour cela, il regroupe un certain nombre d'objets technologiques lui permettant d'accomplir sa tâche. Cet ensemble d'objets technologiques est purement fonctionnel. Le chargement d'une cuve peut être un objet mission, il regroupe dans ce cas les objets technologiques nécessaires au remplissage tels qu'une vanne, une pompe... On retrouve une décomposition similaire à celle établie pour les objets opératifs (exemple de la figure 2.1).

Cette notion de mission est assez naturelle dans l'étude des systèmes industriels. Considérons par exemple, un réacteur dans une industrie chimique. Cet objet agit sur la matière d'oeuvre par l'intermédiaire des objets technologiques qui le composent (vannes, agitateur, capteur de niveau, régulateur de chauffage...). Cet ensemble est formé dans le but de remplir une mission. Nous voyons apparaître aussi la notion de hiérarchie des missions, un des composants du réacteur est un régulateur de chauffage, qui est lui même un objet mission. L'organisation des objets missions suit la hiérarchie des missions du processus. Ces objets ne peuvent effectuer simultanément qu'une seule mission. Un objet mission est dit élémentaire, lorsqu'il appartient au niveau le plus bas de la hiérarchie , et dialogue par conséquent directement avec des objets technologiques.

La décomposition en objets mission est duale de celle des objets opératifs. Cette décomposition est généralement statique, toutefois nous allons montrer à l'aide d'un exemple que celle-ci peut être dynamique.

Prenons l'exemple illustré à la figure 2.4 du transfert d'un produit d'un réservoir de stockage R0 vers des cuves R1 et R2.

Nous pouvons imaginer qu'au cours du temps trois missions sont possibles:

Transfert de R0 vers R1
Transfert de R0 vers R2
Transfert de R0 vers R1 et R2

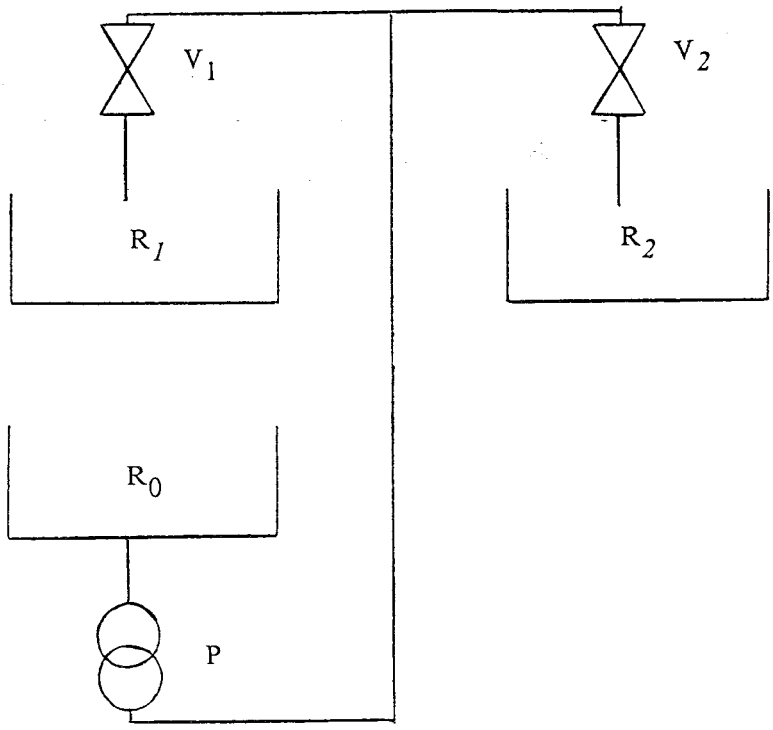


figure 2.4

Chaque mission concerne des objets technologiques différents. Nous constatons que la décomposition est dépendante de la mission et qu'une mission n'a de sens qu'à un instant donné. Ceci nous amène à considérer une décomposition évolutive, un centre de décision est donc nécessaire pour gérer l'utilisation cohérente des objets technologiques.

Dans le cas d'objets technologiques placés en situation de ressources communes, un objet mission particulier de niveau décisionnel organise l'activation des objets technologiques concernés de façon à régler les conflits d'accès. Dans l'exemple précédent, avec uniquement les deux premiers cas, en les considérant comme incompatibles, il convient d'introduire un objet arbitre entre les deux missions: transfert de R0 vers R1 et transfert de R0 vers R2.

Un objet mission peut lui même être vu comme une ressource pour les niveaux supérieurs. En effet, plusieurs objets missions peuvent envoyer des ordres à un objet mission ressource. Considérons l'exemple (figure 2.5) d'une cuve recevant à certains moments un produit A, à d'autre un produit B, sans qu'il y ait simultanéité.

L'objet mission "remplissage" reçoit des ordres de deux autres objets mission hiérarchiquement supérieur. La ressource commune est gérée dans ce cas par un dialogue entre ces deux objets. Dans ce dialogue, il est possible d'établir une priorité, un temps d'attente entre les deux opérations...

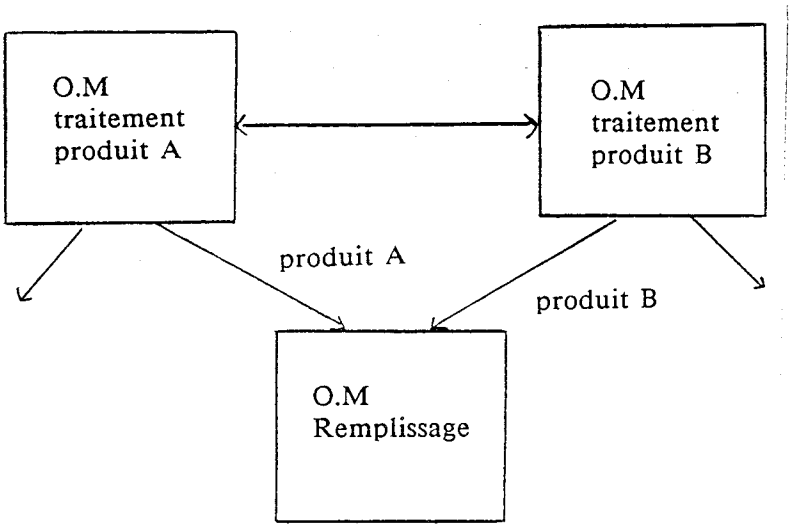


figure 2.5

2.4 PROBLEMES LIES A LA DECOMPOSITION

Bien qu'une définition de l'objet technologique ait été donnée précédemment, il est évident que la relative indépendance des objets technologiques est laissée à l'appréciation du concepteur.

Si pour certains, une vanne comprend la vanne proprement dite et ses capteurs de fin de course. Pour d'autres, la vanne et ses capteurs constituent des objets différents.

Cette notion de relative indépendance est retrouvée au niveau des objets opératifs. Mais cette fois le problème se pose pour le regroupement des objets technologiques liés à l'influence des débits matières.

Dans le cas d'une ressource commune, figure 2.4, où les vannes V1 et V2 ne doivent jamais être ouvertes simultanément, il semble naturel de regrouper le comportement des objets réservoir d'alimentation, pompe, V1, V2, au sein du même objet opératif. Toutefois certains concepteurs pensent, qu'il est préférable de regrouper le comportement de la pompe et du réservoir dans un premier objet opératif, les vannes et leur cuve respective dans deux autres objets opératifs.

Il n'est pas aisé de définir les frontières entre les objets opératifs. En effet, nous pouvons soit regrouper l'influence de tous les objets technologiques dans un seul objet opératif ou au contraire associer à chacun d'eux un objet opératif. Il est évident, que ces deux cas extrêmes sont à éviter, un juste compromis doit être trouvé par le concepteur.

La décomposition en objets missions présente les même problèmes. De plus, ces objets doivent être décomposés de manière, qu'à un instant donné, chacun d'eux effectue au plus une mission.

2.5 L'OBJET PARTIE COMMANDE

[ALB 82] [TOU 82] [DAL 83] [MOT 84] [BAU 86]

Précédemment, nous avons déterminé le but des objets missions. Ils décrivent l'aspect fonctionnel de notre simulation. Celui-ci peut être remplacé par une structure de commande. Cette substitution se matérialise par une projection de la structure de commande sur les objets missions(figure 2.6).

L'objet partie commande est dépendant de la structure du système de commande envisagée, en particulier relativement au système de communication entre les sous-parties de commande.

Une première façon de modéliser un objet de commande est de considérer une spécification formelle et d'en faire la simulation. Les outils tels que graphes d'état, grafcet, réseau de pétri, graphe booléen structuré hiérarchisé, langage Easymitis, se prêtent bien à ce type d'approche. Ils permettent de faire la liaison avec un certain nombre de dispositifs d'aide à la conception d'automatismes allant jusqu'à la mise en oeuvre matérielle.

Un deuxième niveau de description de l'objet commande est possible. Il correspond à ce qui se réalise sur une machine programmée donnée, comme par exemple un automate programmable du commerce. On a alors recours à la simulation de l'automate, afin de contrôler l'adéquation entre la commande et le cahier des charges.

L'objet ou les objets de commande doivent suivre les règles du langage employé, aussi bien sur le plan statique que dynamique. Il convient ainsi de modéliser précisément le déroulement cyclique de la machine et de le rendre relatif à l'échelle de temps procédé.

L'objet commande introduit une dimension supplémentaire dans la simulation de nos systèmes. Il devient maintenant possible de simuler ausi bien l'ensemble partie opérative-spécification fonctionnelle que l'ensemble partie commande-partie opérative.

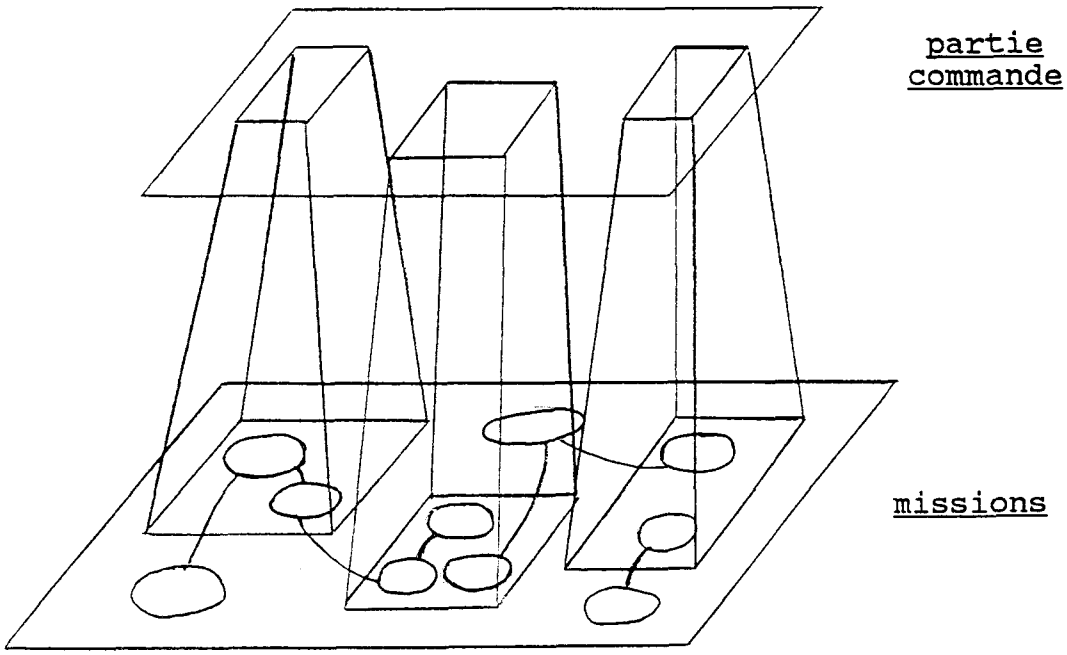


figure 2.6

Il est toutefois, à noter, que la description de la commande n'est pas toujours facile à obtenir. Il faut en effet, pouvoir teni. compte de la structure, soit en système centralisé, soit en système réparti et/ou hiérarchisé.

2.6 COMMANDE CENTRALISEE ET COMMANDE REPARTIE [DEN 83] [VAL 87]

En ce qui concerne la mise en oeuvre d'un système de commande, il est possible de concevoir des supports informatiques, tels que les automates, de plus en plus puissants,

afin d'effectuer une commande centralisée de l'installation. Ceci revient à projeter sur l'ensemble des objets missions un seul dispositif de commande.

Il est possible aussi d'envisager des supports informatiques interconnectés en réseau de façon à effectuer une commande répartie.

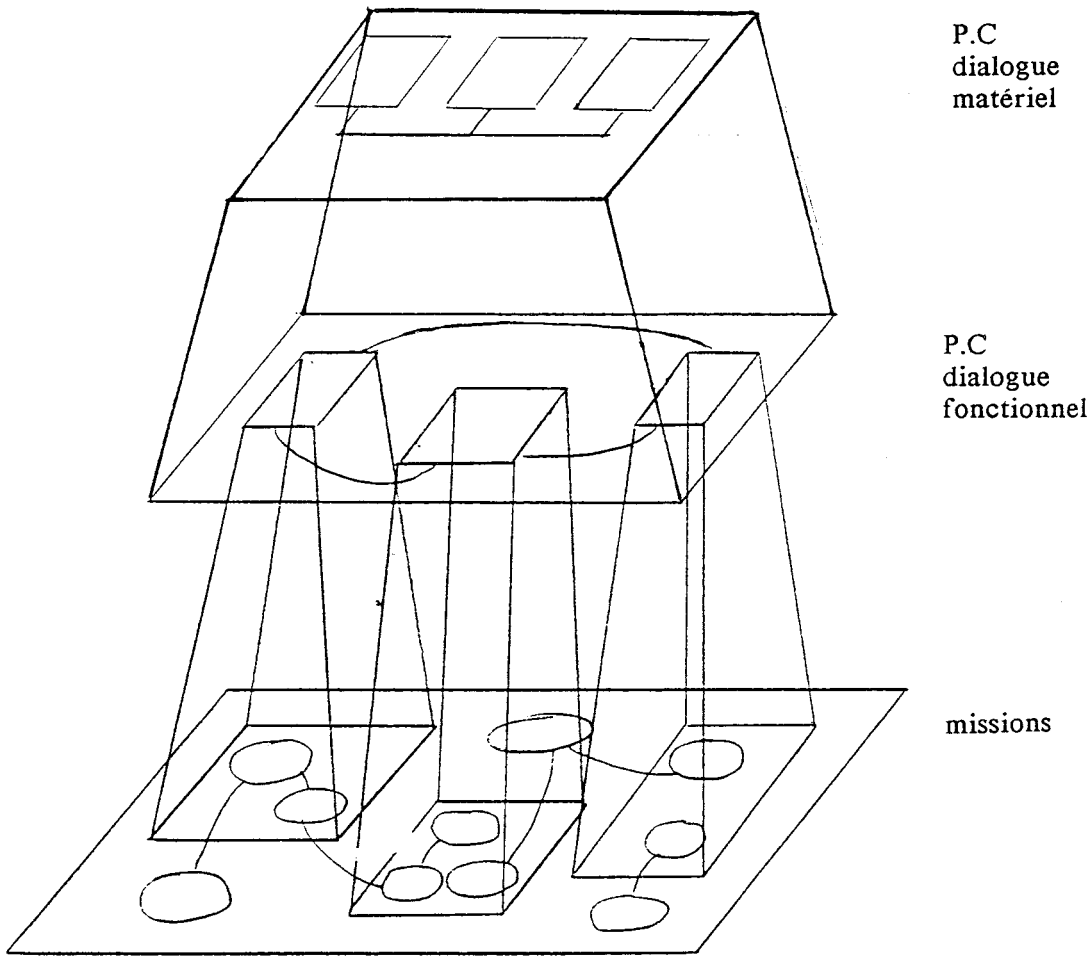


figure 2.7

Les problèmes, qui se posent, sont alors liés à leur communication et à la mise en oeuvre d'une répartition cohérente de l'application, c'est à dire comment définir la projection sur le niveau mission. Celle-ci doit suivre l'organisation des missions et essentiellement utiliser la notion de hiérarchie, qui a pu y être introduite. La projection du niveau commande sur le niveau mission ne doit pas couper une mission.

Pour une projection donnée, la communication entre les parties commandes coïncide fonctionnellement avec certains éléments de dialogue inter-missions. Il convient alors de la transformer en structure physique de communication. Celle-ci peut évidemment conserver la notion de message dans une communication point à point. Mais, il est maintenant plus fréquent de devoir introduire un objet réseau de communication gérant les dialogues et leurs essentiels conflits (figure 2.7).

Dans tous les cas, le système de communication doit garantir toutes les fonctionnalités du dialogue et en particulier une bonne synchronisation des objets missions par le biais des messages en appel réponse.

2.7 RELATIONS ENTRE OBJETS, MESSAGES, MISE A JOUR

Le message est le moyen de communication entre des objets. Un message est constitué du nom de l'objet destinataire, ce qui rend le dialogue explicite, et du traitement à effectuer sur cet objet avec éventuellement passage de paramètres.

L'appel-réponse est le principe de base du fonctionnement des messages. Un objet est en attente, tant qu'il n'a pas reçu une réponse au message envoyé. Cette réponse peut être employée pour émettre un nouveau message. Plusieurs messages ne peuvent pas être envoyés simultanément, sauf lors de l'introduction du parallélisme.

Une représentation chronologique des messages est particulièrement utile. Un exemple est donné sur le graphique de la figure 2.8.

Dans notre exemple, les objets sont symbolisés par une lettre majuscule, les messages par une lettre minuscule.

Un objet A doit envoyer successivement les messages b et c respectivement aux objets B et C.

A la réception du message b, l'objet B doit envoyer successivement les messages d et e respectivement aux objets D et E.

A la réception des messages c, d, e, les objets respectifs peuvent répondre directement sans envoyer d'autres messages.

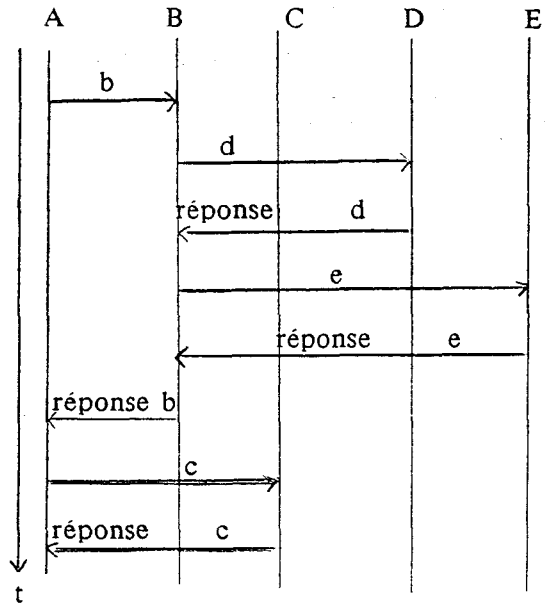


figure 2.8

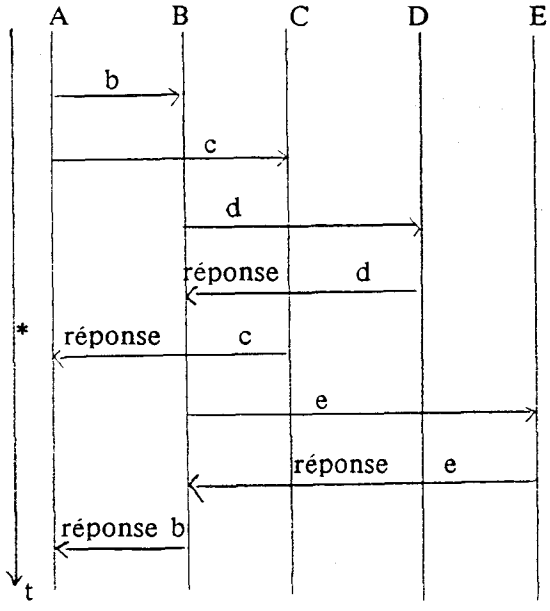
L'objet A envoie le message b à l'objet B. Cet objet A ne peut envoyer le message c qu'après avoir reçu la réponse du message b. Comme le montre le chronogramme, il en est de même pour tout autre objet.

Dans la pratique des systèmes de commande de conduite industrielle, il est nécessaire de pouvoir envoyer plusieurs messages simultanément.

Prenons l'exemple figure 2.9, où l'objet A doit envoyer les messages b et c simultanément, afin de représenter un parallélisme, puis un message f à l'objet E. Les messages d et e sont envoyés successivement par l'objet B, comme dans l'exemple précédent.

Ce dernier diagramme montre l'enchaînement des messages.

L'objet A envoie le message b à l'objet B et le message c à l'objet C. L'objet B envoie à son tour le message d. A sa réponse, imaginons que l'objet C retourne la réponse du message c à l'objet A. Dans ce cas, ce dernier objet positionne un indicateur de synchronisation, qui permettra à A de continuer son traitement, à la réponse du message b.



* indicateur de synchronisation

figure 2.9

L'objet B envoie alors le message e, la réponse de celui-ci entraîne celle du message b. L'objet A peut envoyer le message f à E.

Il existe un autre moyen de dialogue entre les objets, qui permet les mises à jour de tous les objets déclarés dépendants de l'objet établissant cette communication. Les messages sont alors implicites. Plusieurs objets peuvent dépendre d'un même objet. Cette relation de dépendance est décrite à leur définition. A la réception d'un message spécifique sur l'objet maître, tous les objets dépendants subissent une mise à jour liée à celle de l'objet maître. L'usage de dépendance réciproque et conditionnelle permet de représenter implicitement des structures d'automates complexes multi-objets. Toutefois, l'étude de la stabilité globale de l'ensemble doit être soigneusement effectuée de façon à éliminer les cas critiques. Ce type de communication requiert donc une très grande attention à la déclaration des objets maîtres, des objets dépendants et ne doit pas être utilisé trop systématiquement, car il risque de faire perdre le sens des réalités d'organisation du système. Le moyen de dialogue peut toutefois être envisagé de façon utile dans la notion d'objets composites.

2.8 EXEMPLES D'ILLUSTRATION

premier exemple: [BOS 79]

Soit une chaîne de manutention et de traitement figure 2.10 dont nous donnons le cahier des charges.

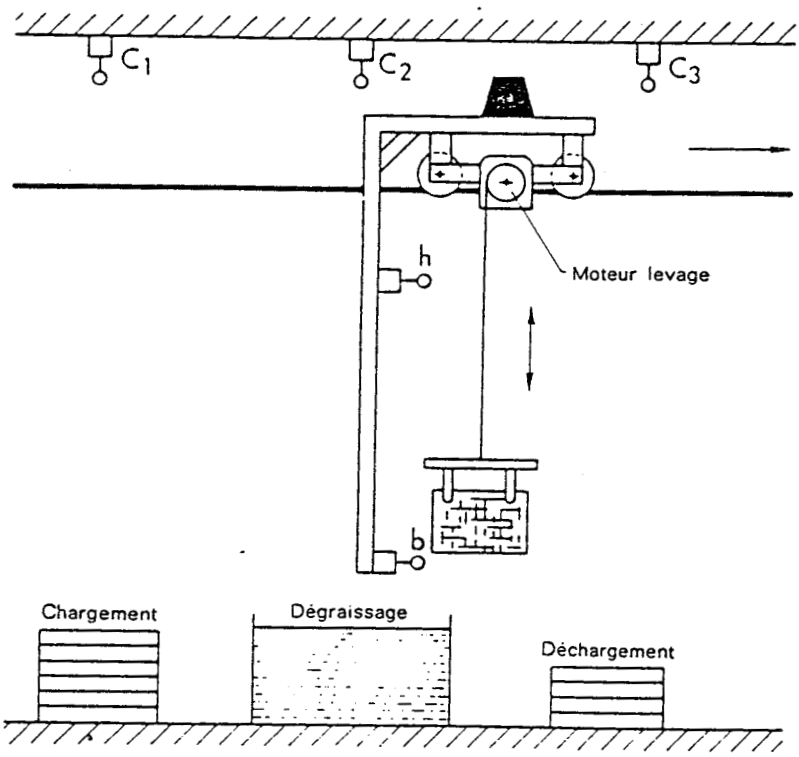


figure 2.10

Sur ordre de départ, à condition que le chariot se trouve en position C1 et le panier en position haute, les actions suivantes s'effectuent.

- -Déplacement du chariot jusqu'à ce qu'il soit au dessus de la cuve (C2);
- -Descente du panier jusqu'à la position basse;
- -Trempage pendant 30 secondes;
- -Remontée du panier en position haute;
- -Avance du chariot jusqu'à la position de déchargement (C3);
- -Le retour au poste de chargement se fait sur ordre de l'opérateur.

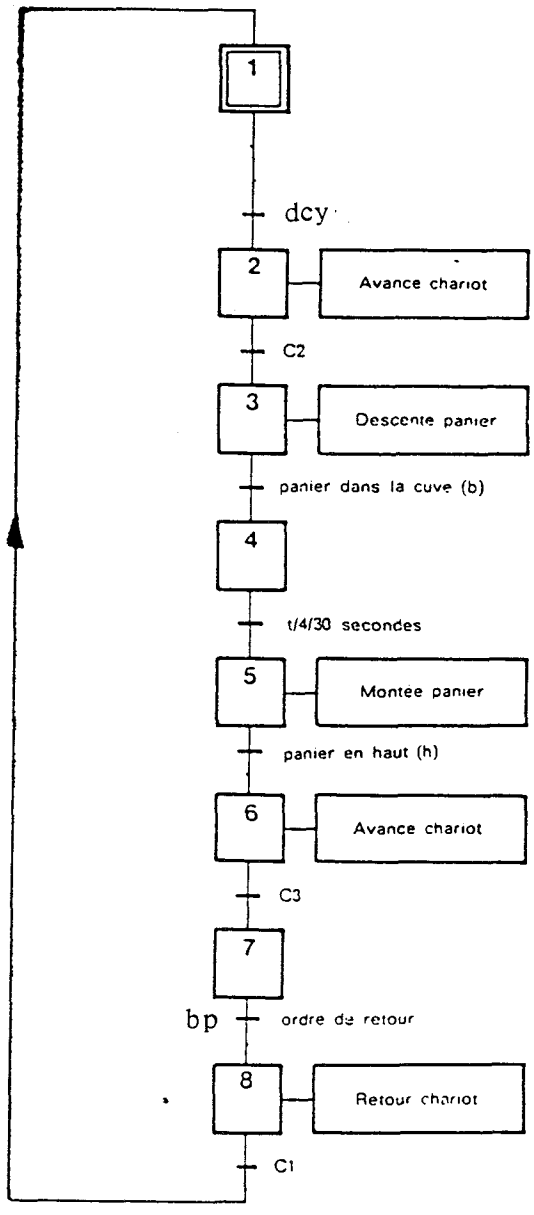


figure 2.11

Le grafcet correspondant à ce cahier des charges est représenté figure 2.11

La décomposition est donnée figure 2.12. Dans cet exemple, aucun dispositif de commande n'est développé. Nous distinguons les objets missions "mission chariot" et "mission monte charge", ils sont écrits spécialement pour cette application. Les objets "mission chariot" et "mission monte charge" communiquent d'une part entre eux et d'autre part avec les objets technologiques "bouton poussoir", "dcy", "chariot" et "monte charge". Ces deux derniers objets sont instances d'une même classe décrite dans la nomenclature. De façon duale, il y correspond des objets opératifs instances de la même classe "déplacement linéaire". Les objets technologiques "bouton poussoir" et "dcy"

dialoguent avec le même objet opératif "opérateur" susceptible de modéliser les actions de l'opérateur de conduite.

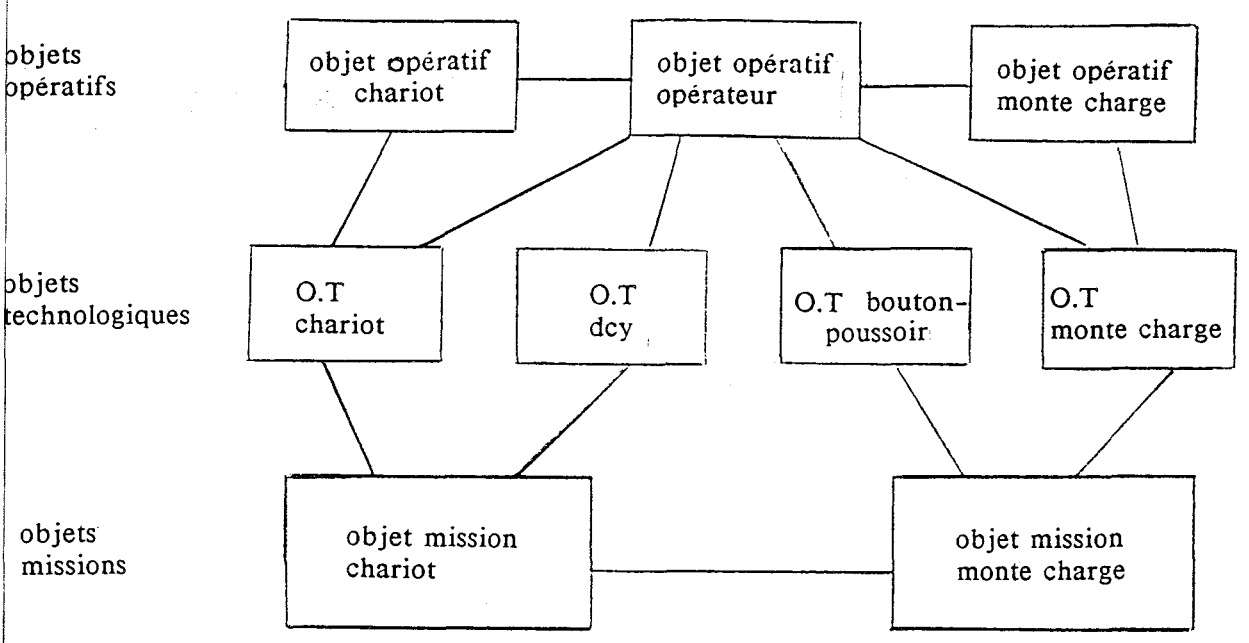


figure 2.12

deuxième exemple: [BAU 86]

Un malaxeur M reçoit des produits A et B pesés par la bascule C et des briquettes solubles, amenées une par une par un tapis d'amenage. L'automatisme permet de réaliser un mélange comportant les trois produits (figure 2.13).

Le cycle de fonctionnement est le suivant:

-L'action sur le bouton départ cycle "dcy" provoque le pesage et l'amenage des produits, éventuellement en simultanéité,

-pesage du produit A jusqu'au repère a, puis pesage du produit B jusqu'au repère b, suivis de la vidange de la bascule C dans le malaxeur,

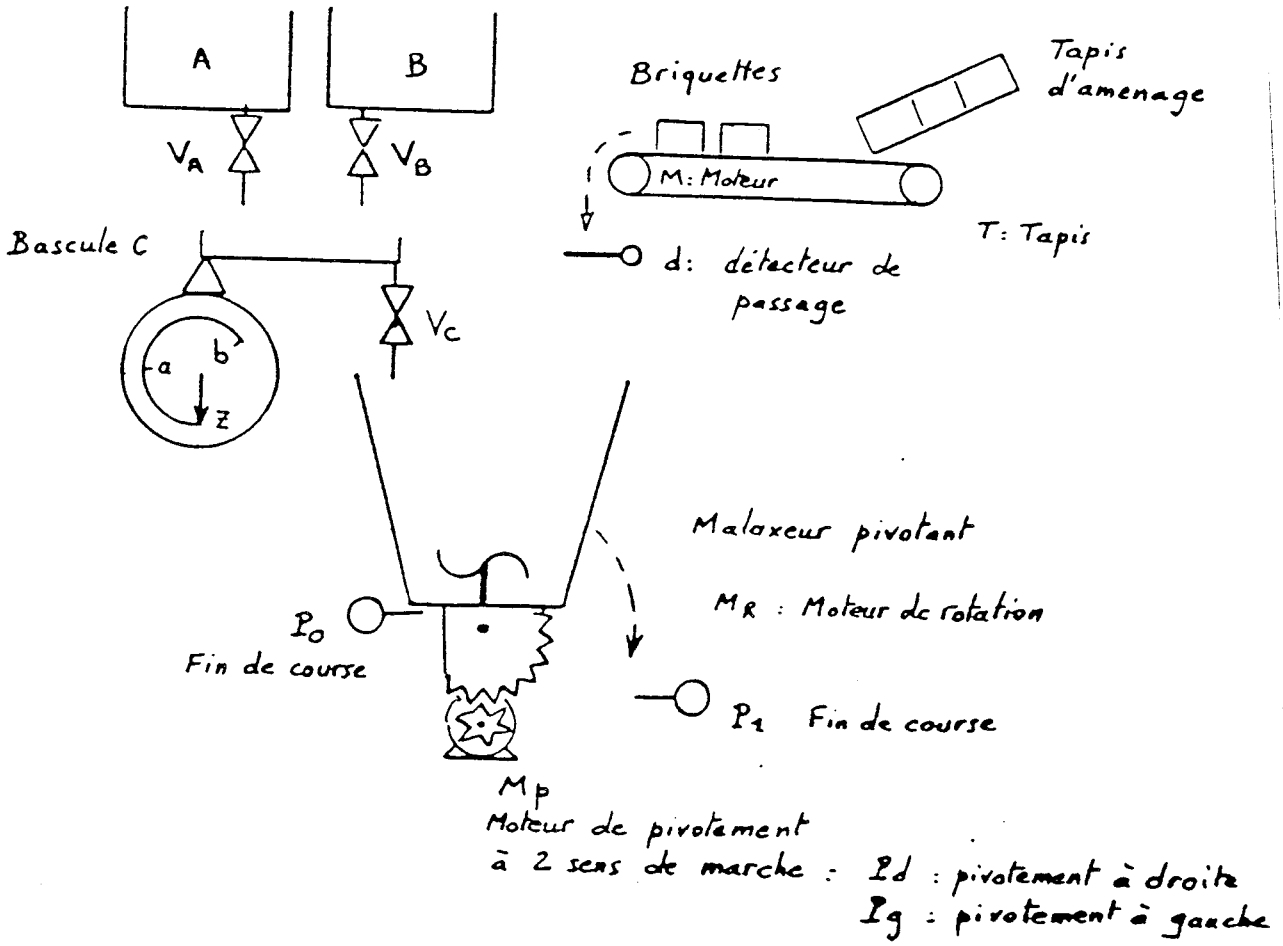


figure 2.13

contrainte: le mélange A+B, une fois fait, ne reste stable que pendant un temps TL1. Le début de la vidange de la bascule C doit donc se produire avant cette limite.

-amenage de N briquettes,

contrainte: le remplissage des briquettes ne peut pas se faire avant l'amenage du mélange A+B. Il peut se faire en même temps.

-Le cycle se poursuit par la rotation du malaxeur et par son pivotement, la rotation du malaxeur étant maintenue pendant la vidange. le début du pivotement ne peut se faire qu'après un intervalle de temps t à partir du début de rotation du malaxeur TL2 $\leq$ t $\leq$ TL3.

-Le cycle se termine par le retour du malaxeur en P0.

Nous définissons le grafcet correspondant (figure 2.14) en utilisant des éléments de synchronisation de façon à obtenir une solution optimale en jouant sur le temps masqué compte-tenu des contraintes.

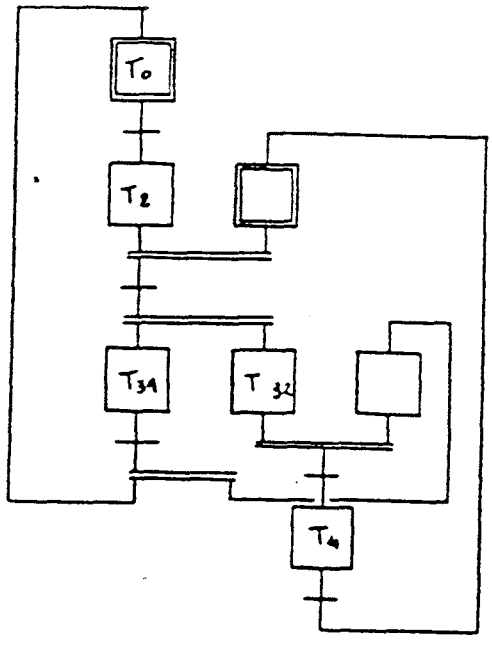


figure 2.14

Les différentes opérations sont:

T2: Préparation des liquides, qui comprend:

amenage et dosage de A,
amenage et dosage de B,
stockage de A+B.

T3: Remplissage du malaxeur, avec:

T31 amenage de A+B,
T32 amenage et comptage des briquettes.

T4: Malaxage décomposable en:

T41 malaxage,
T42 évacuation.

La décomposition en représentation de système d'objets est donnée figure 2.15. Nous utilisons ici un dispositif de commande supérieur, qui se projette sur les objets missions "mission malaxeur" et "mission chargt bascule", qui est un objet mission élémentaire. L'objet mission "mission malaxeur" distribue les différentes tâches aux quatre objets missions élémentaires "mission chargtAB", "mission chargtBr", "mission moteur", "mission vidange". Les objets "mission chargAB" et "mission chargBr" ne peuvent pas être confondus en un seul objet mission, car leur mission est simultanée et indépendante. L'objet opératif "op malaxeur" est le coordonnateur des trois objets opératifs "op chargt", "op moteur", "op vidange". Cet objet est indispensable pour le calcul du niveau de produit dans le malaxeur en fonction de celui de la bascule donné par "op bascule", des objets opératifs "op chargt" et "op vidange". Il peut également communiquer ce niveau à l'objet opératif "op moteur" pour qu'il puisse calculer sa vitesse de rotation en fonction de la charge.

Cette décomposition n'est pas unique, il aurait été possible d'envisager une communication entre les objets "mission malaxeur" et "mission charg bascule", une simple communication entre ces objets aurait été suffisante dans notre cas.

Il existe un objet opératif "opérateur", capable de dialoguer avec les objets opératifs et technologiques, et dont le rôle est de représenter l'introduction, par un pilote de conduite, de consignes de fonctionnement ou de forçages sur les objets. Le rôle de cet objet sera développé au cours d'un chapitre ultérieur.

Nous montrons la similitude entre la décomposition objet du système et de celle

obtenue à partir de SADT donnée à la figure 2.16. Nous pouvons faire le rapprochement entre les différentes boîtes de l'actigramme et les objets opératifs. Nous voyons que ces deux représentations présentent la même hiérarchie entre éléments.

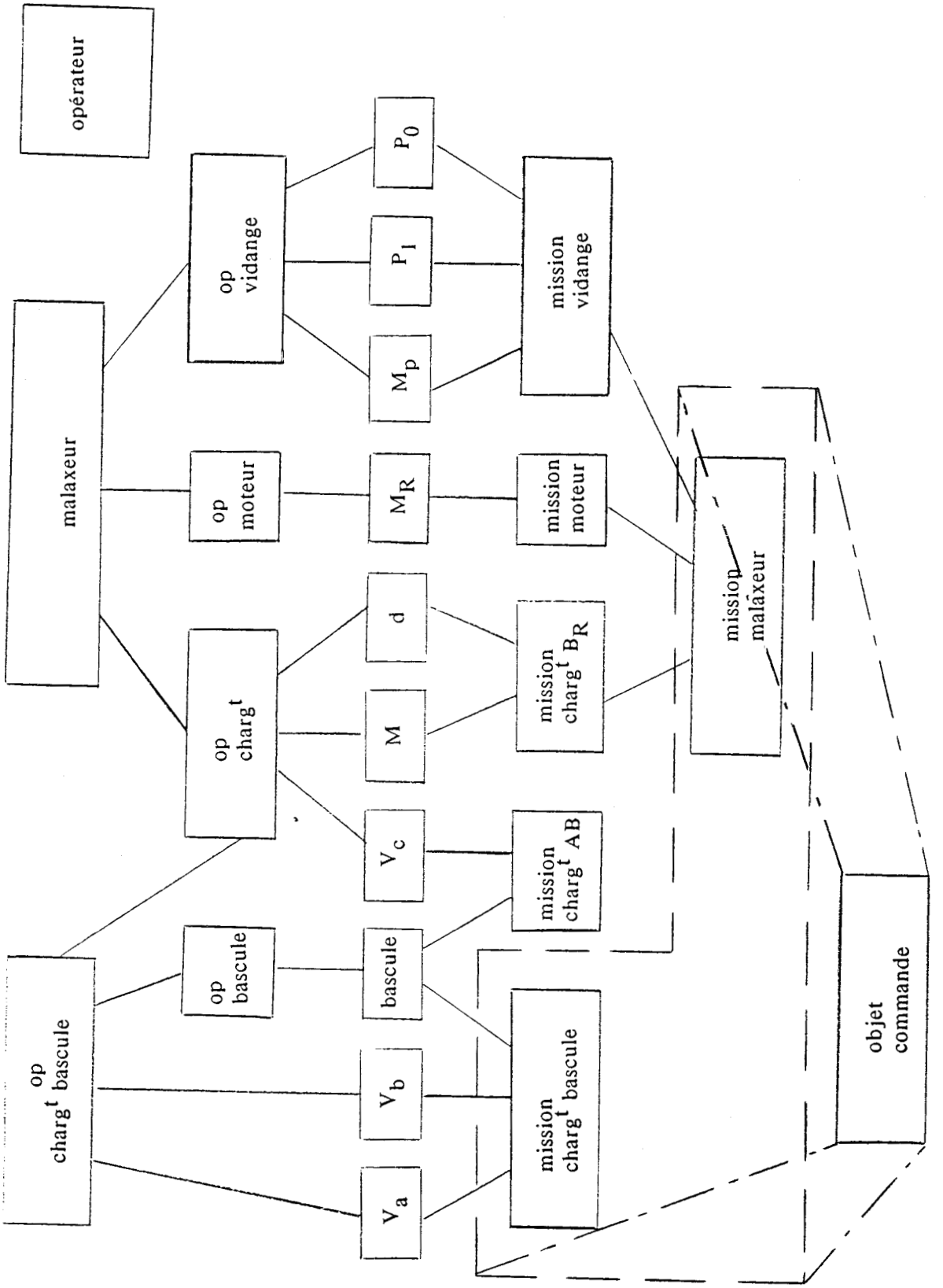

opérateur
malaxeur
op charg^t bascule
op vidange
op moteur
op charg^t
op bascule
V_a
V_b
bascule
V_c
M
d
M_R
M_p
P_1
P_0
mission charg^t bascule
mission charg^t AB
mission charg^t B_R
mission moteur
mission vidange
mission malaxeur
objet commande

figure 2.15

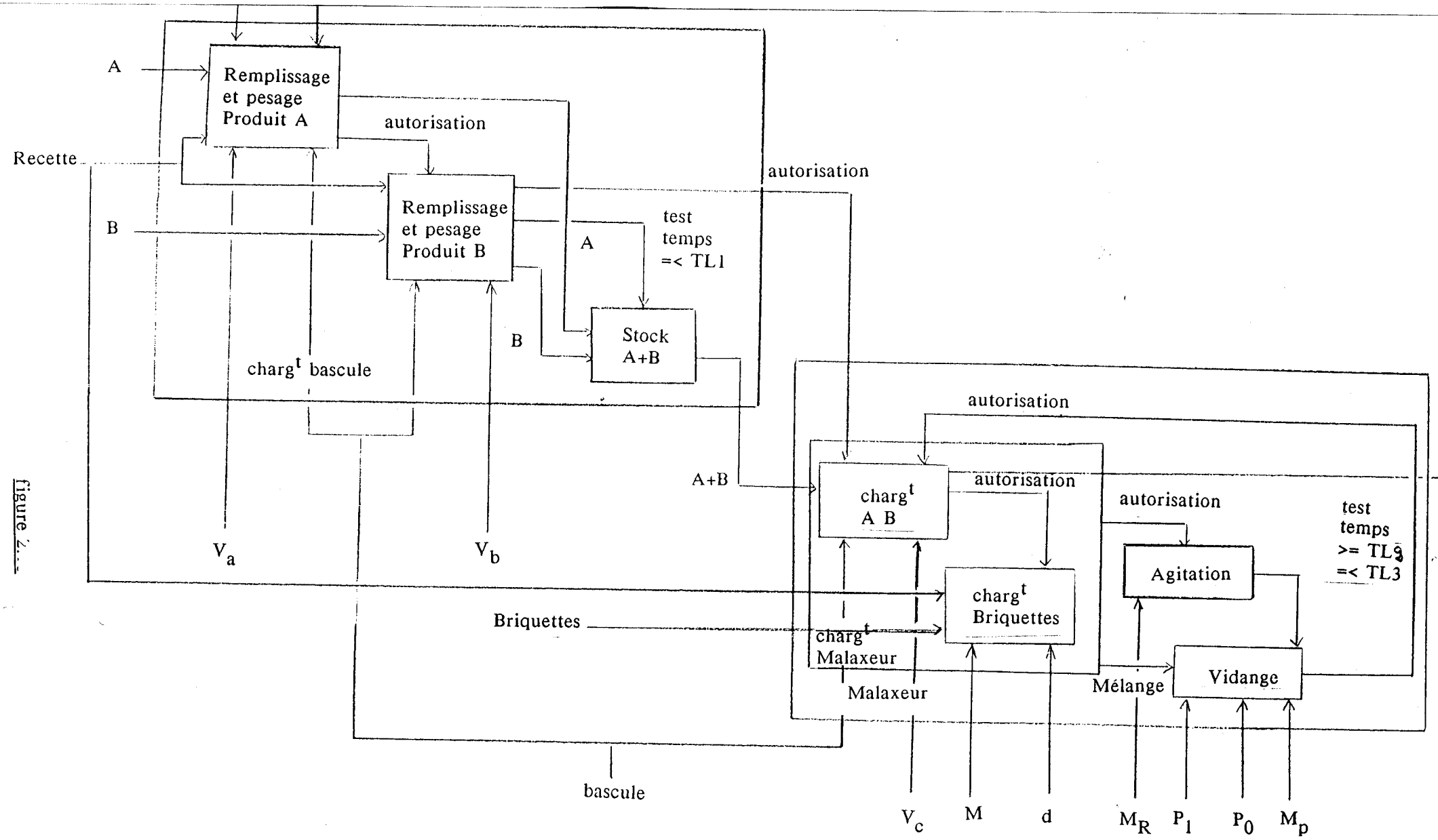


figure 2...

actigramme

CONCLUSION

Dans ce chapitre, nous avons donné une décomposition orientée objets d'un système industriel. Cette décomposition comporte, des objets opératifs dont le rôle est de représenter le comportement de la matière d'oeuvre, des objets missions contrôlant les différentes missions du processus, les interactions entre ces objets missions et opératifs s'effectue par les objets technologiques. L'objet commande permet de décrire les spécifications d'une commande centralisée, répartie, hiérarchisée.

Une des principales caractéristiques des éléments de cette décomposition est leur généricité. Un modèle générique, décrit une seule fois, peut être utilisé de façon multiple, à condition d'en faire l'instanciation à chaque utilisation. Ce caractère a guidé notre choix vers la programmation d'objet pour notre modélisation.

CHAPITRE 3

Programmation objet vers la simulation

INTRODUCTION

Au cours du précédent chapitre, nous avons vu que la décomposition d'un système, nous conduit naturellement à considérer une description orientée objets.

Nous allons maintenant montrer que les concepts de certains langages coïncident parfaitement avec nos définitions. Parmi les nombreuses possibilités qui s'offrent à nous pour réaliser les objectifs de simulation et de prototypage des processus par lots, nous avons choisi le langage Smalltalk, pour sa souplesse d'emploi qui rend toute modification d'un objet immédiatement effective, et parce qu'il donne accès à un environnement de programmation particulièrement performant.

3.1 LES LANGAGES D'OBJET

[MEY 80] [BOO 86] [IEE 84] [WIE 84] [COX 86] [SCH 86] [COO 86] [PAS 86] [DUF 86] [BIG 86] [ECO 87]

Le développement de la programmation structurée fit progresser la méthodologie de programmation. Cette nouvelle technique fournissait une façon systématique d'organiser les programmes de grande complexité, mais n'assurait pas la fiabilité et la maintenance des systèmes. La structuration d'un système reste à l'appréciation du concepteur, c'est pourquoi des difficultés surviennent à la fusion de plusieurs programmes. De plus un sous-programme n'est pas obligatoirement réutilisable.

La programmation objet apporte des concepts nouveaux vis à vis de la méthodologie de programmation. Elle fournit une structure aussi précise que celle du domaine exploité, c'est à dire que les objets représentent les comportements de l'environnement étudié: par exemple le domaine "banquier" contient des clients, des employés de banque, des coffres, des comptes,..., le domaine "graphique" contient des points, des lignes, des cercles,...

L'approche orientée objet d'un système consiste à identifier les objets du domaine concerné, leurs relations et leurs propriétés. Cette technique assure une réutilisabilité des objets. Le fait, que ces derniers soient indépendants, permet un développement de la programmation par plusieurs individus et une maintenance aisée.

La représentation informatique, d'un domaine évoluant continuellement, devient intéressante dans la mesure où celle-ci accepte les évolutions ou les modifications sans remettre en cause la validité de ce système.

Les techniques de programmation conventionnelles n'ont pas cette propriété. Bien qu'initialement un programme soit correct, au fur et à mesure des modifications il devient de moins en moins fiable. Pour pallier ces inconvénients d'évolution, de réutilisabilité et de collaboration de développement, les concepteurs de langages ont conçu diverses techniques pour contrôler les programmes telles que le type abstrait de données, l'héritage, le modèle générique.

Ces concepts ont donné naissance à quatre grandes familles de langages.

3.1.1 Les types abstraits

Comme la programmation structurée la méthodologie de "type abstrait de données" accentue le caractère local. L'objectif est d'associer variables et procédures au sein d'un même module. Une procédure ne peut manipuler que les variables locales du module dont elle dépend. Les modifications internes à un module n'ont aucune incidence à l'extérieur de celui-ci.

Ces modules sont nommés communément "types abstrait de données" et seuls les éléments, ayant comme modèle générique un type, peuvent avoir accès à ses procédures. Ces éléments sont dits "instances" du type.

Soit un exemple [POU] [POU 86], tiré d'une extension du langage Forth, "Object Oriented Forth", où figure un type "point", représentant un point à partir de nombres entiers, qui sont l'abscisse et l'ordonnée.

```
TYPE> POINT

2 VAR x                    ] déclaration des
2 VAR y                    ] variables

OPS>
: PT@ x @ y @;             ] déclaration des
: PT! x ! y !;             ] procédures

ENDTYPE> POINT
```

Les variables décrivent la représentation d'une instance, elles sont appelées "variables d'instance". Ces variables sont définies à la déclaration du type, et leur valeur est propre à chaque instance.

Les procédures représentent les circonstances potentielles dans lesquelles une instance du type est utilisée.

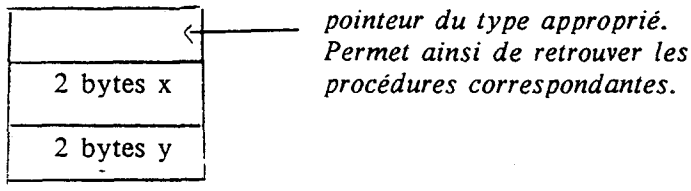
Lors d'une inspection du dictionnaire Forth seul le mot "point" apparait, les déclarations internes telles que x, y, PT@, PT! x sont invisibles. Ce phénomène s'appelle "<u>l'encapsulation</u>".

L'encapsulation est obtenue à la compilation, en modifiant les pointeurs, qui lient les mots du dictionnaire. Les pointeurs font désormais le lien entre les différents types.

```
reste du                 POINT   x   y   PT@   PT!              POINT
dictionnaire                                                      ↑
                                                                 pointeur
                                                                 du type
```

Soit POINT TOTO une création d'instance du type POINT. Une nouvelle variable de type POINT appelé TOTO est alors créé dans le dictionnaire

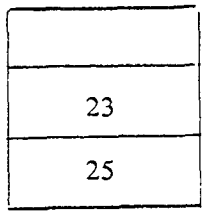
TOTO



Cette instance emploie les opérations COM@ et COM!, à l'aide de la syntaxe adéquate.

23 25 TOTO PT!
Les nombres 23 et 25 sont stockés dans TOTO comme abscisse et ordonnée.

TOTO



Le fait, que les opérations peuvent avoir des définitions multiples, est un des caractères essentiels des types abstraits de données. En effet le sens d'une opération est dépendant du type dans lequel elle est déclarée. Ce caractère est connu sous le nom de "polymorphisme".

Une variable d'instance, déclarée dans un nouveau type, peut être une instance, dont le type a été défini préalablement.

Soit l'exemple d'une définition d'un type, dont les variables d'instance sont des instances de POINT.

```
TYPE> RECTANGLE

POINT origine
POINT coin

OPS>
: RECT@ origine PT@ coin PT@;
    .
    .
    .
ENDTYPE> RECTANGLE
```

Les variables d'instance origine et coin emploient les opérations du type POINT et sont utilisées par les opérations du type RECTANGLE: Dans l'opération RECT@, les variables origine et coin sont manipulées par des opérations du type POINT.

Il est possible de réutiliser des opérations d'un type préalablement défini dans un nouveau type, sans les redéclarer. Ce nouveau type hérite alors de ces opérations.

Définissons un type "POINTDIM3", donnant les coordonnées et les opérations sur un point dans un espace de dimension 3, à l'aide des opérations du type POINT. Pour ce, nous employons le mot Forth "INCLUDE".

```
TYPE> POINTDIM3

2 VAR X
2 VAR Y
2 VAR Z

OPS> INCLUDE> POINT   <------  utilisation des opérations
: T! Z ! ;                     déclarées dans le type POINT
: T@ Z @ ;

ENDTYPE> POINTDIM3
```

Le langage C++ [STR 87] [STR] est une extension orientée objet du langage C. C++ fait appel à la méthodologie des types abstraits. Ceux-ci fournissent le caractère local des procédures associées aux variables locales.

Certains types prédéfinis existent en C++. A ces derniers le programmeur peut en adjoindre de nouveaux. Ils sont alors appelés "classe". Une classe et un type prédéfini diffèrent dans leur définition mais non dans leur utilisation.

La définition d'une classe comporte une partie spécification nécessaire à la représentation d'objet du type. Un groupe de procédures manipulent ces objets.

```
class date  {                                          spécifications
                                   SPECIFICATION       et procédures
     int   jour, mois, année;  <------------'          privées

Public:
     void set(int, int, int);
     void get(int*, int*, int*);   PROCEDURES          spécifications
     void next ();                                     et procédures
     void print ();                                    publiques
};
```

Dans notre exemple, la fonction set(int, int, int) permet d'initialiser un objet de la classe date. Mais, cette initialisation peut se faire dès la déclaration grâce à des fonctions nommées "constructeur" définies dans le type. Le nom de ces fonctions est le même que celui de la classe.

```
class date {

      int   jour, mois, année;
Public:
      date(int, int, int);                    // jour  mois  année
      date(char*);                            // date donnée en chaîne
                                                    de caractères
      //...
};
```

Nous montrons dans cet exemple qu'une classe peut avoir plusieurs fonctions "constructeur". A l'initialisation, le compilateur sélectionne le "constructeur" adapté. Il faut donc que ces fonctions se différencient par le nombre et le type de leurs arguments.

```
Ex: date today (20 12 1987)
      date Dec20 ("20 Décembre 1987")
```

D'une façon générale, au sein d'un même type plusieurs fonctions peuvent avoir le même nom, mais elles sont discriminées à la compilation par le nombre et le type de leur arguments.

Ceci impose une déclaration de type pour chaque argument des différentes fonctions. Nous voyons sur l'exemple de la classe complexe ci-dessous qu'il est nécessaire de déclarer plusieurs fonctions pour l'opérateur +. La première solution réalise la somme entre deux nombres complexes, la deuxième entre un nombre complexe et entier double précision, la dernière entre un entier double précision et un nombre complexe.

```
class complexe {

      double  re, im;

Public:
      complex(double r, double i)      {re=r;  im=i; }
      friend complex operator+ (complex, complex);
      friend complex operator+ (complex, double);
      friend complex operator+ (double, complex);
      //...
};
```

Afin de réduire le nombre de fonctions, il est intéressant d'introduire la convertion implicite entre types utilisateurs.
Reprenons l'exemple précédent:

```
class complexe  {

        double re, im;

Public:
        complex(double r, double i=0);  {re=r; im=i; }
        friend complex operator+(complexe, complexe);
        //...
};
```

Dans la fonction "constructor", la partie imaginaire du complexe a pour valeur par défaut zéro. Il n'est donc plus nécessaire de différencier les types d'arguments complexe et entier dans les fonctions, puisqu'un entier est un comlexe particulier.

C++ permet les déclarations hiérarchiques de classe par héritage simple ou multiple.

La notion d'héritage simple est liée à celle de classe dérivée. Une instance d'une classe dérivée a pour spécifications, d'une part celles déclarées dans la classe dérivée, d'autre part celles de la classe mère.

Si le mot "Public" n'est pas indiqué comme argument d'héritage, seules les instances de la classe dérivée auront accès aux procédures publiques de la classe mère. Dans le cas contraire, l'accès aux procédures publiques des deux classes est garanti.

L'accès aux spécifications et aux procédures publiques ou privées d'une classe est rendu possible par l'héritage multiple. Les classes déclarées sous le mot clé "friend" bénéficient de cet héritage.

Le dialogue entre les objets se fait par envoi de message dans lequel est précisé le destinataire, le sélecteur(appel d'une procédure), les arguments de traitement.

A un même message, transmis à des destinataires différents, ne correspond pas obligatoirement un traitement identique.

3.1.2 Les langages de Frames [DES 86]

Dans les langages de frames, les variables et les procédures d'un type sont des attributs ou "slots", auxquels est associée une valeur.
Au concept d'héritage s'adjoint une notion supplémentaire, la notion de multiplicité. Une classe ou type peut hériter des attributs de plusieurs types.

Prenons un exemple tiré du langage LRO. La notion d'héritage est définie par le mot clé "sorte-de". La classe C_1 possède à la fois ses propres attributs P_1, P_2... ,P_n mais aussi les attributs des classes C_2, C_3 et ceux des classes dont C_2 et C_3 héritent.

(classe C_1 (sorte-de C_2 C_3)
P_1
P_2
.
.
.
P_n)

Dans les langages de frames, la classe permet de regrouper des objets semblables. Tous les objets d'une classe donnée possèdent les mêmes attributs, et ne diffèrent que par leurs valeurs.

Entre objets, la notion d'héritage a aussi un sens. Elle permet à un objet d'une classe donnée de "pointer" sur un objet de classe différente et d'hériter ainsi de ses attributs et de leurs valeurs. Cet héritage entre objets est également récursif et multiple.

Les différents héritages entre classes d'une part, et entre objets d'autre part peuvent être représentés par des graphes(figure 3.1).

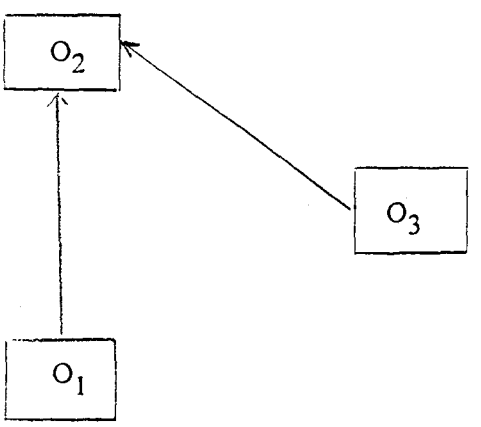


figure 3.1

Admettons que la figure 3.1 représente le graphe d'héritage entre les objets O_1, O_2 et O_3, le graphe d'héritage entre classes serait identique.
Les objets O_1 et O_3 héritent des attributs de O_2.

Cette technique d'héritage demande quelques précautions. En effet, ajouter un objet ou un lien d'héritage à un objet existant oblige parfois à supprimer des liens qui deviennent redondants.

Reprenons l'exemple précédent, auquel nous ajoutons un lien entre l'objet O_1 et O_3(figure 3.2).

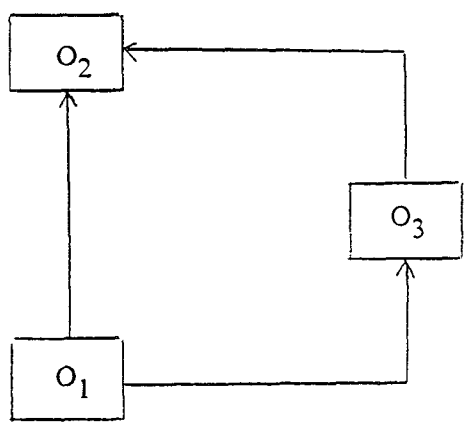


figure 3.2

Nous voyons que le lien entre O_1 et O_2 devient redondant et par conséquent, il est indispensable de le supprimer(figure 3.3).

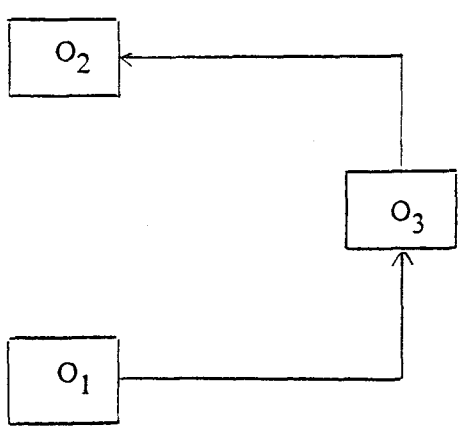


figure 3.3

Nous pouvons imaginer aussi le cas où nous supprimons un lien, ou un objet possédant des liens(figure 3.4).

Deux cas extrêmes sont envisageables, soit les liens entre objets sont conservés malgré la suppression de l'un d'entre eux, soit les liens entre les objets sont définitivement perdus.

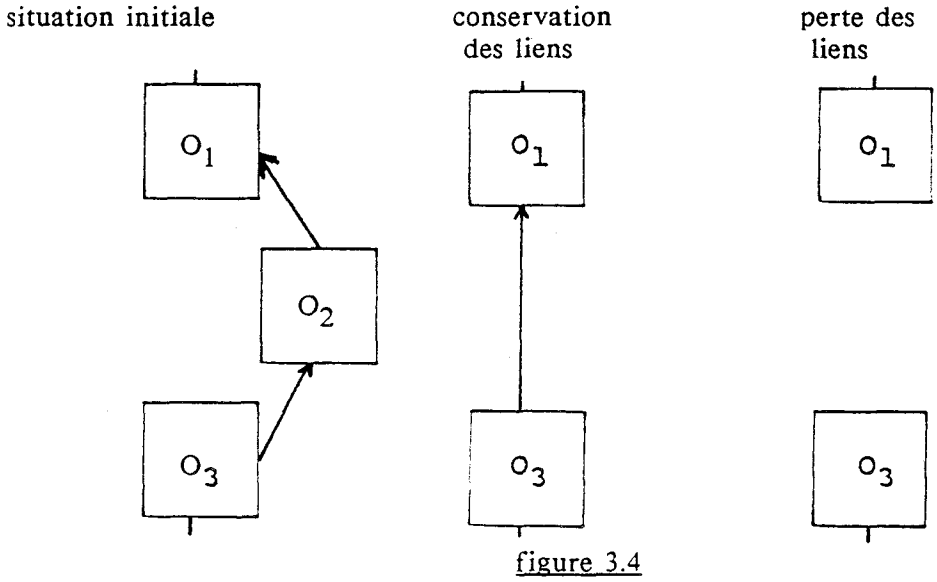


figure 3.4

Différentes techniques d'héritages sont possibles, le balayage en profondeur, en largeur ou éventuellement les deux. Dans ce dernier cas, la technique est déterminée dans le type (le_cool) [coo]

3.1.3 Les langages acteurs [LIE 81]

Dans les langages acteurs, la notion de "type-instance" n'existe plus, seule subsiste la notion d'objet. Les acteurs communiquent par message. La délégation permet de créer de nouveaux acteurs partageant leur comportement avec les acteurs déjà existants. Dans ces langages tout est acteur, donc en particulier les messages, les procédures, les variables. Une variable est un acteur capable de répondre au message lui demandant sa valeur. Quand un acteur reçoit un message, il donne la possibilité à chacune de ses procédures (méthodes) de répondre, par délégation.

Les acteurs délégués connaissent l'acteur à qui s'adresse le message. Un point est un acteur. Pour ne pas à avoir à redéfinir toutes les possibilités d'un tel acteur pour chaque point de l'écran, il est indispensable de créer un acteur représentant le comportement général d'un point. L'acteur désignant un point particulier possède les

valeurs des coordonnées qui lui sont propres et d'autres variables le distinguant des autres points.

L'acteur, représentant le comportement général d'un point, possède des variables et des procédures propres à son usage, et d'autres utilisées par des "acteurs extensions" les points en particulier. Les "acteurs extensions" se partagent donc l'usage de ces dernières.

Nous montrons à la figure 3.5 un schéma de la notion de délégation.

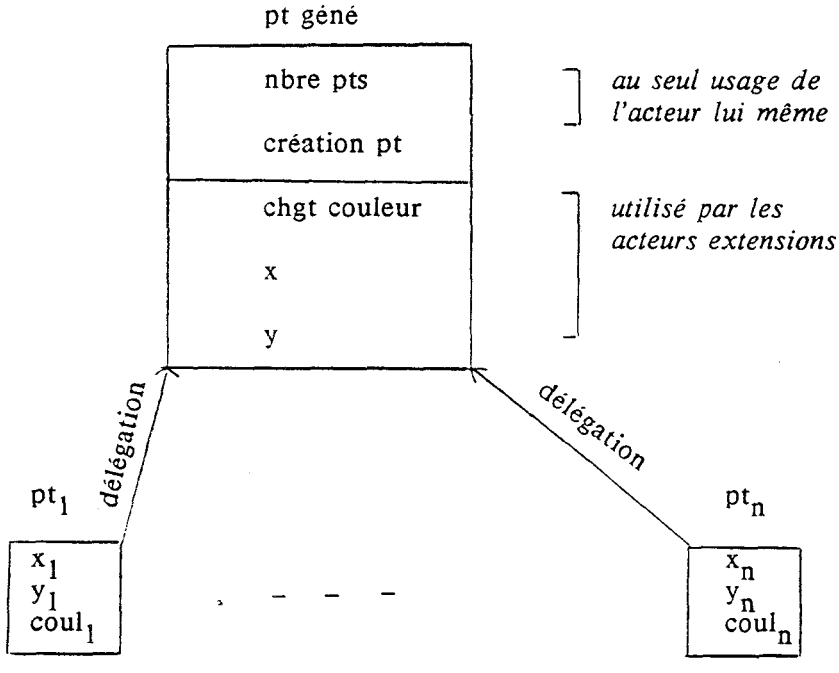


figure 3.5

Un acteur extension possède des variables, des procédures et une liste d'acteurs. Ces derniers se substituent à l'acteur extension, si celui-ci n'est pas capable d'effectuer l'action demandée.

Envisageons, maintenant, un point dans un espace de dimension trois. Il faut alors créer un acteur extension point3D de l'acteur point, que nous avions considéré dans un espace de dimension deux. Cet acteur, point3D, possède la variable et les procédures relatives à la troisième dimension, et une liste d'acteurs contenant le seul acteur point.

Nous pouvons envisager aussi un acteur extension pointCouleur qui est une extension de l'acteur point. L'acteur extension possède les variables, les procédures qui lui sont propres, et une liste d'acteurs contenant l'acteur point.

Il est possible de créer un nouvel acteur point3DCouleur, c'est une extension des acteurs point3D et pointCouleur. Ces deux derniers se substituent à l'acteur point3DCouleur lorsque celui-ci n'est pas capable de répondre à certains messages.

Sur cet exemple, les comportements des acteurs point3D et pointCouleur sont relativement indépendants, mais dans certains cas il peut subsister des problèmes d'incompatibilité entre les différents acteurs se substituant à l'acteur extension. Afin de résoudre ces conflits, la délégation est obtenue par le premier acteur de la liste capable de répondre au message.

3.2 SMALLTALK

[GOL 83] [GOL 84] [KRA 83] [DIG 86] [COI 82] [KAE 86]

Smalltalk a été créé par Xerox à Palto Alto et développé parallèlement par:

- Dec
- Tektronix
- Apple
- L'université de Burkeley

L'objectif de Smalltalk est d'orienter la programmation vers le dialogue. Ce langage est totalement intégré, il possède son propre système d'exploitation, bien que des versions nouvelles existent sous un système d'exploitation ordinaire (UNIX, MS-DOS).

3.2.1 Concepts

Le langage Smalltalk repose sur les notions d'objet, de classe, d'héritage, de message.

L'objet est l'entité essentielle de ce langage. Il associe des variables et des méthodes. Les variables personnalisent l'objet, tandis que les éléments caractérisant son comportement sont contenus dans les procédures appelées méthodes.

objet	variables:	décrivent les états ou les connaissances sur l'objet
	méthodes:	déterminent les actions que l'objet peut exécuter

L'unique structure de contrôle est la transmission de messages entre les objets.

La notion de classe générique permet de regrouper des objets acceptant le même modèle de comportement. Une classe est décrite une fois pour toute, elle est utilisée par différentes instanciations. Tout objet est l'instance d'une classe.

En Smalltalk toute classe est l'instance d'une classe particulière appelée MétaClasse. Une métaclasse a le même comportement vis à vis des classes, que les classes vis à vis des objets. Une métaclasse est une instance de la classe MétaClasse, qui est une instance d'elle même.

Toutes les instances d'une même classe possèdent les mêmes variables, elles ne diffèrent que par leurs valeurs et utilisent les mêmes méthodes en réponse à un même message.

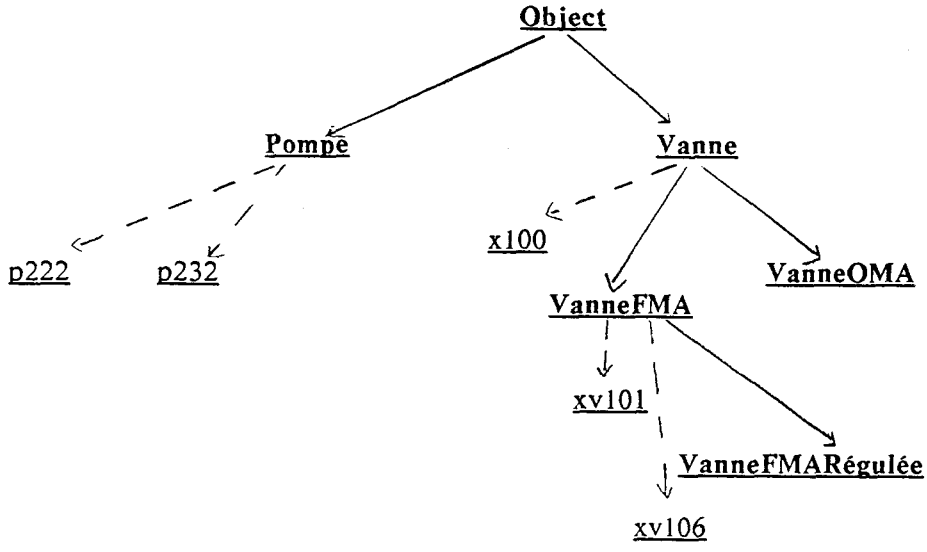
classe:	vanne	état débitMaximun	} *variables*
		ouverture fermeture	} *méthodes*
objet:	xv101	xv106	

Comme en Simula, il existe une hiérarchie entre classes. Une sous-classe hérite des variables et des méthodes de la hiérarchie correspondante. Elle possède également ses propres variables et méthodes.

A la réception d'un message, la recherche d'une variable ou d'une méthode à effectuer se fait en remontant la hiérarchie. Ceci peut engendrer un phénomène de masquage, si une méthode est déclarée dans plusieurs classes de la même hiérarchie.

La notion d'héritage écourte le temps de recherche existant dans les langages acteurs. En effet, si un acteur n'est pas capable de répondre à un message, il délègue ses pouvoirs à d'autres acteurs, celui capable de répondre doit être recherché. En contre partie, l'espace mémoire occupé par un objet est fonction de la hiérarchie des classes, puisque les objets possèdent toutes les variables de sa classe et de ses sur-classes. Cet espace peut devenir volumineux et entraîner un accès à une mémoire virtuelle ralentissant considérablement le système.

Deux notions sont liées à une classe, l'appartenance obtenue par instanciation et l'inclusion obtenue par héritage. (figure3.6)



Dans cette figure, les classes sont matérialisées en caractère gras.

———— héritage

- - - - instanciation

figure 3.6

En Smalltalk, il existe une classe racine appelée "object", toutes classes héritent directement ou indirectement de cette classe.

En parallèle à l'organisation des classes, il en existe une pour les méta-classes. La hiérarchie des méta-classes n'est pas la même que celle des classes (figure 3.7). Il faut préciser, que l'organisation des classes données est relative à Smalltalk/V

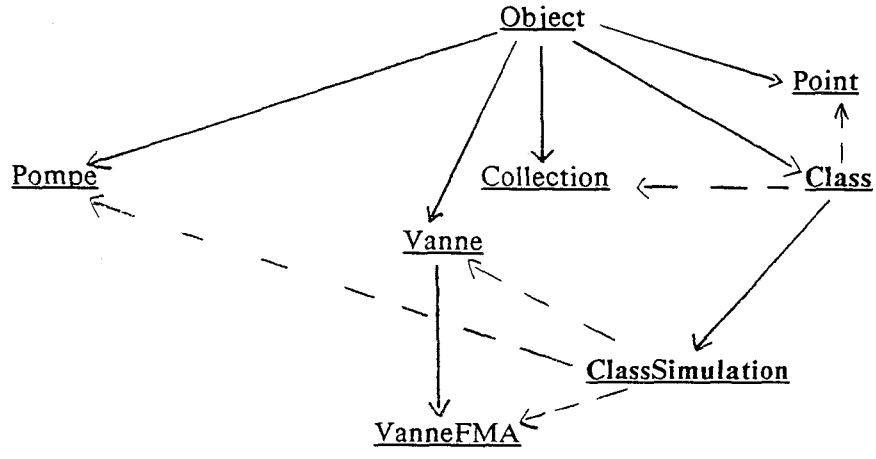
Sur cette figure, nous voyons que la classe Collection, par exemple, est une instance de la métaclasse Class et, que cette même classe Collection hérite des variables et des méthodes de la classe Objet.

Il existe donc deux arborescences distinctes:

celle des classes.

celle des méta-classes.

L'utilisateur a la possibilité de créer des objets, des classes et des méta-classes, au même titre que ceux existants dans le système.



Dans cette figure, les méta-classes sont matérialisées en caractère gras.
_____ héritage
- - - - instanciation

figure 3.7

3.2.2 Classe

Une classe est définie par:

- des variables de classe
- des méthodes de classe
- des variables d'intance
- des méthodes d'instance
- des variables globales de classes ou pool

variables de classe:

Ce sont des variables globales au sein de la classe, c'est à dire qu'elles sont communes à toutes les instances de la classe et de ses sous classes. Elles sont accessibles aux méthodes de classe et d'instance.

méthodes de classe

Ce sont des procédures, qui portent globalement, sur les instances, sur les variables de classe. Ces procédures peuvent être, des manipulations globales, des créations d'instances, des initialisations d'instances. Par exemple, si une variable de classe contient la liste des instances de cette classe, une manipulation de cette liste est faite par une méthode de classe.

variables d'instance

Leurs valeurs sont propres à chaque instance, et caractérisent chacun des représentants d'une classe. Elles ne sont visibles que de l'objet auquel elles appartiennent. Elles constituent la partie déclarative de l'objet.

méthodes d'instance

Ces méthodes constituent la partie fonctionnelle de l'objet, elles exécutent des procédures d'exploitation de la partie déclarative de l'objet. L'ensemble de ces méthodes décrit les comportements possibles d'une instance de la classe. Seules les instances permettent d'y accéder.

Leur structure se compose d'un sélecteur, éventuellement des variables de travail, et des expressions ou algorithmes.

La méthode, dont le sélecteur correspond à celui du message reçu, est exécutée.

Les variables de travail permettent, si besoin est, les mémorisations intermédiaires au sein de la méthode.

Les expressions et algorithmes restituent un comportement spécifique de l'objet et envoient des messages à d'autres objets.

variables globales de classe ou pool

Ce sont des variables partagées par des instances de différentes classes. Ces variables vont à l'encontre de la notion d' encapsulation et par conséquent sont à utiliser avec précaution, bien qu'elles puissent être pratiques dans certains cas.

3.2.3 Message

La seule façon d'accèder à un objet est l'envoi de message. La messagerie est bidirectionnelle, c'est à dire qu'un message envoyé à un récepteur, implique toujours une réponse.

Tout objet peut dialoguer avec un autre. Il est donc possible d'établir une communication entre objet et classe.

Un message est constitué du nom de l'objet récepteur, d'un sélecteur désignant le type de manipulation et éventuellement d'un ou plusieurs arguments.

RECEPTEUR	SELECTEUR	ARGUMENTS

Les éléments constitutifs d'un message peuvent être eux mêmes des messages.

L'objet récepteur est l'objet qui, à la réception du message, exécute une action et répond à ce message.

Dans le message: 3 + 2

3 est le récepteur, instance de la classe des entiers, à qui le sélecteur + est envoyé. Ce sélecteur correspond à celui de la méthode instance + de la classe du récepteur. 2 est l'argument attendu par la méthode.

En Smalltalk, il existe trois types de messages:

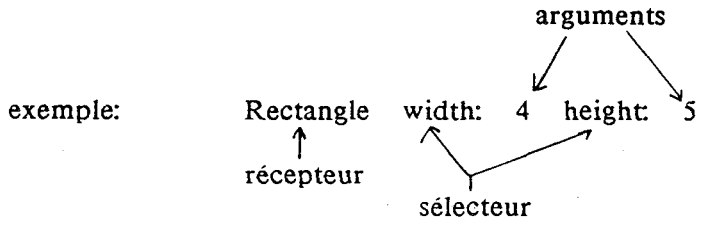
- Les messages unaires qui ne comportent pas d'argument.

exemple: rectangle ↑ récepteur center ↑ sélecteur

- Les messages binaires sont des messages à un argument dont le sélecteur est composé d'un ou deux caractères, qui ne soient ni des chiffres, ni des lettres. On y trouve en particulier les opérateurs arithmétiques.

exemple: 3 + 2

- les messages à mots clés peuvent avoir plusieurs arguments.



Le nom du sélecteur est width: heigh:.

Un message se lit de gauche à droite, avec priorité d'abord aux sélecteurs unaires puis aux sélecteurs binaires. Seul le parenthésage peut modifier l'ordre de priorité.

Le point est un séparateur de message tandis que le point-virgule permet d'envoyer des messages en cascade à un même récepteur
exemple: pen home; up; turn: 90; go: 110.

Par ailleurs, il existe deux pseudo-variables importantes. L'une "self" permet à un objet de s'envoyer un message. L'autre "super" donne accès aux méthodes des classes hiérarchiquement supérieures, en évitant ainsi les phénomènes de masquage.

3.2.4 Bloc

Dans une méthode, il est possible de définir un bloc, qui est un objet représentant une séquence d'actions exécutable en différé. Les messages d'un bloc sont enfermés entre crochets. L'exécution de celui-ci se fait à la réception du message "value".

exemple: [somme:= somme + 1] value.

Comme les messages, certains blocs possèdent des arguments, ils reçoivent donc des messages avec arguments. Le nombre d'arguments du message doit correspondre au nombre d'arguments défini dans le bloc.

exemple: [:x:y| somme:= somme + x + y] value: 3 value: 4

arguments du bloc

Cette notion de bloc permet, d'effectuer des actions sur les éléments d'une collection, d'établir l'équivalent des structures de contrôles tel que:

ifTrue: ifFalse:
whileTrue: whileFalse:
timesRepeat:
.
.
.

La dynamique des pointeurs, appelé "late binding", constitue un des atouts de Smalltalk, contrairement à certains langages objets "early binding" où les pointeurs sont statiques et interdisant par conséquent toute modification ultérieure.

La liaison dynamique présente l'avantage d'interpréter la méthode compilée à la réception du message. Ceci entraîne une indépendance des objets, puisque la modification de l'un n'implique aucune perturbation sur les autres. Seule la réponse au message est modifiée. L'opérateur peut donc effectuer des changements sur les objets à tout instant, ce qui est impossible en liaison statique.

Puisque toute référence est symbolique, une méthode peut donc être recompilée sans avoir à recompilér les objets qui l'appellent.

En contre partie, la recherche de la méthode appropriée au message reçu est faite à l'exécution, ceci ralenti considérablement le système et ne permet le contrôle qu'à l'exécution.

Les méthodes, à leur définition dans une classe, sont compilées en bytes-code, qui ne sont interprétés qu'à l'exécution. Si cette technique affecte quelque peu la vitesse d'exécution, elle assure une certaine portabilité de Smalltalk.

Un des aspects essentiels de Smalltalk est la convivialité obtenue par le multi-fenêtrage, les "popUpMenu" et le découpage de chaque fenêtre en différents panneaux.

Une fenêtre restitue des informations textuelles ou graphiques concernant un objet. Il existe dans ce cas, une dépendance entre l'objet et sa représentation. L'utilisateur dialogue avec cet objet principalement par l'intermédiaire de la souris, en sélectionnant et en utilisant le menu relatif au panneau.

Les changements apportés à l'objet, par une intervention de l'utilisateur, sont notifiés à la vue qui réagit en conséquence, bien souvent en affichant l'objet modifié. Il est possible de construire une représentation structurée composée de sous-vues dépendantes. Chaque sous-vue présente un aspect de l'objet visualisé. Dans ce cas, une modification de l'objet dans l'une des sous-vues entraîne les mises à jour adéquates aux autres sous-vues.

Smalltalk permet l'accés aux composants du langage, ceci se fait à l'aide, d'une fenêtre "inspecteur" donnant accés aux variables d'instance d'un objet, d'une fenêtre "browser" renvoyant le contenu des classes, d'une fenêtre "debugger" donnant le contexte d'exécution et d'expédition de messages.

3.3 OBJECTIFS D'UNE SIMULATION

La maîtrise du comportement des systèmes industriels complexes, et en particulier des processus par lots tant au niveau prédictif qu'au niveau opérationnel dépend de la modélisation du comportement des composants agissant sur la matière d'oeuvre et de leur interactions. Il est donc nécessaire de définir, des états supports des comportements fonctionnels pour une modélisation statique, des événements générateurs de changement d'état pour une modélisation dynamique.

Il faut utiliser la connaissance du fonctionnement du système. L'objectif est l'élaboration d'une modélisation du comportement de systèmes industriels permettant de réaliser des études prévisionnelles d'une part et d'autre part d'assurer une aide au suivi opérationel.

Pour cela, il est indispensable de faire ressortir les objectifs d'un système industriel de nature:

- fonctionnelle, ce sont les missions du système.

- d'exploitation, englobant les notions de sécurité des personnes et des biens, de disponibilité, d'optimisation des coûts et du temps, des ressources et des opérations de maintenance.

Les techniques de modélisation classiques sont trop synthétiques. Toutes les tentatives algorithmiques de génération automatique de ces modèles ont avorté ou se révèlent insatisfaisantes.

La simulation envisagée est événementielle. Son but essentiel est de définir précisément l'évolution du processus relativement aux défauts des objets technologiques ou de la partie opérative. De cette façon, concepteurs et opérateurs de conduite prennent conscience des situations non désirables, ils le font sans se soucier des problèmes d'insécurité, de non contrôle, de non fonctionnement du système.

Cet objectif s'obtient à des différents niveaux.

Le premier niveau correspond au contrôle du mode opératoire, l'utilisateur se limite à une vérification de l'enchaînement du mode opératoire. L'opérateur ne fait intervenir aucune anomalie des automatismes, il se contente de contrôler visuellement l'enchaînement des opérations.

Il est intéressant à un autre niveau de tester la commande, il faut s'assurer que des verrouillages existent vis à vis des occurrences dangereuses de la partie opérative. La commande doit être mise à contribution. Ceci s'obtient en effectuant des forçages sur les objets technologiques, nous pouvons forcer une vanne à rester ouverte. Des perturbations, provoquées sur les objets opératifs, modifient l'évolution du processus et permettent par conséquent de tester la commande. Ces types de forçage créent des configurations anormales, des anomalies de conception. Ils mettent en évidence les protections prévues dans la commande face aux occurrences encourues. Ceci apporte une aide aux concepteur.

Le dernier niveau favorise le contrôle de l'incidence de défaillances d'objets, ce contrôle s'effectue tant au niveau des objets technologiques qu'au niveau des objets opératifs. Il permet de prendre conscience de l'importance d'une défaillance d'un objet sur le déroulement de processus. Il évalue également les répercussions des défaillances des objets non pilotés par la commande.

Un autre but essentiel de cette simulation est d'enseigner aux opérateurs la conduite d'un processus. Les défauts surviennent stochastiquement à l'insu de l'opérateur. Ce dernier peut tester ses réactions face aux différents événements, il se trouve confronté aux problèmes rencontrés sur le site.

La prise en considération des réactions d'un opérateur est un objectif supplémentaire développé dans notre modélisation du processus. Nous pouvons ainsi établir une base experte obtenue par la collaboration de pilotes confirmés, de techniciens de maintenance, de concepteurs... Il faut que les experts soient capables de formaliser leurs connaissances, l'intérêt de cet objectif en dépend. En effet, cette base experte est destinée à enseigner aux apprentis pilotes les réactions les mieux adaptées aux situations déjà vécues. Il est évident, que le soin apporté au développement des bases de connaissances influe sur la validité de cet objectif.

Pour satisfaire ces objectifs, il est nécessaire d'établir une communication aisée entre la machine et l'opérateur. Celle-ci s'obtient par l'animation d'un synoptique et par les interventions de l'opérateur. Ces interventions sont possibles grâce à des panneaux listes contenant les objets sur lesquels on peut créer des forçages opératifs ou des actions de conduite, et des panneaux listes contenant les interventions de l'objet sélectionné dans l'un des deux premiers panneaux.

Afin de développer facilement toute modélisation d'une application, il est possible à tout instant d'ajouter des méthodes dans une classe, de créer de nouvelles classes, de changer d'objet dans une application. Ces modifications n'engendrent pas de temps

d'attente important, dû à la compilation, restructuration... N'oublions pas que la simulation envisagée est événementielle et que nous ne sommes pas tenus aux contraintes imposées par le temps réel.

3.4 NOTION D'ANOMALIES DE LA PARTIE OPERATIVE

Tout système de production est dépendant d'un grand nombre d'aléas. Ils sont dûs aux variations de qualité des matières d'oeuvre, des phénomènes de vieillissement et d'imprécision des objets, des perturbations extérieures,....

De ce fait, nous introduisons la notion d'anomalies de la partie opérative. Celles-ci tiennent compte des défaillances des objets technologiques, dont le comportement était supposé jusqu'à présent parfait. Cette notion d'anomalies est nécessaire compte-tenu des objectifs de la simulation. Ces défauts ont déjà fait l'objet d'études prévisionnelles de sureté telles que:

- Les AMDEC (Analyse des Modes de Défaillance de leurs Effets et de leurs Criticités). Ils recensent toutes les défaillances de tous les composants et en déterminent les effets sur le système en supposant que les autres composants fonctionnent normalement. Bien qu'apportant une grande quantité d'informations, les AMDEC sont limités à la descriptions de scénarios de propagation de pannes simples.

- Les arbres de défaillances constitués par des ensembles structurés en arborescence de scénarios incluant les pannes multiples menant à un événement redouté.

- Les graphes d'état, ensembles structurés en réseau de scénarios modélisant le système.

Ces outils permettent d'élaborer un diagnostic, dont le but est de rechercher la ou les causes explicatives de symptômes observés, tels que:

- procédure de mise en état "sûr"
- localisation des pannes
- reconfiguration du système
- maintenance à mettre en oeuvre
- évaluation de la disponibilité du système.

Un diagnostic s'appuie, entre autre, sur les compte-rendus des capteurs de sécurité, dont le rôle est de contrôler l'état du processus. Si la commande donne l'ordre

d'ouverture d'une vanne, il est clair qu'au bout d'un certain temps, en l'absence de défaillance cette vanne est ouverte. L'adjonction d'un fin de course FCO, qui indique l'ouverture effective de la vanne joue un rôle sécuritaire à la seule condition de l'utiliser, non pas de façon combinatoire, mais en y intégrant la composante temps d'une part, mais aussi d'autres aspects liés à la commande elle-même. On peut ainsi atteindre un niveau de sécurité bien supérieur, simplement par redondance logicielle.

Par exemple une bâche alimentaire, mise en ressource commune, peut alimenter deux réacteurs à travers deux vannes distinctes, mais l'alimentation simultanée n'est pas autorisée. Un test sécuritaire est de conditionner l'ouverture d'une vanne à la fermeture de l'autre. Ceci peut être géré au niveau de la partie commande, si elle existe ou au niveau de missions d'alimentation des réacteurs par échange de messages. Le test supplémentaire du capteur de fermeture vient en redondance. Par contre on ne peut se fier complètement et uniquement à ce capteur; la mémorisation par l'état de la vanne, variable d'instance gérée par l'objet, réduit l'influence des défaillances fugitives, qui affectent les capteurs et leurs interfaces [ADE].

Dans notre étude l'introduction des défaillances est faite dans le but d'observer le comportement global du processus. Imaginons en effet, le cas de la vidange d'une cuve par l'intermédiaire d'une canalisation et d'une vanne.

Nous voulons introduire une variation de débit, dûe à un encrassement, au cours de la vidange. Il est possible d'envisagér que cet encrassement provienne du tuyau ou de la vanne. L'intérêt de cette introduction d'anomalie est de constater son effet sur le système, nous ne voulons pas mettre en évidence la provenance de cette anomalie. A ce niveau, nous cherchons à introduire les conséquences d'une telle défaillance et non la cause première. Il semble alors naturel d'introduire cette notion de défaut au sein de l'objet opératif correspondant, d'autant plus que ceci évite une description et une messagerie importante.

Il existe une notion d'anomalie relative aux variations temporelles des événements. Les modèles opératifs peuvent suivre des lois stochastiques. Les variations, qui en découlent, entraînent dans certains cas des anomalies. Celles-ci peuvent être découvertes seulement après plusieurs itérations de la même séquence d'événements. Nous considérons, après un nombre suffisant d'itérations, qu'il n'y a pas d'anomalie, si le système n'a pas été mis en défaut.

3.5 INTRODUCTION DES ANOMALIES DANS LA SIMULATION

L'introduction des défauts est essentielle pour satisfaire les buts de notre simulation, nous la faisons au niveau des objets opératifs.

Il est possible de forcer des variables d'instance caractéristiques du processus à l'aide de messages spécifiques. Ces variables subissent des variations numériques adaptées à l'anomalie, que nous désirons mettre en évidence. C'est le cas d'une variation de débit.

L'anomalie peut s'obtenir par l'utilisation de variables binaires caractérisant la présence ou l'absence de défauts particuliers, de cette façon nous pouvons représenter la présence ou l'absence de court-circuit, défaut de connexion...

Sur cet exemple, nous recensons les forçages possibles sur une vanne:

- Ouverture complète
- Ouverture partielle, dont le pourcentage est précisé
- Non ouverture
- Fermeture complète
- Fermeture partielle, dont le pourcentage est précisé
- Non fermeture
- Non commande, par défaut de liaison

Lorsqu'un opérateur veut vérifier la fiabilité de la commande, et le contrôle de l'incidence des défaillances, il contrôle l'existence des verrouillages vis à vis des occurrences dangereuses. Celles-ci sont provoquées par l'opérateur lui même, en introduisant les anomalies.

Dans le cas où nous voulons former un pilote néophyte ou capter les réactions d'un pilote chevronné, les défauts surviennent, soit par une procédure spécifique, soit stochastiquement.

La procédure spécifique peut être algorithmique ou experte. Le cas expert permet de représenter les défauts causés par certaines occurrences, et ceux dont les causes sont mal définies.

Les défauts stochastiques sont introduits relativement aux caractéristiques de chaque objet comme par exemple le "mtbf" (mean time between failure), selon l'environnement dans lequel évolue l'objet.

CONCLUSION

Dans ce chapitre nous avons défini les concepts des langages objets et en particulier ceux de Smalltalk. Nous avons montré que ce type de langage est adapté à nos besoins de prototypage et de convivialité grâce à l'environnement dynamique de Smalltalk. Il est possible de satisfaire les objectifs de la simulation avec l'introduction de la notion d'anomalie.

Le dialogue entre l'utilisateur de la simulation et les différents objets dépend des objectifs, dont la multiplicité correspond à celle des rôles de l'opérateur dans une installation et de ses interventions sur le processus. C'est pourquoi, nous abordons une étude de ces divers aspects dans le chapitre suivant.

INTRODUCTION

Afin de satisfaire les objectifs fixés pour notre simulation, il est indispensable de prendre en considération les interventions effectuées par un pilote de processus industriel. Nous commençons d'abord par les recenser pour les restituer ensuite à travers un objet opératif particulier dit "opérateur", capable de dialoguer avec tous les objets technologiques, opératifs, et/ou les objets de la partie commande.

On peut ainsi soit réaliser un simple transfert des ordres d'un opérateur humain vers les autres objets, soit utiliser une véritable base experte, obtenue à partir de l'analyse du comportement de pilotes confirmés.

Cette base de connaissances peut être utilisée à des fins très diverses et en particuliers lors de l'apprentissage de pilotes néophytes.

Le couplage entre Smalltalk et Prolog permet de prendre en compte ces connaissances.

4.1 ROLE DU PILOTE DANS UNE INSTALLATION
[ADE] [BAU 86] [VAL87] [FRA 87]

L'apport de l'automatisation dans une installation assure une amélioration de la valeur ajoutée. Cette amélioration est exprimable en terme d'objectifs parfois contradictoires.

Le premier effet souvent recherché est un accroissement de la productivité du système, qui exprime une amélioration de l'usage des investissements, une meilleure rentabilité et une compétitivité accrue.

Pour obtenir une grande flexibilité, deuxième objectif important, il faut pouvoir adapter, l'outil de production et la commande associée, à de nouveaux besoins, à des gammes de produits plus diversifiées.

Un troisième effet, de l'automatisation de plus en plus prisé, est la qualité et la constance des produits assurées par une meilleure répétabilité de la valeur ajoutée.

L'automatisation permet encore d'assurer la production dans des environnements hostiles (milieu marin, spacial,...), pénibles (charges lourdes, travail répétitif, milieu bruyant,...), ou dangereux (nucléaire, atmosphère explosive,...).

Toutefois, l'automatisation reste encore très partielle dans beaucoup de processus. En effet, il s'avère très difficile en pratique d'intégrer dans la partie commande la totalité des savoir-faire humains, de sorte que certaines tâches restent confiées à des intervenants humains.

A ces causes dites techniques viennent s'ajouter des considérations économiques de compétitivité, des considérations financières imposant un fractionnement des investissements, des considérations sociales...

Certaines tâches restent donc manuelles et l'automatisation devra donc prendre en compte la spécificité du travail humain, en assurant le dialogue entre les intervenants et le système automatisé, leur sécurité dans l'exécution de leurs interventions sur le processus. De plus, l'opérateur doit établir une relation entre son installation et le monde extérieur.

En réalité, le rôle de l'opérateur est très divers. A chaque tâche correspond souvent un opérateur particulier.

Il nous paraît intéressant de les identifier et de spécifier les fonctions de chacun d'eux.

Le premier opérateur intervenant dans une installation automatisée est le décideur. Il fait les choix de production et établit le cahier des charges, en fonction de son expérience, fixant les objectifs et les contraintes de fonctionnement, de sécurité, de coût de l'installation. Il crée une connexion entre l'installation et le monde extérieur, en apportant des solutions aux problèmes conjoncturels de production, commerciaux, économiques, d'environnement... Il doit prendre la décision et informer les nouveaux objectifs de production tels que les recettes à exploiter, la quantité et la qualité des produits, à obtenir dans les meilleurs délais, sans omettre de considérer d'autres aspects comme le temps de fonctionnement, les incidents probables...

Le concepteur propose des solutions fonctionnelles capables de satisfaire les règles fixées par le cahier des charges.

Pour cela, il décide de laisser certaines missions manuelles, lorsque les automatismes ne sont pas capables de les assurer ou à un prix de revient excessif. Néanmoins, ces tâches manuelles peuvent être guidées ou comparées à une base experte.

Ces solutions tiennent compte des évolutions possibles. En effet, un cahier des charges, au cours du temps, subit des modifications, des extensions, des adaptations à de nouveaux besoins ou contraintes.

Afin de réduire les coûts, le concepteur réutilise des techniques de conception déjà employées et facilite la maintenance du programme de commande, qui devient très rapidement onéreuse.

Le pilote a pour rôle de mettre en route et de suivre l'installation. Il adapte logiciel et matériel aux besoins de la valèur ajoutée de l'installation, en intervenant sur des données caractéristiques de fonctionnement, pour des réglages, ou des mises au point. Il est chargé d'adapter au mieux l'installation aux demandes de production. Ces interventions s'effectuent lors d'un déroulement normal de fonctionnement, hormis toutes pannes, à l'occasion de modifications majeures. Il s'appuie sur une documentation complète établie lors de la conception et du suivie de l'installation.

Le pilote de conduite vérifie en permanence le respect des contraintes de production, de sécurité... Il surveille le bon déroulement des missions de l'installation, intervient, notamment en cas d'incident. Il doit communiquer avec les automatismes, ceci demande une certaine qualification qui, auparavant, était inexistante. En effet, jadis, il lui suffisait de prévenir la personne en poste d'effectuer ou de cesser une action, ou éventuellement de le faire lui même. Maintenant, l'opérateur doit dialoguer avec la partie commande du procédé. Ceci demande de la part de l'opérateur une compréhension des messages transmis par les automatismes. Le dialogue étant basé sur le principe de question-réponse. En outre, la qualification requise rend le dialogue fiable.

Différents modes de marche et d'arrêt sont introduits dans la commande du processus, de manière à permettre la conduite. Le pilote peut ainsi définir sept modes de fonctionnement.

- Le mode automatique, où le degré d'intervention de l'opérateur est relativement réduit. Il se contente d'observer et d'agir à des instants prédéfinis ou en cas de panne.

- Le mode semi-manuel, où le système de commande suit l'évolution de la partie opérative sans toutefois lui appliquer les commandes calculées. L'opérateur prend alors le rôle de commande, il génère des ordres aux objets technologiques selon sa connaissance du système.

- Le mode réglage, où l'opérateur peut introduire des points d'arrêt de façon à obtenir soit un fonctionnement pas à pas, mission par mission,... en fonction des éléments, qu'il juge nécessaire de mettre en évidence et auxquelles il apporte son expérience.

- Le mode cycle à cycle, qui est en fait une marche automatique avec un point d'arrêt privilégié, où l'opérateur peut introduire tous les paramètres nécessaires au cycle automatique suivant et autorise la continuation du programme.

- Le mode de forçage, en fonction d'une situation particulière, où l'opérateur a la possibilité d'interrompre une mission en cours et d'en forcer une autre.

- Le mode d'arrêt ou de détournement se rencontre soit lors d'un arrêt d'urgence par intervention de l'opérateur, soit en cas de discordance et d'alarme où il peut y avoir nécessité de dérouter le processus vers une procédure de repli ou de marche dégradée.

- Le mode manuel, où le seul maître d'oeuvre est l'opérateur.

Le modèle de fonctionnement de la partie commande, choisi par le concepteur du système, ne correspond qu'à un ensemble de situations prévues, c'est à dire retenues par le concepteur parmi un ensemble de situations possibles. Or, il est impératif de pouvoir faire face à des situations non prévues, en général pour des raisons économiques compte tenu de leur faible probabilité, voire imprévisibles. Seul un pilote peut alors intervenir et prendre les décisions requises par cette situation. Il assure une fonction de conduite et de surveillance du système automatisé. Cette fonction est plus ou moins assistée par des moyens informatiques, pupitres, etc... La décision prise est issue d'une expérience personnelle et provient de situations vécues à l'intérieur et à l'extérieur de l'installation. Le pilote doit utiliser les moyens disponibles pour tenir à jour une

documentation aussi précise que possible, compréhensible par d'autres opérateurs, afin d'assurer sa succession, la maintenance, l'apprentissage...

Une autre activité est celle de l'opérateur de commande. Il applique les différents modes de marche, d'arrêt et de forçage introduits par le pilote. De manière à faire face à des situations prévues ou non par les automatismes, cet opérateur de commande intervient sur les paramètres de fonctionnements ou effectue certaines commandes laissées manuelles.

La maintenance est assurée par un opérateur, qui remet en service l'installation, ou une partie de l'installation, après une défaillance d'un ou plusieurs de ces objets.

Afin de remplir sa mission, le pilote doit être capable de prélever toutes les informations significatives, nécessaires à l'analyse de la situation. Il doit agir sur le système, soit directement sur la partie opérative, en cas de dépannage, entretien prévisionnel, soit par l'intermédiaire de la partie commande, en utilisant les différents modes de marche et d'arrêt...

Il effectue un diagnostic, ce qui peut éviter les temps d'arrêt, et la dégradation du matériel liée aux pannes. Cette tâche est délicate et demande une certaine expérience. Le diagnostic fait appel de plus en plus à des moyens informatiques et, plus particulièrement, aux systèmes temps réel à base de connaissances. Il établit minutieusement une documentation permettant de faire le bilan du renouvellement matériel.

Il est évident, que ces différentes fonctions ne sont pas strictement distinctes. Entre elles, il existe un certain recouvrement. D'autre part, ces fonctions peuvent être assurées par un ou plusieurs opérateurs. Le nombre d'intervenants est souvent dépendant de l'importance de l'installation.

4.2 LES MODES DE FORCAGES [ADE] [BAU 86] [DEF 86]

Ils sont indispensables au démarrage ou lors de tests de l'installation. Ils rendent possible la modification de variables de recette en cours de fonctionnement, ils sont dans ce cas, de simples forçages de paramètres. Ils permettent également de résoudre les problèmes de sécurité ou ceux liés au fonctionnement dégradé, en cas d'anomalie, et sont alors des forçages de structure modifiant le déroulement normal d'une ou plusieurs missions.

Les forçages peuvent être utilisés grâce à la notion de point d'arrêt et de point d'entrée introduits dans les missions. Le but des points d'arrêt est de résoudre les modes d'arrêt,

de réglage et de fonctionnement cycle à cycle. Il est nécessaire de leur associer des points d'entrée permettant de reprendre le fonctionnement normal du procédé.

La disposition des points d'arrêt et d'entrée rend les forçages possibles, sous certaines conditions. Les points d'entrée ne peuvent pas être placés n'importe où. On ne peut pas y avoir accès, venant de n'importe quel point d'arrêt, dans n'importe quelles conditions. Cela dépend essentiellement du déroulement des missions.

En particulier, lorsque des missions se déroulent en parallèle, un point d'entrée figurant dans une branche du parallélisme peut être atteint seulement à partir d'un point d'arrêt appartenant à la même branche du parallélisme. En effet, sur la figure 4.1, il n'est pas possible d'accéder au point d'entrée PE à partir des point d'arrêt PA1, PA3, PA4. Dans cet exemple, seul le point d'arrêt PA2 permet l'accès au point PE. Il est donc nécessaire d'associer au point d'arrêt et d'entrée un numéro correspondant au numéro de la branche du parallélisme. Seuls les points d'entrée, dont le numéro correspond à celui du point d'arrêt forcé, peuvent être atteints.

Les points d'arrêt et d'entrée peuvent avoir une influence seulement au sein d'une mission, ils ne sont alors utilisables qu'à l'intérieur de cette mission. On peut arrêter le traitement de la mission ou dérouler des procédures particulières prévues au sein de celle-ci. Ceci permet de créer des forçages d'une situation bloquée dûe à une action conditionnelle ou itérative. Un forçage sur la variable représentative de l'élément bloquant est alors effectué. Ce type de forçage permet d'initialiser, de shunter une partie de la mission...

Des points d'entrée peuvent être atteints à partir d'autres missions. De ce fait, il est possible de lancer une mission particulière ou de quitter une situation bloquante. Lors d'une défaillance générale, du type panne d'énergie, ou lors d'une anomalie d'actionneur ou de procédé, la solution est de forcer le système vers une mission de sécurité prévue en fonction de la position dans le déroulement du traitement en cours. Ceci permet aussi de détourner une mission vers une autre, dans un but de sécurité ou dans un but purement fonctionnel.

Un ordre de forçage peut détourner une situation vers une autre, auquel cas l'ordre est ponctuel, la commande évolue aussitôt à partir de cette situation.

Cet ordre peut être un forçage et un blocage d'une situation, l'ordre continu assure le maintien du forçage et donc un blocage de la partie commande, la situation n'évolue qu'après suppression de la condition bloquante.

Le forçage ou le blocage peuvent être causés par la non transmission d'une information ou l'envoi d'une valeur erronée. Il est facile d'interrompre ou de modifier les informations que peuvent transmettre les objets technologiques de compte-rendus à la commande. Il est possible aussi de créer un état bloquant en intervenant sur les messages destinés aux objets technologiques d'action.

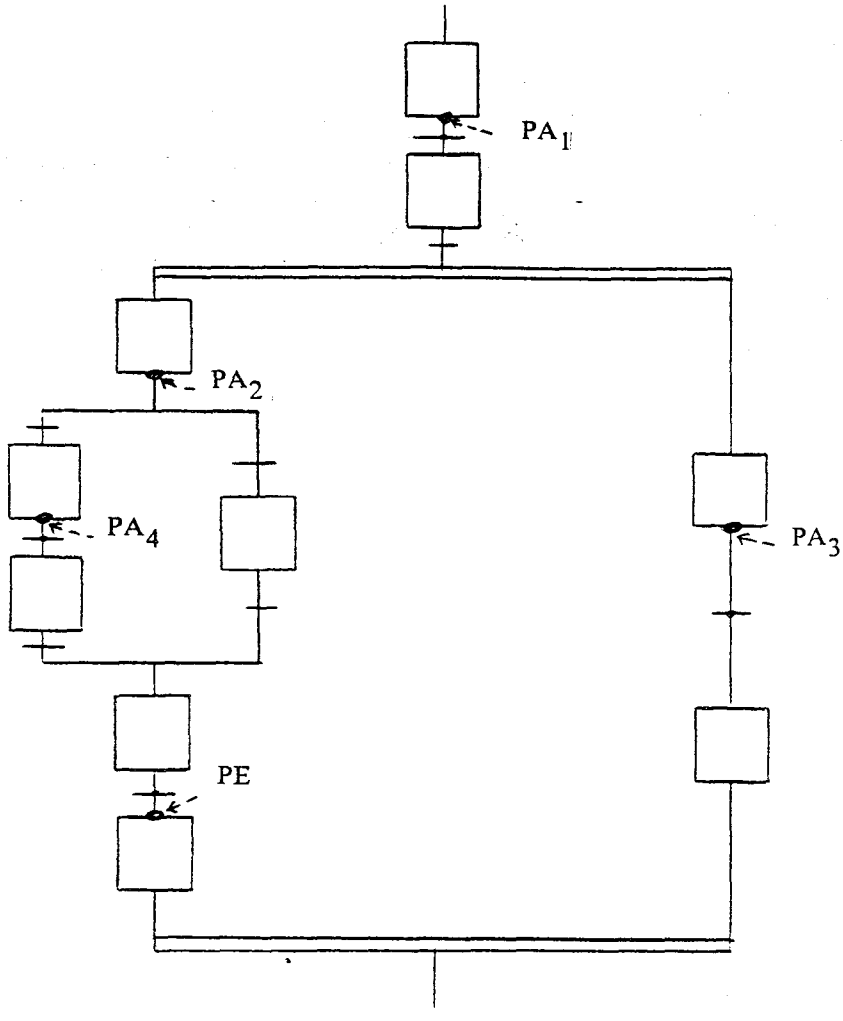


figure 4.1

Lors d'une modélisation avec partie commande, il est important de pouvoir valider le comportement du système lors d'un forçage opérateur. L'objectif pouvant être de savoir s'il est opportun de laisser à l'opérateur la possibilité de forcer une opération à n'importe quel moment en cours d'exploitation ou dans des phases particulières de nettoyage ou de mise au point.

Il convient toutefois de noter à ce stade que la simulation n'étant pas exhaustive à cause de la difficulté à définir une séquence de test parcourant tous les chemins possibles du graphe de commande, il serait plus judicieux d'admettre une modulation des modes d'action de l'opérateur en fonction des modes de marche ou d'arrêt du système. Toute action non prévue dans un mode donné doit être inopérante.

4.3 LE PILOTE SIMULE

Etant donné, les objectifs de notre simulation, tant au niveau test de la partie commande qu'au niveau apprentissage de l'opérateur, il est important d'introduire le comportement de l'opérateur dans notre simulation. Ceci est fait par l'intermédiaire d'un objet opératif dit "opérateur" capable de représenter la plus part des rôles cités précédemment.

Le rôle du décideur et du concepteur n'est pas modélisé, nous ne cherchons pas à simuler la création d'une installation, mais à représenter son fonctionnement. Toutefois, il est possible d'imaginer que le rôle du concepteur est modélisé par le système permettant le développement d'une application. Ceci commence par le choix des classes, prélevées de la nomenclature générale, de manière à constituer une nomenclature adaptée au besoin de l'application. L'instanciation de ces classes permet de définir les objets nécessaires. Un dispositif de commande, répondant aux règles du langage employé, est éventuellement construit. Le concepteur achève sa tâche par la construction du synoptique de l'installation, celui-ci permet la visualisation des évolutions et les interactions lors de la simulation.

Le rôle de l'opérateur, qui met au point l'installation, est facilité par des interventions possibles sur des variables caractéristiques du fonctionnement, lui donnant ainsi l'accès aux réglages et aux tests aboutissant à des compromis satisfaisants. Ceci lui permet une vérification de la marche normale, en excluant toutes les pannes possibles, et en dehors du contexte de production.

Le pilote peut surveiller sur son synoptique le bon déroulement des missions de l'installation. Il peut influencer les paramètres relatifs à la production, la qualité des matières premières, l'énergie disponible ou consommée,... par forçage opératif. Ceci permet de trouver la solution la mieux adaptée aux besoins de production compte tenu des impératifs conjoncturels. Un pilote néophyte peut ainsi se familiariser avec l'installation et comparer ses réactions avec celles d'un pilote chevronné, dont les interventions constituent une base de connaissances.

L'opérateur de commande agit sur des variables caractéristiques d'intervention de commande manuelle, ou d'application de modes de fonctionnement. Il provoque aussi des forçages autorisés par les automatismes.

Il est possible de créer des défaillances, soit par forçage opérateur, soit survenant stochastiquement, soit à partir de la base experte, ceci dans le but d'entraîner aux diagnostics l'opérateur de maintenance et à établir une base de connaissances relatives à ses réactions. Cet opérateur peut tenir à jour une documentation complète d'aide à la maintenance.

Afin que cet objet opérateur puisse simuler le comportement à tout niveau d'un opérateur dans une installation, il doit pouvoir dialoguer avec tous les objets opératifs, sur lesquels il peut alors créer un forçage opératif, une modification sur la matière d'oeuvre. Cela permet, entre autre, de modéliser les considérations économiques, financières, sociales, de moduler les approvisionnements des matières premières dans l'application. Il est tout à fait envisageable d'introduire des variations de quantité et de qualité de produit dans un réservoir, si la livraison de ce produit n'est pas prise en compte dans l'application et si la qualité de ce produit importé est indépendante de l'installation étudiée.

L'opérateur peut également établir un dialogue avec chaque objet technologique. Il peut ainsi effectuer les actions manuelles, les observations, la maintenance et les dépannages nécessaires. Un dialogue avec la partie commande lui permet de provoquer des forçages et des interventions autorisés par les automatismes.

Par ces faits, nous rendons toutes les interventions opérateurs possibles, elles sont obtenues par simple envoi de messages.

Cet objet opérateur permet d'intègrer une base de connaissances capable de restituer le comportement d'un pilote confirmé.

Pour cela, il est indispensable de pouvoir intégrer le savoir-faire d'un ou plusieurs experts. En effet, il est nécessaire que toute personne intervenant sur le processus contribue à l'élaboration du système expert. Il ne faut pas oublier que bien souvent les opérateurs ont une perception instinctive du déroulement du processus . Il faut par conséquent essayer d'obtenir d'eux par de judicieuses questions une formalisation de leurs connaissances.

Le pilote simulé ne doit pas systématiquement proposer une seule réponse faisant face à un incident. A l'apparition d'un problème mineur dans l'installation, chaque opérateur propose une solution, pas nécessairement optimisée. Il est donc capital que le pilote simulé restitue plusieurs actions possibles, la plus efficace bien sûr, si elle est connue, mais aussi toutes celles qui viennent à l'esprit des opérateurs. Lors de la simulation un choix parmi ces solutions est donc à faire, ceci présente un grand intérêt pour l'apprentissage de pilotes. Ils peuvent prendre connaissance de plusieurs solutions et en constater les conséquences.

A l'apparition d'anomalies, le pilote simulé peut proposer une séquence de tests à effectuer et en déduire les causes les plus probables.

4.4 RELATION DE SMALLTALK AVEC PROLOG. LE PILOTE EXPERT [NOY 84] [FER 85] [BEN 87] [LOU 87]

Les langages orientés objets prennent un place de plus en plus importante dans les systèmes experts. Résoudre un problème dans un domaine spécifique réclame une grande quantité de connaissances liées à cette activité. Un système expert orienté objet présente l'avantage de pouvoir fragmenter le domaine d'activité et de regrouper les informations de même nature et les règles s'y rapportant. Nous allons montrer l'apport de Prolog/V dans un environnement tel que Smalltalk/V.

Il est possible d'intégrer Prolog/V à Smalltalk/V . Dans les méthodes Smalltalk/V, il est aisé d'envoyer une série de buts Prolog/V à une base de connaissances qui est une instance de la classe Prolog ou une de ses sous-classes. Après avoir satisfait ces buts, Prolog/V retourne un tableau de solutions, qui peuvent être utilisées dans le programme Smalltalk.

Dans les définitions de Prolog/V, il est possible d'envoyer à Smalltalk un message avec le prédicat "is" et d'unifier la réponse avec une variable Prolog, la variable Prolog est dans ce cas appariée à la réponse du message. Ceci permet au sein de la base de connaissances d'évaluer une expression Smalltalk et de rendre totale l'interaction entre Prolog/V et Smalltalk/V.

Contrairement à un Prolog standard, Prolog/V peut utiliser plusieurs bases de connaissances simultanément et établir une communication entre elles. En effet, une base de connaissances correspond à la définition d'une classe Smalltalk, le dialogue entre deux bases est obtenu par l'intermédiaire du prédicat "consult".

Les classes sont créées dans la fenêtre logicBrowser à partir d'un compilateur propre à Prolog/V. Ce compilateur est une sous-classe de celui de Smalltalk. Comme en Smalltalk, les sous-classes de Prolog/V bénéficient du caractère polymorphique, il est possible de définir les mêmes clauses dans différentes classes, une clause étant un fait ou une règle Prolog, sans pour autant que le traitement sur ces clauses soit identique. Les sous-classes héritent des prédicats définis dans leurs super-classes, ce qui simplifie l'écriture des clauses d'une base de connaissances.

```
coupureEnergie :
        coupureEnergie(VanneFMA, #fermeture)
        coupureEnergie(VanneOMA, #ouverture)
        coupureEnergie(Moteur, #arrêt)
```

figure 4.2

En Prolog/V, toutes clauses ayant le même nom dans une classe sont regroupées et compilées au sein d'une même méthode. Le nom de la méthode devient par conséquent

le nom des clauses qu'elle rassemble accompagné des ":". Une méthode Prolog/V est toujours compilée comme une méthode Smalltalk avec arguments figure 4.2.

Il existe une variable de classe "Database" de la classe "Logic" surclasse de Prolog, cette variable est un dictionnaire dont chaque élément est une base de faits évolutive. Ceci est fortement employé à la construction de la base de connaissances du pilote expert.

L'intérêt, d'un système expert écrit à partir de Smalltalk, est la possibilité d'agir sur tous les objets d'une classe dont les variables d'instance vérifient certaines conditions. En effet, en considérant les instances de la classe des vannes, il est possible d'y introduire une défaillance. Les vannes, ayant un débit supérieur à un débit donné, sont supposées être montées dans un circuit imposant une pression élevée. Par conséquent, la défaillance provient d'une fuite éventuelle, survenant au bout d'un certain temps d'utilisation.

Smalltalk apporte d'autres intérêts à Prolog. Comme dans tout système expert les faits ne faisant pas partie de la base de connaissances peuvent être ajoutés. De plus, ces nouveaux faits appliqués à un objet peuvent désormais être connus de toutes les instances de la classe. Le fait d'introduire Prolog dans la simulation favorise le prototypage, un des caractères essentiels de notre outil. L'environnement convivial offert par Smalltalk tel que le multi-fenêtrage, souris, menus, est un atout important pour la lisibilité, l'accessibilité aux connaissances et la construction des clauses.

Le fait, que chaque classe Smalltalk peut avoir sa propre base de connaissances simplifie l'écriture de la partie experte de la simulation.

Le pilote expert a un rôle important dans la simulation. Il est le seul capable de construire une base de faits et de règles à appliquer. Sa contribution est primordiale et délicate. En effet, la validité du système expert découle de la minutie et de l'exactitude qu'il a apporté à la construction de ces bases. En voulant bâtir une base de connaissances, il est possible de créer dans certains cas des incohérences dûes, à des erreurs de transcription du raisonnement ou d'éventuelles lacunes de l'expert.

Il est clair, qu'à l'heure actuelle les méthodologies de création et de maintenance des systèmes experts, dont disposent les industriels, sont quasi-inexistantes.

La phase la plus délicate de la création d'un système expert est celle de l'acquisition de la connaissance. L'extraction du savoir de l'expert met en jeu la validité du système, avec les conséquences dramatiques qu'il peut en résulter. Cette tâche est du ressort du cogniticien, qui doit essayer d'obtenir le maximum de connaissances et tester la cohérence de la base.

Dans notre cas, le fait, que l'expert construit lui même sa base de faits au cours de multiples simulations, éventuellement encadré par un cogniticien, est important. En effet, l'apparition de défauts, survenus stochastiquemment ou introduits par l'opérateur,

rappelle ou fait prendre conscience au pilote de certains problèmes. Il peut alors compléter la base de connaissances en y apportant des solutions.

La partie experte de notre simulation présente un atout important. En effet, cette partie se compose de plusieurs bases de connaissances indépendantes. Ceci permet à différents experts possédant un domaine de connaissance très spécialisé d'établir leurs bases de connaissances, sans avoir nécessairement à confronter leur point de vue.

CONCLUSION

Nous avons défini le rôle de l'opérateur dans une installation et les moyens qui lui permettent de remplir sa mission. Pour les raisons évoquées au cours de ce chapitre, nous jugeons utile de simuler le comportement de ce pilote. Ceci est concevable dans la mesure où nous introduisons des bases de connaissances construites par des pilotes confirmés et destinés à l'enseignement.

Il faut maintenant exploiter les possibilités de la simulation. Pour cela, quelques notions doivent être développées, avant de profiter de cet outil.

CHAPITRE 5

Mise en oeuvre de la simulation

INTRODUCTION

Les méthodologies de décomposition d'un système et le rôle de l'opérateur étant définis, nous allons présenter les moyens de mise en oeuvre de cet outil.

Au niveau de la structure, certains objets sont indispensables au déroulement de la simulation, bien qu'ils n'appartiennent pas à la décomposition donnée dans les chapitres précédents. Il s'agit des objets dont le rôle est de gérer la notion de temps, le parallélisme, le graphique, l'interactivité.

Les moyens interactifs employés dans cet outil permettent l'élaboration de la simulation tant au niveau conception qu'utilisation. Ceci sont obtenus grâce à la convivialité proposée par un environnement tel que Smalltalk.

5.1 STRUCTURATION DE LA SIMULATION

L'organisation de la programmation est schématisée par la figure 5.1. On y retrouve les objets opératifs, technologiques, missions, commandes et opérateur, présentés dans les chapitre précédents.

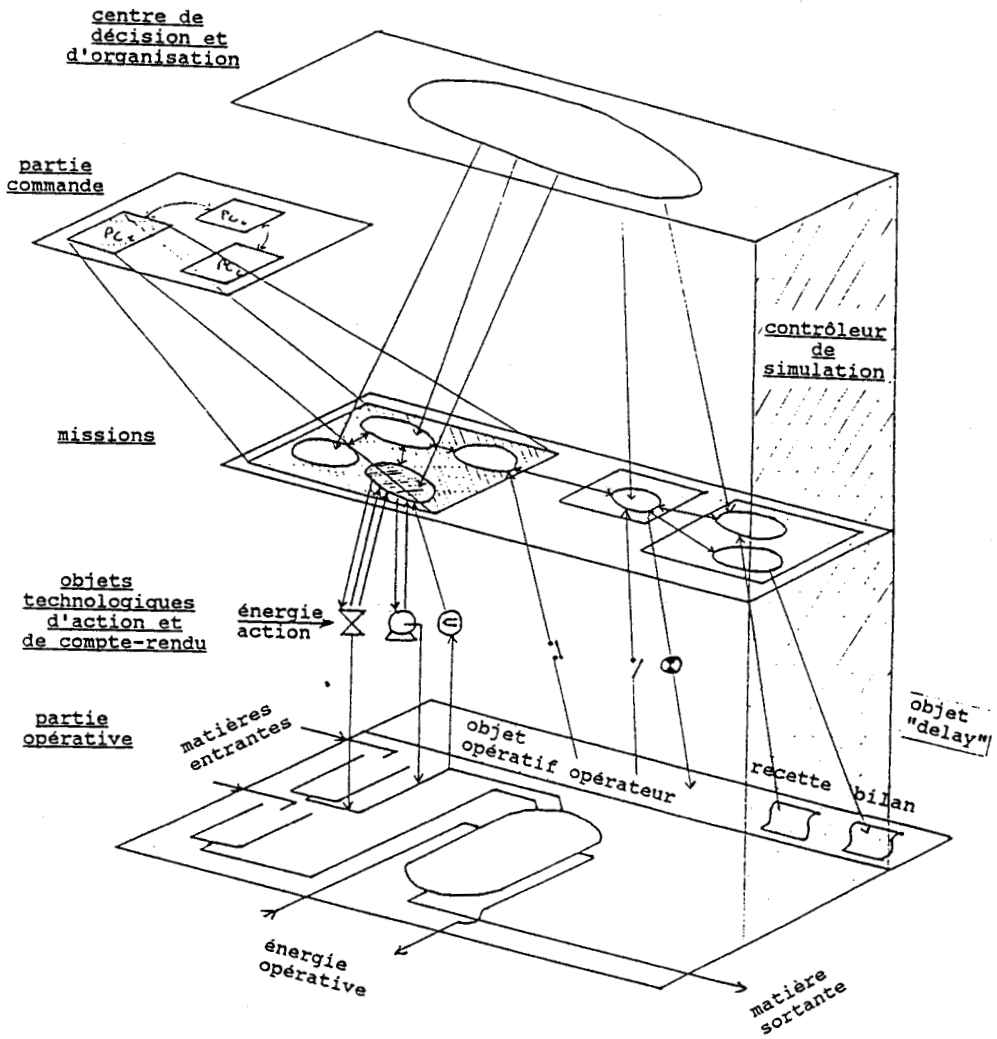


figure 5.1

Il faut leur associer:

- L'objet "delay", introduit la notion de temps dans la simulation. En effet, il nous permet de fixer la durée d'un pas de simulation, de considérer la notion de temporisation que l'on retrouve dans les objets missions, technologiques et opératifs. La temporisation permet d'évaluer la durée d'une mission. Les objets technologiques utilisent cette notion, afin de restituer le temps de réponse quand ils reçoivent un ordre. La temporisation fait intervenir le facteur temps dans l'évolution du processus, définie dans les objets opératifs.

Cet objet "delay" peut être un générateur d'événements. Il permet de tenir compte de la dynamique des objets et provoque des événements survenant aléatoirement selon les caractéristiques des objets.

- Le centre de décision et d'organisation gére l'activité des objets missions et plus particulièrement les problèmes de décompositions dynamiques ou de ressources communes.

- Toute l'organisation de la simulation repose sur un objet dit "contrôleur de simulation", qui assure essentiellement la gestion du parallélisme, l'évolution pas à pas et la mise en oeuvre du dialogue avec l'utilisateur. Cet objet possède une variable d'instance contenant la liste des objet missions actifs. Un objet mission est dit actif, lorsque sa mission dure plusieurs pas de simulation. A chacun de ces pas, le contrôleur donne la possibilité aux éléments de la liste d'effectuer un fragment, relatif à la durée du pas, de leur mission. Ces objets, exécutent leur mission au fil des pas de simulation. Nous disons, dans ce cas qu'il y a une autogestion des objets puisque dans un premier temps, ils reçoivent de la commande ou d'un objet mission, l'ordre d'effectuer une mission, puis ils la réalisent petit à petit à chaque fois que le contrôleur leur en donne la possibilité. Quand un objet actif a terminé sa mission, il est supprimé de la liste.

Ces mêmes objets peuvent être utilisés sans appartenir à la liste du contrôleur, lorsque l'objet remplit sa mission lors du pas en cours.

Le contrôleur fixe le pas de simulation dont la durée est modifiable à tout instant. Ceci est un point très intéressant de la simulation. En effet, les phases longues ou celles qui semblent les moins intéressantes, peuvent subir une accélération. Le phénomène inverse peut s'appliquer aux phases méritant une attention particulière de la part de l'opérateur.

Un objet opératif peut faire part de son évolution, s'il a reçu un ordre d'un objet mission, via un objet technologique. Dans le cas d'une anomalie intempestive, l'objet opératif ne reçoit pas obligatoirement d'ordre d'un objet mission. Afin de restituer les anomalies intempestives, le contrôleur détient la liste des objets opératifs subissant ce type de défaillances et leurs permet d'évoluer à chaque pas de simulation.

Le contrôleur sert aussi aux objets technologiques, qui possèdent un état par défaut, c'est à dire un état que l'objet retrouve en l'absence de toute commande. Par exemple une vanneFMA se ferme dès qu'elle n'est plus sollicité par la commande d'ouverture, celle autorisant le passage de l'air comprimé. Cet état "fermée" est celui par défaut pour ce type de vanne. Le contrôleur permet aux objets technologiques figurant dans la liste des objets technologiques ayant un état par défaut, de retrouver cet état de repos, s'ils n'ont pas reçus de commande au cours du pas.

A cet objet est attachée la fenêtre de simulation, par laquelle est obtenue l'interactivité pendant la simulation. Ceci lui permet de prendre en considération toute action de l'opérateur.

5.2 GESTION DU PARALLELISME

Dans le mécanisme classique d'implantation de Smalltalk, l'envoi d'un message provoque l'adjonction de la méthode compilée, se rapportant au sélecteur du message, dans une pile des contextes. A la réponse du message, la méthode compilée est enlevée de la pile. L'exécution d'une méthode nécessite l'envoi de nouveaux messages, qui sont eux mêmes empilées. Seuls, ces messages peuvent être empilés. Tant que la pile n'est pas vide, il est impossible d'envoyer un message n'ayant aucune relation avec la méthode en cours. Cette stratégie interdit le multi-tâches dans la version Smalltalk/V utilisée.

Le parallélisme s'obtient en perturbant l'exécution des méthodes. Il faut pouvoir envoyer un ou plusieurs messâges sans en attendre les réponses. Il existe plusieurs possibilités pour obtenir ce pseudo-parallélisme.

- Soit en créant différentes piles, autant que d'actions à effectuer simultanément. Ceci demande une gestion de piles et une commutation dans l'exécution de chaque pile. Il est difficile de définir les points de commutation sans modifier les méthodes. En effet, au sein d'une même méthode certains messages doivent être envoyés lors du même pas de simulation, c'est le cas d'une affectation de variable, de comparaison de variables,... alors que d'autres doivent être envoyés sur plusieurs pas, pour obtenir des évolutions simultanées.

- Soit en faisant intervenir plusieurs pointeurs de pile. Il faut alors charger la pile des méthodes compilées à effectuer en parallèle, puis les exécuter. Dans ce cas, il existe un pointeur associé à chacune des méthodes figurant dans la pile. Une gestion de pointeurs est nécessaire afin de permettre aux méthodes compilées d'envoyer leurs messages. La définition des points de commutation reste difficile à maîtriser.

- Soit en modifiant la compilation d'une méthode de façon à interrompre les méthodes compilées. Dans l'écriture d'une méthode, le concepteur place lui même les points de commutation pour permettre l'envoi d'autres messages.

Les deux premières techniques sont plus élégantes, mais elles ne sont pas employées d'une part, du fait de la difficulté à définir les points de commutation et d'autre part, à cause d'un manque d'informations délivrées par Digitalk, tant au niveau de la constitution de la pile, qu'au niveau de la compilation des méthodes.

Le parallélisme est obtenu par l'écriture modifiée des méthodes développées au sein des classes d'objets missions et commandes.

La figure 5.2 montre une exemple d'écriture de ces méthodes.

```
exécutionRéact
        "exécute un programme de commande du réacteur
        de l'exemple Rhodia."
        $£. dcy état Z.
        charg méthode: #charge.
        $£. agit méthode: #marche Z.
        $£. charg finCharge Z.
        $£. dissol méthode: #chauff Z.
        $£. dissol finChauff Z.
        charg désactive.
        $£. crist refroid Z.
        $£. agit fin Z.
        $£. charg vidange Z.
        self réinit.
```

figure 5.2

A cette méthode ne correspond plus une seule méthode compilée, mais une collection ordonnée de méthodes compilées. Le premier élément de cette collection n'est jamais exécuté, il permet l'édition de la méthode. Les autres méthodes compilées correspondent chacune à une fraction de la méthode, délimitée par le caractère Z placé par le concepteur. Ce caractère est facultatif en fin de méthode. Les caractères $£ permettent de renvoyer la réponse booléenne du message suivant. Tant que la réponse est fausse, la même méthode compilée est employée. Dès que la réponse est vraie la méthode suivante est exécutée.

Dans notre exemple, nous utilisons le message "charg méthode: #charge". Ce type de message est envoyé à un objet mission. Il rend ce dernier actif et, complète par conséquent la liste des objets mission du contrôleur. L'objet mission "charg" reçoit l'ordre d'effectuer la méthode "charge", il la réalise au cours de la simulation. Alors que

le message "charg désactive" est une mission exécutée par le même objet, durant un pas de simulation. L'instance s'envoie le message "réinit", qui la replace alors dans la situation initiale. Il est à noter que ces méthodes ne peuvent pas avoir de variables de travail.

A la création d'une telle méthode, la collection de méthodes compilées s'obtient à l'aide de l'objet "interpréteur". Cet objet repère le sélecteur de la méthode avec ses arguments et les caractères, Z délimitant une méthode compilée, $£ désignant la réponse booléenne de la méthode compilée.

L'exécution de ce type de méthode se fait par le message

destinataire méthode: argument

Destinataire est l'objet recevant ce message et sur lequel la méthode argument doit être exécutée. Argument est un symbole si le message à exécuter est unaire, dans le cas contraire, c'est un tableau dont les éléments constituent le sélecteur et les arguments indispensables au message.

La réception de ce message par un objet mission ou de commande affecte la variable d'instance "méthode" de l'objet. Sa valeur devient alors la collection ordonnée de méthodes compilées. Il existe aussi un index pointant la méthode compilée qui doit être exploitée, c'est à dire celle qui doit figurer dans la pile. La variable d'instance contenant la liste des objets missions actifs du contrôleur est également complétée.

A chaque pas de simulation, chacun des objets de cette liste exécute la méthode compilée correspondant à l'index. Ce dernier reste inchangé tant que la réponse de la méthode compilée est fausse. Par conséquent, au pas de simulation suivant la même méthode compilée sera empilée. Sa réponse est le résultat du message précédé par les caractères "$£". Dans le cas où la réponse devient vraie, l'index est alors incrémenté. Un objet est supprimé de la liste des objets missions actifs, quand la réponse de la dernière méthode compilée de la collection est devenue vraie.

L'objet contrôleur gère l'activité des objets missions qui eux mêmes gèrent celle des objets technologiques et opératifs en leur envoyant des messages.

5.3 L'INTERACTIVITE

Le départ de la simulation fait apparaître un logo, ainsi qu'une fenêtre, constituée d'un panneau liste et d'un panneau texte, représentée figure 5.3

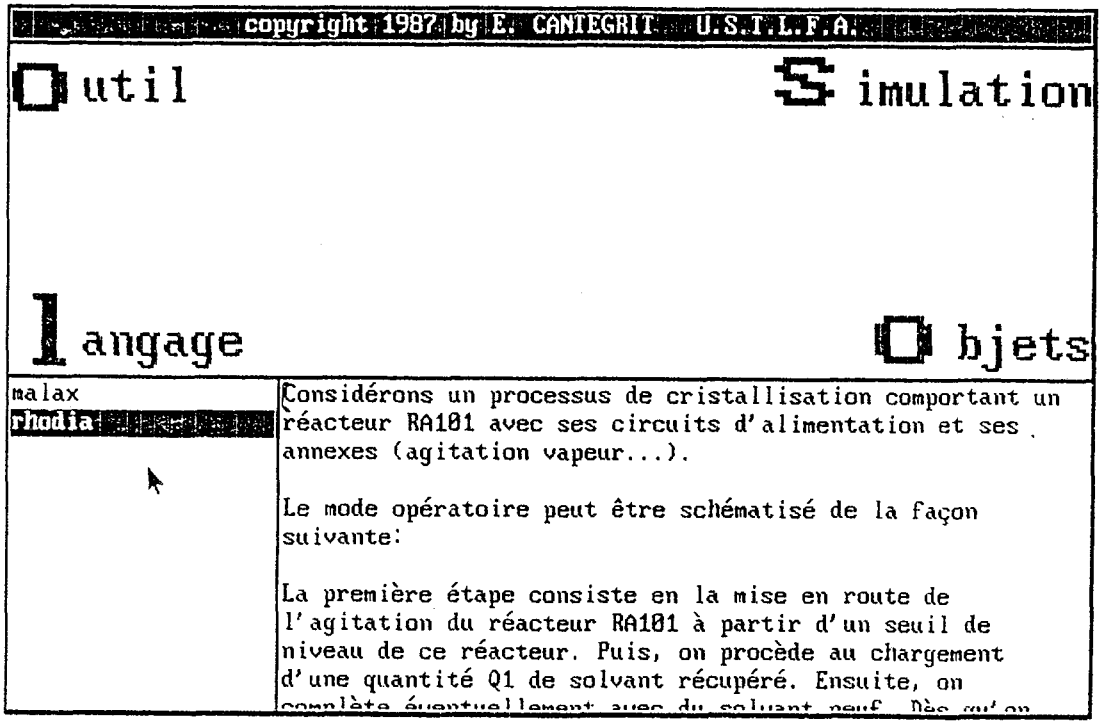


figure 5.3

Le panneau liste contient toutes les applications créées, la sélection d'un de ces éléments affiche un libellé explicatif du cahier des charges de l'application dans le panneau texte.

Un menu est lié au panneau liste. Il donne le moyen à l'utilisateur de créer une nouvelle application, de supprimer, d'installer celle qui est sélectionnée.

Dans le menu système, existe une option "bibliothèque" où s'effectue la création des diverses classes d'objets missions, technologiques, opératifs ou de commandes. Ceci se fait au sein d'une fenêtre intitulée "bibliothèque" représentée figure 5.4

Cette fenêtre est composée de deux panneaux liste et d'un panneau texte.

Il est possible d'afficher les sous-classes correspondantes à la sélection "mission, technologique, opératif" dans le panneau liste de gauche. La sélection, d'une de ces sous-classes, affiche les variables se rapportant à cette classe dans le panneau texte, et les sélecteurs existants dans le dictionnaire de méthodes de la classe dans le panneau liste de gauche. Ces sélecteurs correspondent aux méthodes d'instance ou de classe selon la sélection "instance" ou "classe".

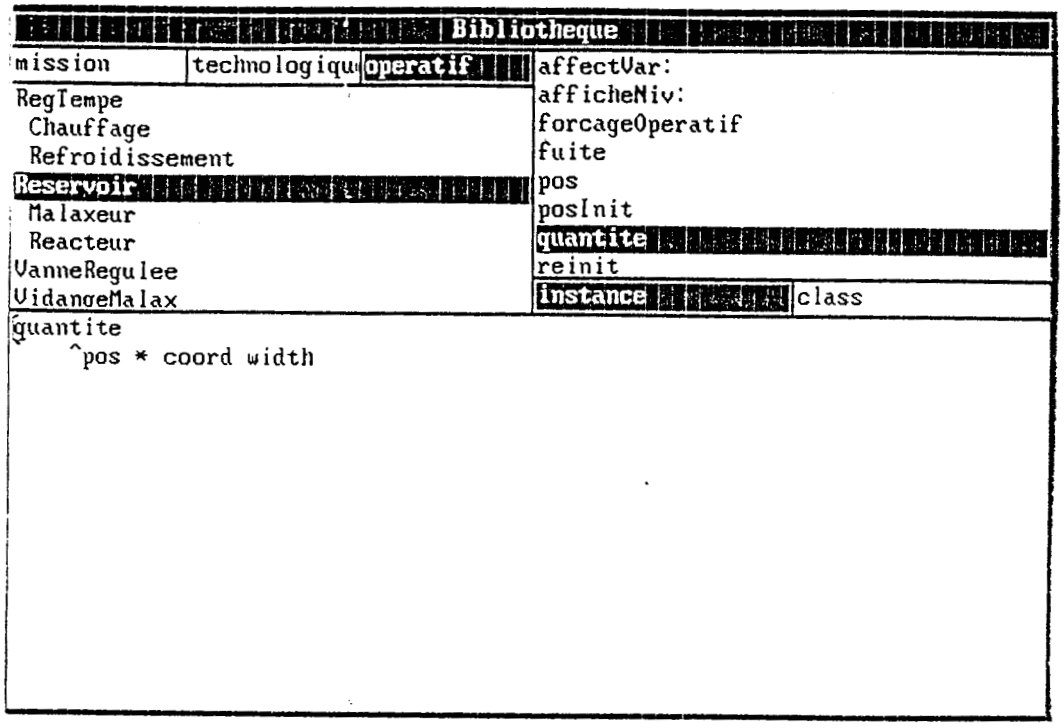


figure 5.4

L'usage d'un des sélecteurs dans le panneau concerné transmet le contenu de la méthode désignée dans le panneau texte.

Au panneau liste contenant les classes est lié un menu permettant de sauvegarder une classe dans un fichier, de créer de nouvelles classes ou sous-classes, d'imprimer le contenu d'une classe, de construire la représentation graphique de la classe sélectionnée.

Le menu, attaché au panneau liste des méthodes, autorise la suppression, l'ajout, la sauvegarde dans un fichier, l'impression d'une méthode.

Le menu du panneau texte est le menu standard de ce type de panneau en Smalltalk/V, il permet toute intervention classique au sein d'un tel panneau.

L'installation ou la création d'une application entraîne l'ouverture d'une fenêtre "instanciation" représentée figure 5.5. Toutes les instanciations des classes nécessaires à l'application se créent dans cette fenêtre.

instanciation des objets de l'exemple: rhodia		
mission	**technologique**	**operatif**
Cristallisation	BoutonPoussoir	OpBascule
Dissolution	Capteur	RegTempe
MalaxChargBrique	Compteur	Chauffage
MalaxChargVanne	Moteur	Refroidissement
MissCharg	MoteurPivot	**Reservoir**
MissMalax	MotMelangeur	Malaxeur
MissMalaxRot	Pompe	Reacteur
Agitation	ObjTechChar	VanneRegulee
MissMalaxVidange	ObjTechLev	VidangeMalax
ObjMissChar	Signal	
instances	donner le nom:	
xv103	positionLiquideInitiale:	
tc1012	couleurProduit:	
hxv100		
li1010		
hp		
p232		
ti1013		
tc1013		
li2310		
xv101		

figure5.5

Celle-ci se compose de quatre panneaux liste et d'un panneau texte.

Quatre entêtes correspondent aux quatre panneaux liste pour désigner les différentes sous-classes "mission, technologique, opératif" et les instances. Sous chacun, de ces quatre entêtes, se trouve un panneau liste contenant, soit la liste des sous-classes appropriées, soit la liste des instances de l'application.

Une sélection dans l'un des panneaux contenant une des listes des sous-classes provoque l'apparition de questions dans le panneau texte auxquelles le concepteur doit répondre, afin d'instancier la classe désignée. Les questions posées portent sur le nom de l'instance, des affectations de variables tel que le niveau initial d'une cuve, ou le nom des objets auxquels l'instance envoie des messages...

Toute instance créée se trouve dans le panneau liste prévu à cet effet. Il est possible de supprimer certaines d'entre elles à l'aide du menu de ce panneau.

Le menu du panneau texte permet de valider une instance, de créer l'édition des liens, lorsque toutes les instances de l'application ont été validées. Il permet aussi de sauvegarder toutes les instances de l'application dans des fichiers, de récupérer ces fichiers pour lancer une application, d'avoir accès aux fenêtres "synoptique" et "simulation".

La fenêtre "synoptique" représentée figure 5.6 permet de construire la représentation de l'application. Il suffit de pointer une instance du panneau liste pour obtenir son symbole graphique et de le déplacer dans le panneau graphique pour lui donner son emplacement dans le synoptique.

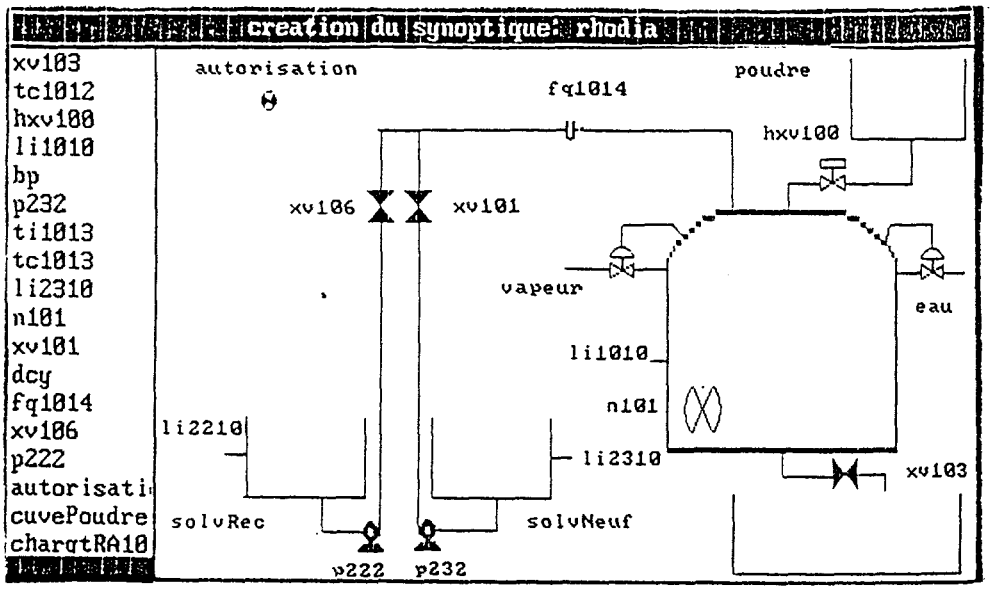


figure 5.6

Le menu du panneau graphique donne toutes les possibilités nécessaires à la construction graphique, que nous évoquerons plus tard.

Celui du panneau liste permet de valider le synoptique, de mettre ce graphe dans un fichier et d'avoir accès à la fenêtre "simulation".

Cette fenêtre "simulation" donnée figure 5.7 se compose de quatre panneaux liste et d'un panneau graphique dans lequel est représenté le synoptique, qui est alors animé.

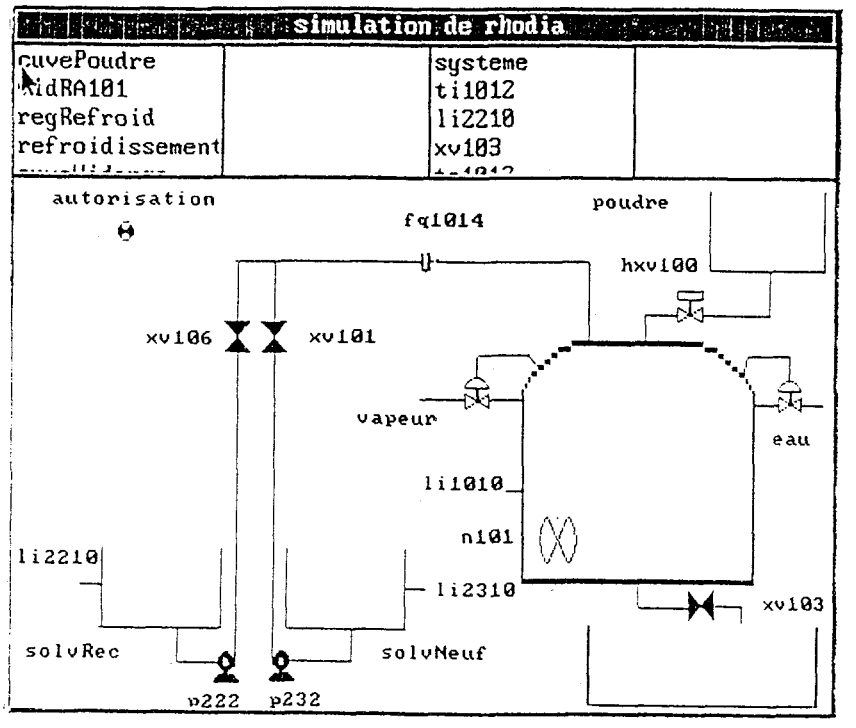


figure 5.7

Les panneaux liste servent aux interventions de l'opérateur. Les panneaux de gauche sont utilisés pour réaliser des forçages opératifs, l'opérateur sélectionne un objet dans le panneau à l'extrême gauche, la liste des forçages possibles sur cet objet apparaît dans le panneau du centre gauche. Les listes de droite donnent les fonctions normales d'un opérateur. Dans le panneau du centre droite figure la liste des objets sur lesquels l'opérateur peut agir. Les paramètres accessibles sont alors présentés dans le panneau à l'extrême droite.

Sur le synoptique, apparaissent toutes les informations accessibles à l'opérateur et nécessaires au pilotage.

Le menu lié au panneau graphique permet de modifier la durée des pas de simulation, de lancer la procédure d'initialisation de la simulation.

L'opérateur peut interrompre le déroulement de la simulation grâce à une primitive contrôlant l'état de la souris. Seule une intervention souris peut perturber le déroulement du programme.

Dans toutes les fenêtres, chaque menu des différents panneaux possède une rubrique "help", qui guide l'utilisateur dans ses interventions en fonction du panneau dans lequel il intervient.

Au cours d'une application, l'utilisateur peut à tout moment changer d'objet, c'est à dire qu'il peut remplacer par exemple une vanne classique par une vanne fermée par manque d'air. Pour cela, il lui suffit de supprimer l'objet non désirable de la liste des instances de la fenêtre "instanciation", de créer la nouvelle instance remplaçant la précédente, si le nom de la nouvelle instance n'est pas le même que celui de l'ancienne, il faut donner la nouvelle valeur à la variable d'instance des objets concernés. Il est indispensable de refaire une édition des liens, puis de placer à l'endroit voulu la représentation graphique de la nouvelle instance dans le synoptique.

5.4 LE COMPORTEMENT GRAPHIQUE

Les représentations graphiques des classes sont construites à l'aide d'un "bit editor" donné figure 5.8. L'opérateur donne un nom et dimensionne chacune d'elles.

Un bit editor est une fenêtre dans laquelle l'opérateur peut sélectionner ou non des pavés. Chaque pavé correspond à un pixel de l'écran. Cette fenêtre est un zoom de la représentation graphique que l'utilisateur désire construire. La forme d'un objet peut être de dimension variable ou non. Si un réacteur a une représentation graphique bien définie, sa dimension ne l'est pas et varie à chaque instanciation. Le concepteur a donc la possibilité de valider la forme qu'il vient de construire, soit en pointant un pavé désignant la représentation de dimension invariable, soit en pointant l'autre pavé, auquel cas cette représentation est de dimension variable. Une classe possède un dictionnaire des représentations graphiques construites désignées par leur nom.

Dans le synoptique, l'opérateur dispose et éventuellement dimensionne les objets. Le menu du panneau graphique permet de faire des rotations, des symétries par rapport à l'axe horizontal des figures. Il est possible d'insérer du texte, de tracer des éléments reliant les différentes figures tel que des tuyaux, des fils électriques...

L'animation de la simulation est obtenue par des modifications telles que des variations de niveaux, des changements de couleur selon l'état de objet, l'apparition ou la

disparition d'objets, mais aussi en utilisant le dictionnaire des différentes représentations graphiques de la classe. Ceci permet d'obtenir une animation comme la vidange d'un réservoir par son pivotement.

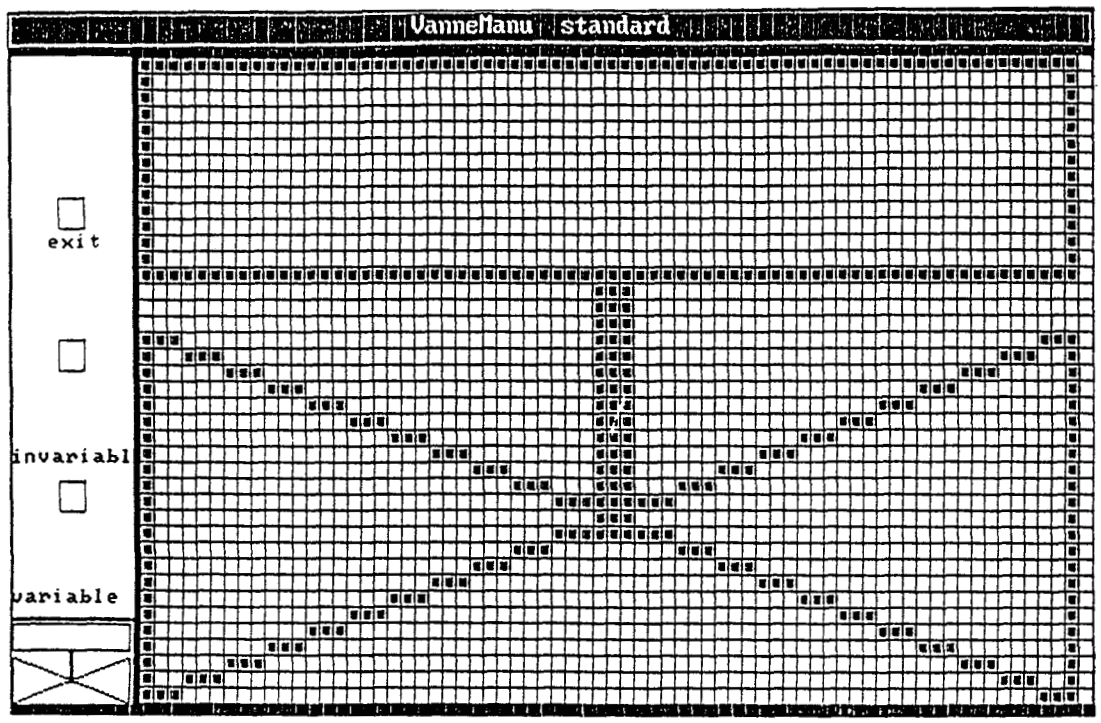


figure 5.8

5.5 SIMULATION ET GESTION DU TEMPS

L'objet contrôleur, assisté de l'objet "delay", gèrent le temps. La durée d'un pas de simulation est fixé par l'utilisateur.

En début de pas, une temporisation est initialisée puis lancée. Quand toutes les actions à effectuer lors du pas de simulation sont achevées, cette temporisation est comparée à la durée du pas. Le passage au pas suivant s'effectue, lorsque la temporisation est au moins égale à la durée du pas.

Il est évident, qu'une opération lancée plusieurs fois peut avoir une durée différente. La modélisation de ces variations s'obtient par l'introduction de lois stochastiques au

sein des objets opératifs. Celles-ci fixent l'importance des fluctuations relatives aux événements.

De cette façon l'accélération ou le ralentissement de la notion de temps joue sur toute la simulation. Ceci semble intéressant dans le cas d'une simulation de contrôle du mode opératoire. En effet, dans ce cas une accélération permet de vérifier un maximun d'informations en un laps de temps minimun. Par contre, dans un simulation didacticielle il est indispensable de garder une notion approchée du temps. Le temps d'évolution d'une phase ne correspond pas forcément au temps réel de la réalisation de cette phase. Ceci est peu important car la simulation est utilisée hors ligne.

Par ailleurs, nous avons déjà mentionné que la simulation considère la dynamique propre à chaque objet. Ceci implique que les objets prennent en compte le temps pendant lequel ils sont sollicités.

CONCLUSION

L'outil de simulation étant développé, à la fois au niveau interne et convivial, il permet à quiconque de créer son application. L'utilisateur ne doit pas nécessairement connaître Smalltalk, sauf dans le cas où il souhaite créer de nouvelles classes.

Il convient toutefois de souligner la difficulté initiale de la formation des concepteurs de simulation à l'esprit objet. L'empreinte procédurale laisse souvent des cicatrices ineffaçables, dans le mode d'abord des problèmes.

Nous allons consacrer le chapitre suivant à mettre en application ce travail, sur un exemple industriel réel. Il permet de rendre compte de la simplicité de mise en oeuvre de la simulation d'une telle installation.

CHAPITRE 6

Exemple d'application: Rhodia

INTRODUCTION

Nous allons développer dans ce chapitre une application nommée Rhodia, caractéristique des processus par lot. Il s'agit d'une partie d'un exemple réel, que nous élaborons depuis la création des classes dont nous donnons quelques exemples, jusqu'à la conduite de la simulation, en mettant un accent particulier sur les raisons de notre décomposition tant au niveau des objets opératifs qu'au niveau des objets missions.

6 EXEMPLE D'APPLICATION: RHODIA

Considérons figure 6.1 un processus de cristallisation comportant un réacteur RA 101 avec ses circuits d'alimentation et ses annexes (agitation, vapeur...).

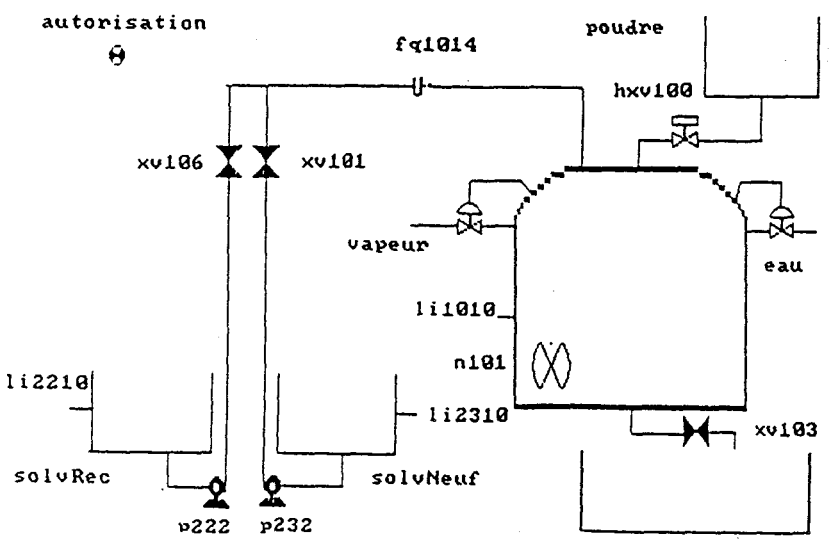


figure 6.1

Le mode opératoire peut être schématisé de la façon suivante:

La première étape consiste en la mise en route de l'agitation du réacteur RA 101 à partir d'un seuil de niveau de ce réacteur. Puis, on procède au chargement d'une quantité Q1 de solvant récupéré. Ensuite, on complète éventuellement avec du solvant neuf. Dès qu'on reçoit l'autorisation on charge le produit solide. Le chauffage programmé du réacteur RA 101 est mis en route. On procède au maintien en température pendant un temps donné t1, pour effectuer la dissolution.

On refroidit ensuite, pour effectuer la cristallisation, jusqu'à un seuil de température. Puis, on maintien l'agitation pendant un temps t2.

Nous commençons par recenser les objets technologiques intervenant dans l'installation.

Nous remarquons sans difficulté que plusieurs d'entre eux appartiennent à des classes identiques. Il est indispensable que les classes de la figure 6.2 soient existantes dans la bibliothèque. Dans le cas contraire, il est nécessaire de les créer.

TC1012 TC1013	Vanne
XV101 XV103 XV106	VanneFMA
HXV100	VanneManu
P222 P232	Pompe
FQ1014	Compteur de débit
N101	Moteur
LI1010 LI2210 LI2310 TI1012 TI1013	Capteur
autorisation	Signal

figure 6.2

Ceci permet d'aborder la description des classes.

Les figures 6.3 - 6.4 et 6.5 ci-dessous concernent respectivement une classe de vannes ordinaires, une classe Vanne fermeture par manque d'air et une classe Vanne manuelle.

```
Technologique subclass: #Vanne
  instanceVariableNames:
    'etat debit '
  classVariableNames: ''
  poolDictionaries: '' !

!Vanne class methods !
```

```
instanciation
        "renvoie un tableau de variables d'instance,
        auxquelles le concepteur donne une valeur
        à l'instanciation."
    ^#(objOperatif debitVanne)! !


!Vanne methods !

affectVar: aDict
    "affectation des variables du
    dictionnaire aDict à l'édition
    des liens"
    objOp:= aDict at: 'objOperatif'.
    debit:= aDict at: 'debitVanne'!

affiche: aColor
    "affichage de la représentation graphique
    avec la couleur aColor"
    |aForm|
    aForm:= BiColorForm fromDisplay:
        (coord translateBy: Controleur superGraphPane frame      origin).
    aForm backColor: aColor;
        displayAt: 0@0!

etat
    "répond l'état de la vanne
    relative à la valeur de la
    variable d'instance etat"
    etat = 'fcf'
    ifTrue: [^'fermée'].
    etat = 'fco'
    ifTrue: [^'ouverte'].
    ^nil!

fermeture
    "envoie un message à l'objet objOp
    indiquant que la vanne a reçu
    l'ordre de se fermer et que son débit
    en position ouverte a pour valeur debit,
        ce qui est utile en cas de défaillance.
    Répond l'état de la vanne."
    etat:= objOp nSrc: self with: debit.
    ^self etat!

ouverture
    "envoie un message à l'objet objOp
    indiquant que la vanne a reçu
    l'ordre de s'ouvrir avec la valeur
    du débit correspondant à une ouverture
        complète. Répond l'état de la vanne."
    etat:= objOp src: self with: debit.
```

```
    ^self etat!

reinit
    "initialisation des
    variables d'instance.
    Vanne en position fermée
    et mise à jour de l'affichage"
    etat:= 'fcf'.
    self affiche: 12! !
```

figure 6.3

Cette classe possède les variables et les méthodes lui permettant de définir l'état, l'instanciation, l'initialisation, l'affichage et l'envoi d'ordres ouvert et fermé aux objets opératifs.

```
Vanne subclass: #VanneFMA
  instanceVariableNames:
    'flag '
  classVariableNames: ''
  poolDictionaries: '' !

!VanneFMA class methods ! !


!VanneFMA methods !

etatParDefaut
    "Permet d'obtenir l'état par défaut
        fermée, dès que le message ouverture
        n'est plus reçue. Renvoie nil si flag
        vaut nil. Si à la fin du pas de simulation
    flag vaut un, mise à zéro de
    cette variable d'instance. Dans
    le cas contraire la vanne est fermée."
    flag isNil
    ifTrue: [^nil].
    flag = 1
    ifTrue: [flag:= 0.
        ^nil].
    self fermeture!
```

```
fermeture
    "envoie le message de fermeture.
    Mise à nil de la variable
    d'instance flag, si la vanne est
    effectivement fermée. Répond l'état
    de la vanne."
    super fermeture = 'fermée'
    ifTrue: [flag:= nil].
    ^etat!

ouverture
    "envoie le message ouverture.
    Mise à un de la variable
    d'instance flag. Répond l'état
    de la vanne."
    flag:= 1.
    ^super ouverture!

reinit
    "initialisation des
    variables d'instances.
    Vanne en position fermée,
    mise à jour de l'affichage
    et de la variable d'instance flag."
    etat:= 'fcf'.
    flag:= nil.
    self affiche: 12! !
```

figure 6.4

```
Vanne subclass: #VanneManu
  instanceVariableNames:
    'flag signal '
  classVariableNames: ''
  poolDictionaries: '' !

!VanneManu class methods ! !


!VanneManu methods !

actionOperateur
    "renvoie un tableau des possibilités
    d'intervention de l'opérateur."
    ^#(ouverture fermeture)!
```

```
etatParDefaut
    "Si à la fin du pas de simulation
    flag ne vaut pas zéro, l'instance
    envoie le message à l'objet objOp
    confirmant son ouverture avec
        la valeur du débit."
  - flag =0
    ifFalse: [objOp src: self
        with: debit * flag]!

fermeture
    "envoie un message fermeture à
    l'objet objOp. Affectation par
    l'opérateur de la variable
    d'instance flag."
    flag:= Prompter
        prompt: 'donner une valeur entre 0 et 1, 1:
         ouverture complete'
        defaultExpression: '0'.
    objOp nSrc: self with: debit * flag.!

ouverture
    "envoie un message ouverture à
    l'objet objOp. Affectation par
    l'opérateur de la variable
    d'instance flag."
    flag:= Prompter
        prompt: 'donner une valeur entre 0 et 1, 1:
         ouverture complete'
        defaultExpression: '1'.
    objOp src: self with: debit * flag.!

reinit
    "initialisation des
    variables d'instances.
    Vanne en position fermée,
    mise à jour de l'affichage
    et de la variable d'instance flag."
    self affiche: 12.
    flag:= 0! !
```

figure 6.5

Ces classes sont des sous classes de celle des Vannes. Elles introduisent une notion supplémentaire. D'une part, la fermeture en l'absence de la commande d'ouverture pour la vanne fermeture par manque d'air, cette classe utilise la notion d'état par défaut exposée dans un chapitre précédent. D'autre part, l'intervention humaine pour la vanne manuelle, l'opérateur envoie des messages d'ouverture et de fermeture en précisant la position.

Les instances de la classe moteurs et entre autre n101 communiquent leur état et leur vitesse de rotation, celle-ci peut être accélérée ou ralentie par l'opérateur, par positionnement à des valeurs prédéfinies.

La classe des pompes est mise ici comme une sous classe de celle des moteurs, à laquelle on associe un débit dépendant de la vitesse de rotation.

La classe Debitmètre est une sous-classe de la classe Compteur. Cette dernière permet toutes les opérations liées à ce type d'objet comme la remise à zéro, le comptage d'un flux depuis la remise à zéro, gel du comptage. D'autres fonctions peuvent être ajoutées, afin d'étendre la gamme de possibilités aux objets de la classe.

Les objets de la classe Capteur transmettent la valeur mesurée par un objet opératif à un objet mission.

Les instances de la classe Signal sont affichées d'une couleur différente selon qu'elles sont actives ou non.

Il faut définir les classes d'objets missions et opératifs nécessaires au développement de l'application. Les instances de ces classes sont en relation en amont et en aval avec les objets technologiques précédents.

Nous exposons en premier lieu la partie opérative du système dont la décomposition en objets est effectuée en fonction des tâches opératives, tout en gardant un caractère générique.

La décomposition est obtenue à partir de l'objet opératif principal de l'application. Dans notre exemple, l'objet opératif RA 101 de la classe Réacteur, auquel nous associons différents objets opératifs, tels que le chargement, le chauffage, le refroidissement et l'agitation, accomplissent chacun une sous tâche opérative bien définie, afin d'obtenir la tâche principale désirée. Ces objets opératifs, à l'aide éventuellement d'autres objets opératifs, doivent, en fonction des ordres reçus, effectuer une ou plusieurs tâches, qui leurs sont propres.

Une instance de la classe Réacteur sous-classe de Réservoir est capable de renseigner tout objet sur la quantité, le niveau, la conductibilité thermique, la température, le temps de dissolution et de cristallisation du produit qu'elle contient et en modifie la couleur selon l'état dissout ou cristallisé du contenu.

La variable d'instance, quantité de produit, que possède RA 101, est utile aux objets

opératifs chauffage, refroidissement et agitation. Ils peuvent en déduire ainsi les modifications apportées, en tenant compte de la quantité de produit dans le réacteur.

L'objet opératif chargement RA 101 de la classe ChargementRéacteur gère l'évolution de la quantité de produit dans le réacteur RA101 en fonction du chargement en solvant récupéré ou neuf, du chargement de la poudre, de la vidange du réacteur. Il transmet à l'objet RA 101 et au capteur LI1010 toute évolution provoquée par un débit sortant ou entrant. Dans le cas d'un débit entrant, le compteur est également informé.

Les objets chargement solvant neuf et chargement solvant récupéré sont instances de la même classe ChargementVannePompe (figure 6.6). Ils envoient le débit résultant à l'objet chargement RA 101, selon l'état des objets technologiques vannes, pompe et du niveau de la cuve alimentaire.

Ceci, nous amène à considérer le comportement opératif de ces objets technologiques au sein d'un même objet opératif. En effet, il n'est pas envisageable de concevoir plusieurs objets opératifs à ce niveau, car le degré d'interdépendance est trop important.

Prenons l'exemple du débit à la sortie de la vanne XV106. Un objet opératif Vanne, nous donnerait ce débit en fonction de l'état de la vanne et du débit entrant, qui est lui même lié à l'état de la pompe et de son débit entrant etc...

Nous voyons rapidement que la description devient lourde et nous oblige à prendre en compte des facteurs inutiles comme le débit à la sortie de la pompe.

Nous cherchons à connaître le débit à la sortie de cet ensemble nommé solvRec en fonction de l'état des objets technologiques mis en jeu ou d'une anomalie telle que l'encrassement. Mais la position de cette perturbation (tuyau entre la cuve et la pompe ou du tuyau entre la pompe et la vanne) nous importe peu.

Par contre rassembler le comportement opératif des circuits d'alimentation de solvant récupéré et de solvant neuf au sein d'un même objet n'est pas souhaitable, car cela nous impose:

d'une part, de faire intervenir des objets dont les comportements opératifs sont totalement indépendants. Un changement d'état de la pompe P232 n'a aucune incidence sur le débit à la sortie de la vanne XV106. Car nous considérons, que le tuyau commun au deux circuits d'alimentation ne présente aucun obstacle au passage du produit, même si l'alimentation de ces deux circuits devient simultanée;

d'autre part, une perte du caractère générique recherché. En effet, cet objet unique ne sera probablement pas utilisé dans d'autres applications.

Dans ces objets, est introduite une panne de la pompe survenant à l'insu de l'opérateur.

```
ChargtVanne subclass: #ChargVaPompe
  instanceVariableNames:
    'pompe pannePompe capt panneCapt debInst flag '
  classVariableNames: ''
  poolDictionaries: '' !

!ChargVaPompe class methods !

instanciation
    "Renvoie un tableau de variables d'instance,
    qui doivent être affectées d'une valeur
    à l'instanciation."
    |coll|
    coll:=OrderedCollection new.
    coll addAll: super instanciation;
        addAll: #(pompe 'capteur de niveau de la cuve
         alimentaire').
    ^coll asArray! !


!ChargVaPompe methods !

affectVar: aDict
    "Affectation des variables du dictionnaire
    aDict à l'édition des liens"
    super affectVar: aDict.
    pompe:= aDict at: 'pompe'.
    capt:= aDict at: 'capteur de niveau de la cuve
        alimentaire'!

debitInit: unDebit
    "Donne la valeur unDebit à
    la variable debTuyA"
    debTuyA:= unDebit!

etatCapt: unObjet
    "Renvoie la valeur du
    capteur de niveau de
    la cuve alimentaire."
    ^capt!
```

```
etatP: unObjet
    "Renvoie l'état de la pompe"
    unObjet = pompe
    ifTrue: [^pannePompe].
    ^nil!

forcageOperatif
    "Renvoie un tableau des forçages possibles."
    |coll|
    coll:= OrderedCollection new.
    coll addAll: super forcageOperatif;
        addAll: #('pannePompe' 'panneCapteur' 'repPompe'
         'repCapteur').
    ^coll asArray!

initFlag
    "Met la variable flag à nil."
    flag:= nil!

nSrc: unObjet with: unDebit
    "Renvoie l'état de la vanne et fixe
    la valeur du débit en fonction de l'état
    de la pompe et de la vanne, sachant que celle-ci
    normalement se ferme."
    |panne debMeth debC|
    panne:= self etatV: unObjet.
    (panne = 0 or: [panne isNil])
    ifTrue: [unObjet affiche: 12.
        ^'fcf'].
    unObjet affiche: 10.
    debMeth:= unDebit * panne
        min: (self debit: (self cuve: unObjet)).
    flag isNil
    ifFalse: [debC:= self occurrence:(debMeth min:
         debInst).
        resRecept rempl: debC]
    ifTrue: [flag:= 1.
        debInst:= debMeth].
    panne = 1
    ifTrue: [^'fco'].
    ^nil!

nSrcM: unObjet
    "Renvoie l'état de la pompe et fixe le
    débit en fonction de l'état de la
    vanne et de la pompe, sachant que celle-ci
    normalement s'arrête."
    |panne debMeth debC|
    panne:= self etatP: unObjet.
    (panne = 0 or: [panne isNil])
    ifTrue: [unObjet affiche: 12.
        ^'ea'].
```

```
    unObjet affiche: 10.
    debMeth:= unObjet vit * panne.
    flag isNil
    ifFalse: [debC:= self occurrence:(debMeth min:
        debInst).
        resRecept rempl: debC]
    ifTrue: [flag:= 1.
        debInst:= debMeth].
    panne = 1
    ifTrue: [^'em'].
    ^nil!

occurrence: unDebit
    "Vidange de la cuve alimentaire
    reservA selon unDebit, fixe la valeur
    de capt et renvoie unDebit."
    reservA pos = 0
    ifTrue: [^0].
    reservA vidange: unDebit.
    panneCapt isNil
    ifTrue: [capt:= reservA pos].
    ^unDebit!

panneCapteur
    "Met la variable panneCapt à un."
    panneCapt:= 1!

pannePompe
    "L'opérateur fixe la valeur de la variable pannePompe."
    pannePompe:= Prompter
        prompt: 'donner une valeur entre 0 et 1, 1: marche permanente'
        defaultExpression: '1'.!

reinit
    "Initialisation des variables
    d'instance."
    super reinit.
    pannePompe:= nil.
    panneCapt:= nil.
    capt:= reservA posInit.
    defaut add: #initFlag!

repCapteur
    "Met la variable panneCapt à nil,
    correspondant à la réparation du
    capteur."
    panneCapt:= nil!
```

```
repPompe
    "Met la variable pannePompe à nil,
    correspondant à la réparation de
    la pompe."
    pannePompe:= nil!

src: unObjet with: unDebit
    "Renvoie l'état de la vanne et fixe
    la valeur du débit en fonction de l'état
    de la pompe et de la vanne, sachant que celle-ci
    normalement s'ouvre."
    |panne debMeth debC|
    panne:= self etatV: unObjet.
    panne = 0
    ifTrue: [unObjet affiche: 12.
        ^'fcf'].
    unObjet affiche: 10.
    panne isNil
    ifTrue: [debMeth:= unDebit
        min: (self debit: (self cuve: unObjet))]
    ifFalse: [debMeth:= unDebit * panne
        min: (self debit: (self cuve: unObjet))].
    flag isNil
    ifFalse: [debC:= self occurrence: (debMeth min:
         debInst).
        resRecept rempl: debC]
    ifTrue: [flag:= 1.
        debInst:= debMeth].
    (panne = 1 or: [panne isNil])
    ifTrue: [^'fco'].
    ^nil!

srcM: unObjet with: unDebit
    "Renvoie l'état de la pompe et fixe
    la valeur du débit en fonction de l'état
    de la vanne et de la pompe, sachant que celle-ci
    normalement est en marche. Nous introduisons
         une panne aléatoire de la pompe."
    |panne debMeth debC|
    panne:= self etatP: unObjet.
         Number random = 1
         ifTrue: [pannePompe:= Number random \\ 100.
                 panne:= pannePompe].
    panne = 0
    ifTrue: [unObjet affiche: 12.
        ^'ea'].
    unObjet affiche: 10.
    panne isNil
    ifTrue: [debMeth:= unDebit]
    ifFalse: [debMeth:= unDebit * panne].
    flag isNil
```

```
ifFalse: [debC:= self occurrence: (debMeth min:
    debInst).
    resRecept rempl: debC]
ifTrue: [flag:= 1.
    debInst:= debMeth].
(panne = 1 or: [panne isNil])
ifTrue: [^'em'].
^nil! !
```

figure 6.6

Afin de montrer comment introduire dans la simulation des systèmes experts, aux objets de la classe Réservoir, (figure 6.7), nous avons associé un micro expert permettant de remplir les cuves vides dont l'alimentation n'est pas prise en compte dans l'application. Nous faisons appel à ce système dans la méthode "vidange:".

Dans la base de connaissances écrite en Prolog/V (figure 6.8), à chacun des réservoirs correspond le message à lui envoyer.

```
Operatif subclass: #Reservoir
  instanceVariableNames:
    'posInit coulProd pos pen '
  classVariableNames: ''
  poolDictionaries: '' !

!Reservoir class methods !

instanciation
        "renvoie un tableau de variables d'instance,
        auxquelles le concepteur donne une valeur
        à l'instanciation."
    ^#(positionLiquideInitiale couleurProduit)! !


!Reservoir methods !

affectVar: aDict
    "affectation des variables d'instance du
    dictionnaire aDict."
    posInit:= aDict at: 'positionLiquideInitiale'.
    coulProd:= aDict at: 'couleurProduit'.!
```

```
afficheNiv: aBiInteger
    "modification graphique
    de la position du niveau."
    |var int|
    ((aBiInteger at: 1) = (aBiInteger at: 2) or:
        [(aBiInteger at: 2) = 1])
    ifTrue: [^nil].
    pen down.
    var:= (aBiInteger at: 1) < (aBiInteger at: 2).
    var
    ifTrue: [pen mask: (BiColorForm color: coulProd)]
    ifFalse: [pen mask: (BiColorForm color: 15)].
    pen place: (Controleur superGraphPane frame origin +
        coord origin +
        (1 @ (coord height - (aBiInteger at: 2))));
        go: coord width - 3.
    var
    ifTrue: [
        (aBiInteger at: 1) < 1
        ifTrue: [(aBiInteger at: 2) > 2
            ifTrue: [int:= 2]
            ifFalse: [^nil]]
        ifFalse: [int:= 1].
        pen fillAt: (Controleur superGraphPane frame origin
                + coord origin +
            (coord width // 2 @
            (coord height - (aBiInteger at: 1) - int)))]
    ifFalse: [
        (aBiInteger at: 1) <= 1
        ifTrue: [^nil].
        pen fillAt: (Controleur superGraphPane frame origin
                + coord origin +
            (coord width // 2
            @ (coord height - (aBiInteger at: 1) + 1)))]!

forcageOperatif
    "renvoie un tableau des forçages opératifs
    possibles."
    ^#(fuite remplissage reparFuite)!

fuite
    "ajoute un tableau, dont les éléments
    sont le symbole fuite et la valeur val,
    à la collection defaut."
    |val|
    val:= (Prompter
        prompt: 'donner une valeur du debit'
        defaultExpression: '100').
    defaut add: (Array
        with: #vidange:
        with: val)!
```

```
pos
    "renvoie le niveau de produit."
    ^pos!

posInit
    "renvoie la position initiale de produit"
    ^pos:= posInit * coord height!

quantite
    "renvoie la quantité de produit
    dans la cuve."
    ^pos * coord width!

reinit
    "initialisation des variables d'instance."
    pos:= posInit * coord height.
    pen:= Pen new.
    pen frame: (coord translateBy:
        Controleur superGraphPane frame origin).
    pen mask: (BiColorForm color: coulProd);
        direction: 0.
    self afficheNiv: (Array with:0 with: pos)!

rempl: unDebit
    "remplissage de l'instance en fonction du débit
    unDebit, avec signalisation de débordement"
    |posInst|
    posInst:= pos + (unDebit / coord width).
    posInst >= coord height
    ifTrue: [GraphPane notifier: 'W A R N I N G'
                content: 'debordement de ',self name
                at: coord origin.
            ^pos:= coord height].
    self afficheNiv:
        (Array with: pos rounded with: posInst rounded).
    ^pos:= posInst!

remplissage
    "forçage opératif remplissage"
    self posInit;
        afficheNiv: (Array with:pos - 2 with: pos)!

reparFuite
    "forçage opératif, réparation.
    supprime l'association dont la clé
    est le symbole vidange de la
    collection defaut."
    |unElt|
    unElt:= defaut detect: [:elt|
        (elt at: 1) = #vidange:]
        ifNone: [].
    defaut remove: unElt ifAbsent: []!
```

```
vidange: unDebit
   "vidange de l'instance en fonction du débit
   unDebit. Appel au système expert pour le remplissage,
       si la cuve est vide et que l'alimentation de cette
       instance est indépendante de l'application en cours."
   |posInst array|
   pos <= 0
   ifTrue: [
               array:= Defaut new:? reservoir(name, x).
               array isNil
               ifTrue: [^pos:= 0].
               self perform: array first last.
               ^pos].
   posInst:= pos - (unDebit / coord width).
   self afficheNiv:
      (Array with: pos rounded with: posInst rounded).
   ^pos:= posInst! !
```

figure 6.7

```
Prolog subclass: #Defaut
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!Defaut class methods !


!Defaut methods !

reservoir('resRec', #remplissage).
reservoir('resNeuf', #remplissage).
```

figure 6.8

Les objets des classes Chauffage et refroidissement appliquent une variation de température, selon la conductibilité et la quantité de produit dans le réacteur. Cette variation dépend également de la température du fluide circulant dans le circuit de chauffage respectivement de refroidissement et de l'état de la vanne régulée.

La vanne régulée est décrite dans la partie opérative, car la régulation est une sous partie opérative. De plus, cette régulation est indépendante de la partie commande. La vanne régulée gère l'ouverture de la vanne par rapport à un modèle que nous avons choisi de pente linéaire.

L'instance de la classe Mélangeur adapte la vitesse de rotation à la charge, qu'elle subit. Cette charge est dûe à la quantité de produit dans le réacteur.

Il est évident, que cette décomposition en objets de la partie opérative n'est pas unique. Elle présente l'avantage d'utiliser des modèles judicieusement conçus et par conséquent génériques. Il est à noter, qu'une décomposition trop fine n'apporte rien, si ce n'est un surcroît de travail à l'instanciation et une messagerie importante. Par contre, supposer qu'un seul objet peut décrire la partie opérative, entraîne une très grande complexité et interdit l'aspect générique. En effet, il est fort peu probable qu'une autre application retrouve exactement la même partie ou sous partie opérative décrite par cet objet.

C'est pourquoi nous avons choisi une telle décomposition, où tout objet opératif peut être utilisé dans d'autres applications.

La décomposition en objets missions s'apparente à celle des objets opératifs, nous y avons introduit un objet commande (figure 6.9), qui se projette sur la mission chargement et les missions générales du réacteur. Ce sont les principales missions, éventuellement décomposées en sous missions, à effectuer sur le processus au cours d'un cycle. Ici l'objet commande est une mission hiérarchiquement supérieure.

```
Mission subclass: #CommandeRh
  instanceVariableNames:
    'charg dissol crist agit dcy '
  classVariableNames: ''
  poolDictionaries: '' !

!CommandeRh class methods !

instanciation
        "renvoie un tableau de variables d'instance,
        auxquelles le concepteur donne une valeur
        à l'instanciation."
    ^#('mission chargt' 'mission dissolution'
       'mission cristallisation' 'mission agitation'
       'bouton poussoir')! !


!CommandeRh methods !
```

```
affectVar: aDict
    "affectation des variables d'instance du
    dictionnaire aDict."
    charg:= aDict at: 'mission chargt'.
    dissol:= aDict at: 'mission dissolution'.
    crist:= aDict at: 'mission cristallisation'.
    agit:= aDict at: 'mission agitation'.
    dcy:= aDict at: 'bouton poussoir'!

execution
    "exécution du programme de commande de Rhodia"
    $£. dcy etat Z.
    charg methode: #charge.
    $£. agit methode: #marche Z.
    $£. charg finCharge Z.
    $£. dissol methode: #chauff Z.
    $£. dissol finChauff Z.
    charg desactive.
    $£. crist refroid Z.
    $£. agit fin Z.
    $£. charg vidange Z.
    self reinit!

reinit
    "Affectation de la variable methode.
    Au cours des pas de simulation suivants
    la méthode execution sera exploitée."
    self methode: #execution! !
```

figure 6.9

Nous distinguons un objet mission chargement du réacteur capable d'envoyer des messages aux objets missions chargement solvRec, chargement solvNeuf, au compteur, au signal d'autorisation d'ouverture de la vanne manuelle et au capteur de niveau du réacteur.

Nous avons choisi cette décomposition pour des raisons analogues à celles explicitées pour objets opératifs.
En effet, l'objet mission chargement gére le chargement du réacteur dans son ensemble, sans se préoccuper des objets indispensables à la mise en oeuvre. Tandis que les objets missions chargement solvRec et chargement solvNeuf (figure 6.10) font intervenir les objets technologiques indispensables à leur mission.

```
Mission subclass: #ChargSolv
  instanceVariableNames:
    'capt pompe vanne nivLim '
  classVariableNames: ''
  poolDictionaries: '' !

!ChargSolv class methods !

instanciation
        "renvoie un tableau de variables d'instance,
        auxquelles le concepteur donne une valeur
        à l'instanciation."
    ^#('capteur de niveau' pompe vanne 'valeur niveau
        mini')! !


!ChargSolv methods !

affectVar: aDict
    "affectation des variables d'instance du
    dictionnaire aDict."
    capt:= aDict at: 'capteur de niveau'.
    pompe:= aDict at: 'pompe'.
    vanne:= aDict at: 'vanne'.
    nivLim:= aDict at: 'valeur niveau mini'!

alarm
    "renvoie un booléen relatif, au niveau donné par le
        capteur et au niveau limite fixé."
    ^capt etat < nivLim!

charge
    "envoie les ordres permettant la charge
    aux objets technologiques concernés."
    pompe marche.
    vanne ouverture!

finCharge
    "envoie le message d'arrêt
    à la pompe, la vanne étant
    fermée par défaut."
    pompe arret! !
```

figure 6.10

Les objets missions dissolution et cristallisation (figure 6.11) gèrent une vanne de régulation branchée sur un circuit de vapeur respectivement d'eau, récupèrent les

informations d'un capteur de température associé au circuit et déterminent la température indispensable afin d'atteindre la mission.

```
Mission subclass: #Dissolution
  instanceVariableNames:
    'vanne delay tpsDiss finChauff capt tempe '
  classVariableNames: ''
  poolDictionaries: '' !

!Dissolution class methods !

instanciation
        "renvoie un tableau de variables d'instance,
        auxquelles le concepteur donne une valeur
        à l'instanciation."
    ^#('vanne de regulation' 'temps necessaire à la
        dissolution' 'temperature necessaire à la dissolution'
        capteur)! !


!Dissolution methods !

affectVar: aDict
    "affectation des variables d'instance du
    dictionnaire aDict."
    vanne:= aDict at: 'vanne de regulation'.
    tpsDiss:= aDict at: 'temps necessaire à la
        dissolution'.
    tempe:= aDict at: 'temperature necessaire à la
        dissolution'.
    capt:= aDict at: 'capteur'!

chauff
    "initialise les variables
    finChauff, delay
    et répond vrai"
    finChauff:= false.
    delay init: self;
        decompt: tpsDiss.
    ^true!

finChauff
    "envoie l'ordre d'ouverture
    à la vanne et la ferme si la variable d'instance
        finChauff est vrai."
    vanne ouverture.
    finChauff
    ifTrue: [vanne fermeture].
    ^finChauff!
```

```
finDelay
    "affecte à vrai la variable
    finChauff."
    finChauff:= true!

reinit
    "Affectation de la variable delay.
    Initialisation de la variable finChauff."
    delay:= Delay new.
    finChauff:= false! !
```

figure 6.11

L'agitation est conduite par un objet mission. Selon la commande, il active ou non le moteur.

Nous avons décomposé hiérarchiquement les missions à effectuer, sans oublier que les objets sont des instances de classes génériques, et qu'un objet n'accomplit qu'une seule mission simultanément.

L'objet opérateur permet les interventions sur la vanne manuelle, sur la commande, la création de perturbations telles que les anomalies ou des dépannages sur les objets technologiques ou opératifs. Suivant que l'opérateur soit déclaré expert ou non, ses réactions constituent la base de connaissances ou sont comparées à celle-ci.

La figure 6.12 représente l'organisation des objets utilisés dans cette application.

Après une dizaine d'applications types des processus par lots chimiques, la bibliothèque contient quasiment toutes les classes nécessaires au développement d'une application. Dans ce cas, le travail du concepteur est réduit à l'instanciation des différentes classes. Avant de lancer la simulation, il lui reste à construire le synoptique en disposant les objets. La figure 6.13 montre le poste de simulation.

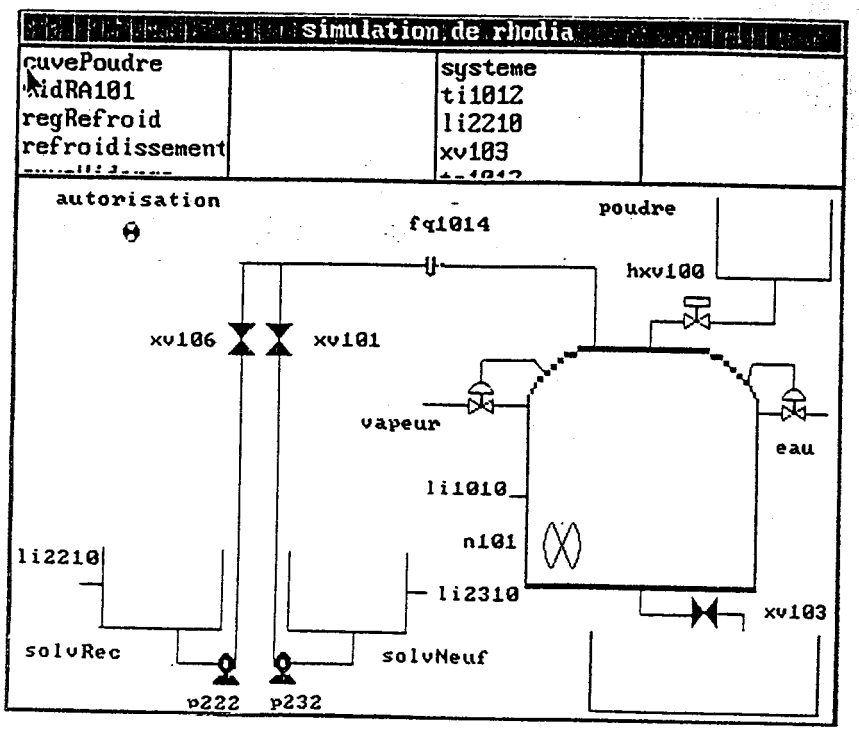

simulation de rhodia
cuvePoudre
idRA101
regRefroid
refroidissement
systeme
ti1012
li2210
xv103
autorisation
fq1014
poudre
hxv100
xv106
xv101
vapeur
eau
li1010
n101
li2210
li2310
xv103
solvRec
solvNeuf
p222
p232

figure 6.13

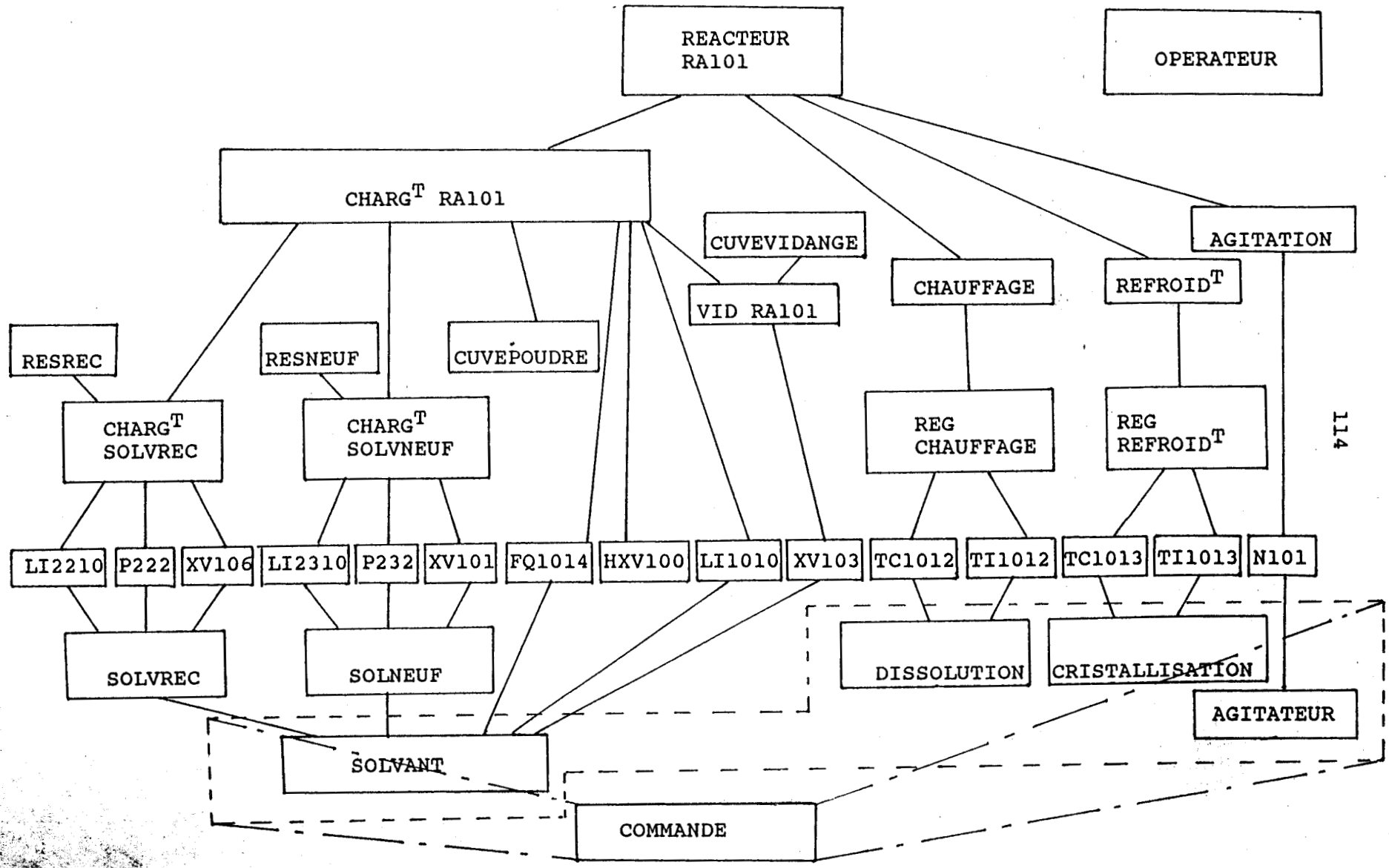

REACTEUR RA101
OPERATEUR
CHARG^T RA101
CUVEVIDANGE
AGITATION
CHAUFFAGE
REFROID^T
VID RA101
RESREC
RESNEUF
CUVEPOUDRE
CHARG^T SOLVREC
CHARG^T SOLVNEUF
REG CHAUFFAGE
REG REFROID^T
LI2210
P222
XV106
LI2310
P232
XV101
FQ1014
HXV100
LI1010
XV103
TC1012
TI1012
TC1013
TI1013
N101
SOLVREC
SOLNEUF
DISSOLUTION
CRISTALLISATION
AGITATEUR
SOLVANT
COMMANDE

figure 6.12

CONCLUSION

CONCLUSION

Au cours de ce chapitre, nous avons tenté de proposer au lecteur une philosophie de la décomposition d'un processus sur un exemple emprunté aux industries chimiques.

L'outil, que nous utilisons, présente l'avantage de pouvoir s'enrichir avec l'expérience à la condition d'utiliser des décompositions assurant une réutilisabilité des composants. Par conséquent, il faut créer des classes proches des phénomènes et ayant un sens générique.

Nous avons, par ailleurs, montré la possibilité de connecter à la modélisation d'un objet une base de connaissances écrite en Prolog et destinée à résoudre certains problèmes de conduite.

Le travail présenté dans ce mémoire a montré l'intérêt de l'approche objet pour la décomposition des systèmes de production et en particulier des processus par lot.

La modélisation d'un tel système est obtenue par la représentation des missions, de la partie opérative, et des objets technologiques.

La simulation envisagée permet de satisfaire plusieurs objectifs. D'une part, il est important pour un concepteur d'automatisation de pouvoir effectuer une validation du cahier des charges de l'installation. Cette validation doit tenir compte des anomalies possibles de la partie opérative ou des objets technologiques. L'introduction des anomalies se fait au niveau des objets opératifs soit par forçage de la part d'un opérateur de simulation soit par création d'événements relativement à une échelle de temps aléatoire. D'autre part, notre outil permet de considérer le comportement des opérateurs de conduite dans toutes les situations difficiles rencontrées sur le site, afin de le familiariser aux événements survenant en cas de pannes, au démarrage...

Pour réaliser ces objectifs, l'usage du langage Smalltalk est particulièrement intéressant à cause de sa structure qui coïncide parfaitement avec notre décomposition d'un système de production et à cause de son environnement de programmation particulièrement performant. Toutefois certaines extensions ont du être introduites pour modifier le comportement du cycle de base de Smalltalk vis à vis de l'usage de la souris et pour permettre une mise en oeuvre de la notion de parallélisme.

Le couplage avec Prolog a permis par ailleurs de définir facilement la notion d'expert de conduite, base de connaissance représentative du comportement d'un opérateur expérimenté.

Le problème n'est pas complètement résolu et il reste à approfondir en relation avec des industriels le moyen de formaliser les connaissances. Par ailleurs, notre maquette a été réalisée sur IBM PC et ne présente donc pas un niveau de performance convenable dans tous les cas à cause de la taille mémoire et de la vitesse du processeur. Une adaptation sur un matériel beaucoup plus puissant devrait fournir un outil d'autant plus remarquable qu'il ne demande qu'à s'enrichir.

Nous espérons que ce travail sera une contribution au développement de l'usage des techniques d'objet auxquelles nous croyons. Les objets industriels ont de nombreuses facettes et nous sommes conscients que ce mémoire n'a pu faire le tour des problèmes. Nous souhaitons qu'il puisse constituer le point de départ de nombreuses autres études.

BIBLIOGRAPHIE

[ADE]
ADEPA
GEMMA - Guide d'Etude des Modes de Marche et d'Arrêt.
Edition 2 de l'ADEPA.

[ALB 82]
J. Albukerque
Spécification et validation d'automatismes logiques interconnectés.
Thèse de Doctorat de 3ième cycle: EEA Automatique
Toulouse III 1982 (2746).

[BAU 86]
B. Baudel
Génie logiciel industriel:
Contribution à la définition et à la mise en
oeuvre d'un poste d'automatisation et
d'instrumentation.
Thèse de Doctorat
Université des Sciences et Techniques
de Lille Flandres Artois Avril 1986.

[BAU 87/1]
B. Baudel, E. Cantegrit, J.M. Toulotte
Smalltalk and simulation of batch processes
Congrés I.M.A.C.S. Barcelona June 1987.

[BAU 87/2]
B. Baudel, E. Cantegrit, J.M. Toulotte
Utilisation des langages objets pour la simulation des processus par lots
Congrés I.A.S.T.E.D. Paris Juin 1987.

[BAU 87/3]
B. Baudel, E. Cantegrit, J.M. Toulotte
Utilisation de la notion d'objets pour
la modélisation et la simulation des processus par lots.
A paraître.

[BEN 87]
G. Benchimol
Introduction des systèmes experts dans l'entreprise.
Interfaces AFCET mars 1987.

[BIG 84]
Bigre+Globule
Actes des deuxièmes journées d'étude
du groupe de travail AFCET Informatique
sur langages orientés objets
Bigre n° 41 Novembre 1984.

[BIG 86]
Bigre+Globule
Actes des journées AFCET Informatique
langages orientés objet
Bigre n° 48 Janvier 1986.

[COI 82]
P. Cointe
Une réalisation de Smalltalk en VLisp
Techniques et Sciences Informatiques
Vol 1 n° 4 Juillet-Aout 1982

[BOO 86]
G.Booch
Object Oriented Development
IEEE Transactions on Software Engineering
Vol SE-12 Number February 1986.

[BOS 79]
J.C. Bossy, P. Brard, P. Faugère, C. Merlaud
Le grafcet
Sa pratique et ses applications. Educalivre.

[COO]
C.Jullien
Le système le_cool
ACT informatique.

[COO 86]
S. Cook
Languages and Object Oriented Programming
Software Engineering Journal March 1986.

[COX 86]
Brad J. Cox
Object Oriented Programming
An Evolutionary Approach
Addison Wesley 10393 1986.

[ECO 87]
ECOOP'87
European conference on
object oriented programming
AFCET Bigre n° 54 Juin 1987.

[DAL 83]
Y. Dallery, H. Deneux, R. David
Recherche d'une même base de description en vue de la simulation et de la commande d'un atelier flexible, utilisation du grafcet
Congrés automatique AFCET , Besançon Novembre 1983.

[DEF 86]
J. Defrenne
Modélisation de la partie opérative
impact sur la sécurité et la maintenance
des automatismes à évolution séquencielle.
Thèse Docteur ès Sciences Physiques
Université des Sciences et Techniques de Lille Flandres Artois
Fevrier 1986 n° d'ordre 672.

[DEN 83]
H. Deneux, R. David
Spécification et mise en oeuvre d'automates
interconnectés à l'aide du grafcet
R.A.I.R.O. Automatic Systems Analysis and Control.
Vol 17 n° 4 1983 p 339-358.

[DES 86]
J. Desquilbet
LROI:
Une extension de Le_Lisp vers un langage
de représentation d'objet
Rapport de stage I.D.N. 3ième Année et
D.E.A. de productique Juin 1986.

[DIG 86]
Digitalk
Smalltalk/V Tutorial and Programming
Handbook (Digitalk inc., 1986).

[DUF 86]
C.B. Duff
Designing an efficient language
Byte. August 1986.

[FER 85]
J. Ferber
Les langages objets dans les systèmes experts.
Bigre+globule n° 47 Décembre 1985.

[FRA 87]
J.P. Frachet
Une introduction au génie automatique:
Faisabilité d'une chaine intégrée d'outils CAO
pour la conception et l'exploitation des
machines automatiques industrielles.
Thèse Docteur ès Sciences Physiques
Université de Nancy Juillet 1987.

[GOL 83]
A. Goldberg, D. Robson
Smalltalk 80,
The language and its implementation
Addison Wesley 1983.

[GOL 84]
A. Goldberg
Smalltalk 80,
The interactive programming environment
Addison Wesley 1984.

[GRE 85]
GREPA (ouvrage collectif)
Le grafcet: de nouveaux concepts.
Cepadues Editions 1985.

[IEE 84]
IEEE
Abstractions
mapping software complexity
IEEE Software Vol 1 number 4 October 1984.

[IGL 82]
Institut du Génie Logiciel.
Introduction à SADT.
IGL, 1982.

[KAE 86]
T. Kaehler and D. Patterson
A Small Taste of Smalltalk
Byte. August 1986.

[KRA 83]
G. Krasner
Smalltalk 80
bits of history, words of advice
Addison Wesley 1983.

[LIE 81]
H. Lieberman
A preview of Act1
Massachusetts Institute of Technology
Artificial Intelligence Laboratory June 1981.

[LOU 87]
R.Loubeyre, C. Melin
Galopin: An object oriented system for automatic
start-up of continuous chemical process plants.
ECOOP'87 bigre n° 54 Juin 1987.

[MEY 80]
B. Meyer
Quelques concepts importants
des langages de programmation modernes,
et leur expression en Simula 67.
EDF bulletin de la direction des études et recherches
série C Mathématiques, Informatique n° 1, 1980 p 89-150.

[MOT 84]
L. Motus, V.S. Tchugonov and N.I. Artemyeva
Selection of formal model for a batch chemical process control system's software specification.

[NOY 84]
Y. Noyelle
Un système Prolog écrit en Smalltalk 80
Bigre+Globule n° 41 Novembre 1984.

[PAS 86]
Geoffrey. A. Pascoe
Elements of object oriented programming
Byte. August 1986.

[POU]
D. Pountain
Object Oriented Forth
Journal of Forth Application and Research
Vol 3 Number 3.

[POU 86]
D. Pountain
Object Oriented Forth
A new mechanism for designing and writing Forth programs
Byte. August 1986.

[R.P 85]
Rhône-Poulenc.
Documentation d'étude et de réalisation
outils et méthodes de spécification
Avant Projet EASYMITIS.

[SCH 86]
Kurt J. Schmucker
Object Oriented Programming
for the Macintosh
Hayden Books 6565-5 1986.

[STR]
B. Stroustrup
What is "object oriented programming" ?
AT&T Bell Laboratories, Murray Hill, New-Jersey 07974, USA.

[STR 87]
B. Stroustrup
The C^{++} Programming Language
Addison Wesley 12078 1986.

[TOU 82]

J.M. Toulotte S. Thelliez
Applications industrielles du grafcet.
Eyrolles 1982.

[VAL 83]

R.Valette, M. Courvoisier, J.M. Bigou
Les réseaux d'automates: Analyse de la coopération
L.A.A.S.-C.N.R.S.

[VAL 87]

B. Vallespir
Exploitation des systèmes de production
duscrets-continus:
Contribution à une méthode de conception.
Thèse de Doctorat
Université de Bordeaux1 Novembre 1987.

[WIE 84]

G.Wiederhold
Knowledge and Database Management
IEEE Software Vol 1 number 1 January 1986.