THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le grade de

Docteur en Productique :

Automatique et Informatique Industrielle

par

Mohammed BENNANI

Mastère I.D.N

Maître E.E.A

Un Superviseur à Base de Connaissances en Synthèse d'Asservissement : SuBaCoSyAs

Soutenue le 21 Décembre 1990 devant le jury d'examen :

M.	*P. BORNE*	*Président*
Mme	*G. DAUPHIN-TANGUY*	*Rapporteur*
M.	*C. MELIN*	*Rapporteur*
M.	*D. MEIZEL*	*Examinateur*
M.	*D. CORBEEL*	*Examinateur*
M.	*J. P. DELAHAYE*	*Examinateur*
M.	*F. DELEBECQUE*	*Examinateur*

Directeur de Thèse : D. MEIZEL, Professeur à l'U.T.C

A mes parents

AVANT PROPOS

Le travail présenté dans ce mémoire a été effectué au laboratoire d'Automatique et d'Informatique Industrielle de l'IDN, sous la direction de Monsieur D. MEIZEL.

Nous sommes particulièrement reconnaissant envers Monsieur D. MEIZEL, Professeur à l'UTC, Directeur de Recherche. Sa compétence, le temps, l'intérêt qu'il a bien voulu nous consacrer et l'amitié qu'il nous a toujours témoignée nous ont permis de surmonter les périodes difficiles et d'achever dans de meilleures conditions les travaux entrepris.

Nous remercions vivement Monsieur le Professeur P. BORNE, Directeur du LAII et Responsable scientifique de l'IDN, de nous avoir accueilli au sein de son équipe et d'avoir accepté de présider le jury de thèse.

Tous nos remerciements vont à Madame G. DAUPHIN-TANGUY, Professeur à l'IDN, ainsi qu'à Monsieur C. MELIN, Professeur à l'UTC, pour l'intérêt qu'ils ont bien voulu porter à notre travail et d'avoir accepté de le juger en tant que rapporteur.

Nous tenons à exprimer nos remerciements à MM. D. CORBEEL, Maître de conférence à l'IDN, J.P. DELAHAYE, Professeur à l'USTLFA et F. DELEBECQUE, Directeur de recherche à l'INRIA, pour l'honneur qu'ils ont bien voulu nous accorder en acceptant de participer, comme examinateurs, au jury de thèse.

Nous tenons également à exprimer notre gratitude à Mr. D. CORBEEL, Maître de conférence à l'IDN, qui lors des conversations que nous avons eues au début de ce travail, a su nous montrer des chemins féconds. De même, nous sommes particulièrement redevables à MM. J.P. DELAHAYE, Professeur à l'USTLFA et F. RECHENMANN, Directeur de recherche à l'INRIA/ARTEMIS, d'avoir eu la gentillesse de nous prêter les versions sources de "SPT" et "SHIRKA" sans lesquelles ce travail aurait été beaucoup plus pénible.

En outre, que Mr. F. DELEBECQUE, Directeur de recherche à l'INRIA, soit ici remercié pour l'attention qu'il a apportée à ce travail et pour le développement de BASILE qui constitue une "boîte à outil" d'Automatique très précieuse.

Nous tenons enfin à témoigner notre profonde amitié à tous les chercheurs du Laboratoire d'Automatique et d'Informatique Industrielle de l'IDN, pour la sympathie et l'esprit d'équipe qu'ils entretiennent au sein du laboratoire.

TABLE DES MATIERES

INTRODUCTION GENERALE 19

A PANORAMA DES OUTILS INFORMATIQUES POUR LA SYNTHESE DE COMMANDE DES SYSTEMES D'ETATS CONTINUS 23

Table des matières du chapitre A

I **LES SYSTEMES DE C S C A O** 29

1. INTRODUCTION 29

2. HISTORIQUE ET CLASSE DES PROBLEMES TRAITES PAR C S C A O 30

3. CARACTERISTIQUES GENERALES DES SYSTEMES DE C S C A O 31

4. EXEMPLES DE SYSTEMES DE C S C A O 32

5. CONCLUSION 36

II **GENERALITES SUR LES SYSTEMES A BASE DE CONNAISSANCES ET LES SYSTEMES EXPERTS** 38

1. LES MODULES D'INTERFACE AVEC L'EXTERIEUR 38

2. LA BASE DE CONNAISSANCES 39

3. MOTEUR D'INFERENCE 40

4. AUTRES MODES DE REPRESENTATION DE CONNAISSANCES 45

III **UNE PREMIERE TENTATIVE D'UTILISATION DE SYSTEME A BASE DE CONNAISSANCES EN C S C A O** 48

1. LES ETAPES D'UNE ETUDE D'AUTOMATISATION 48

2. STRUCTURE DU SYSTEME EXPERT 49

3. CONCLUSION 54

IV **CONCLUSION** 55

B FORMULATION DES PROBLEMES D'AUTOMATIQUE 57

Table des matières du chapitre B

I **NOTION DE SIGNAL** 63

II **CARACTERISATION DES SYSTEMES DYNAMIQUES** 67

1. INTRODUCTION 67
2. INTERFACES 67
3. STRUCTURATION DES PROCESSUS 68
4. CALCUL FORMEL DES TRANSFERTS ENTREE-SORTIE 75
5. CONCLUSION 80

III **SYNTHESE DE LOIS DE COMMANDE** 81

1. INTRODUCTION 81
2. STRUCTURE DE COMMANDE 82
3. CONTROLE EN TEMPS DISCRET OU TEMPS CONTINU 86
4. PERFORMANCES EN REGIME PERMANENT 88
5. COMPORTEMENT DYNAMIQUE 94
6. ROBUSTESSE 102
7. CONCLUSION 103

IV **CONCLUSION** 104

C SPECIFICATION ET REALISATION D'UN SYSTEME EXPERT EN C S C A O 105

Table des matières du chapitre C

I **INTRODUCTION** 109

II **LES OUTILS INFORMATIQUES POUR LA REALISATION DU SYSTEME EXPERT** 110

1. DESCRIPTION DE SPT 110

2. DESCRIPTION DE SHIRKA 114

3. CONCLUSION 122

III **DESCRIPTION DE LA BASE DE CONNAISSANCES DU SYSTEME EXPERT** 123

1. MODULE DE MODELISATION 123

2. MODULE DE SYNTHESE 138

3. TRANSFERT DES DONNEES DANS LA BASE DE CONNAISSANCES 149

4. EXEMPLES ET MODE D'EMPLOI 151

IV **CONCLUSION** 163

CONCLUSION GENERALE 165

REFERENCES BIBLIOGRAPHIQUES 169

ANNEXES 177

Annexe 1 : LISTE DES SYSTEMES EXISTANTS DE C S C A O 179

Annexe 2 : TABLE DES TRANSFORMEES EN P ET EN Z 191

Annexe 3 : PRINCIPE DU MODELE INTERNE : Cas monovariable 193

Annexe 4 : LISTE DES FACETTES SHIRKA 199

Annexe 5 : LES FONCTIONS SHIRKA 203

INTRODUCTION GENERALE

Depuis le milieu des années 80, la C S C A O (Conception de Système de Commande Assisté par Ordinateur) joue un rôle important dans la diffusion de la synthèse de lois de commande "moderne" dans le milieu industriel. Ces systèmes construits autour de progiciels numériques sont utiles pour analyser le processus étudié et proposer une structure de commande appropriée.

Notre projet appartient à ce courant. Sa principale originalité réside dans la prise en compte du degré de description du processus étudié dans l'élaboration des lois de commandes appropriées.

Dans ce mémoire nous présentons les différentes étapes de la réalisation d'un système à base de connaissances, d'aide à la synthèse de lois de commande, en utilisant à la fois des règles de production et des procédures numériques au travers d'une modélisation à base d'objets. Cette représentation permet des descriptions de problèmes à des niveaux de détail plus ou moins fin.

Au premier chapitre nous présentons les différents outils informatiques utilisés pour la synthèse de commande des systèmes d'états continus. Aussi, nous décrivons les différentes parties formant un système à base de connaissances. La dernière partie est consacrée à la synthèse bibliographique d'un des premiers essais d'utilisation de système à base de connaissances en C S C A O.

Au second chapitre, nous décrivons les outils automatiques nécessaires à une synthèse de lois de commande. Ainsi, nous présentons la notion de signal et nous caractérisons les systèmes dynamiques. En dernier lieu, nous rappelons les différentes étapes d'une synthèse de lois de commande.

Le dernier chapitre est consacré à la description des outils informatiques utilisés dans notre prototype. Aussi, nous définissons les différents objets, faits et règles de production formant notre base de connaissances. Dans la dernière partie de ce chapitre, des exemples d'application permettent de présenter le mode de fonctionnement de notre système.

Chapitre - A -

PANORAMA DES OUTILS INFORMATIQUES POUR LA SYNTHESE DE COMMANDE DES SYSTEMES D'ETATS CONTINUS

I LES SYSTEMES DE C S C A O 29

1. INTRODUCTION 29

2. HISTORIQUE ET CLASSE DES PROBLEMES TRAITES PAR C S C A O 30

2.1. EVOLUTION DES SYSTEMES DE C S C A O 30

2.2. CLASSE DES PROBLEMES TRAITES 30

3. CARACTERISTIQUES GENERALES DES SYSTEMES DE C S C A O 31

3.1. PRINCIPALES CARACTERISTIQUES 31

3.2. MODES D'UTILISATION 31

4. EXEMPLES DE SYSTEMES DE C S C A O 32

4.1. CLADP 32

4.2. MATLAB 33

4.3. LE PROGICIEL BASILE 34

4.3.1. Principales caractéristiques du système BASILE 34

4.3.1.1. *Structures de données* 34
4.3.1.2. *Fonctions universelles* 35
4.3.1.3. *Notion de macro* 35

4.3.2. Conclusion 36

5. CONCLUSION 36

II GENERALITES SUR LES SYSTEMES A BASE DE CONNAISSANCES ET LES SYSTEMES EXPERTS 38

1. LES MODULES D'INTERFACE AVEC L'EXTERIEUR 38

2. LA BASE DE CONNAISSANCES 39

3. MOTEUR D'INFERENCE 40

3.1. ORDRE DU MOTEUR D'INFERENCE 40

3.2. LES MODES D'INFERENCE 40

3.2.1. Chaînage avant 40

3.2.2. Chaînage arrière 40

3.2.3. Chaînage mixte 41

3.2.4. Exemple 41

3.3. CYCLE DE BASE D'UN MOTEUR D'INFERENCE 43

3.3.1. Restriction 43

3.3.2. Filtrage 43

3.3.3. Résolution de conflits 44

3.3.4. Exécution 44

4. AUTRES MODES DE REPRESENTATION DE CONNAISSANCES 45

4.1. REPRESENTATION CENTREE-OBJETS 45

4.1.1. Notion de spécialisation et d'héritage 45

4.1.2. Notion d'instance 46

4.1.3. Notion d'inférence 46

4.1.3.1. *Attachement procédural* 46

4.1.3.2. *Filtre et filtrage* 46

4.2. REPRESENTATION HYBRIDE 47

4.3. CONCLUSION 47

III UNE PREMIERE TENTATIVE D'UTILISATION DE SYSTEME A BASE DE CONNAISSANCES EN C S C A O 48

1. LES ETAPES D'UNE ETUDE D'AUTOMATISATION 48

2. STRUCTURE DU SYSTEME EXPERT 49

2.1. LA STRUCTURE DE LA BASE DE CONNAISSANCES 50

2.2. LA REPRESENTATION DES CONNAISSANCES 51

2.2.1. Les règles 52

2.2.2. Liste de faits 53

3. CONCLUSION 54

IV CONCLUSION 55

I LES SYSTEMES DE C S C A O

1. INTRODUCTION

Le développement de l'usage de l'informatique est étroitement lié aux progrès technologiques qui permettent :

- d'augmenter la puissance de calcul et la capacité de stockage des ordinateurs tout en réduisant leur coût et leur taille ;

- d'améliorer la communication entre l'homme et la machine.

L'utilisation de calculateurs n'est plus limitée aux tâches "administratives" (gestion, comptabilité, planification) mais se généralise dans toutes les étapes de processus de production industrielle (conception, dessin, fabrication des produits, contrôle des procédés, etc...). Cette diffusion fait l'objet de nouvelles techniques que l'on peut rassembler sous le nom générique de "X A O" : D A O (Dessin Assisté par Ordinateur), F A O (Fabrication Assistée par Ordinateur), C A O (Conception Assistée par Ordinateur), etc. Tous les domaines (électronique, électrotechnique, mécanique, ...) sont dores et déjà des champs d'applications de ces techniques.

Nous nous intéressons ici à la conception de lois de commande pour les systèmes d'état continu où on a défini la C S C A O (Conception de Systèmes de Commande Assistée par Ordinateur).

L'objet de la C S C A O est d'aider l'automaticien dans sa démarche, de la modélisation du procédé à l'implantation de l'algorithme de commande. On se rend compte à l'énoncé des quatre phases de travail (modélisation, analyse du modèle, synthèse du système de commande et implantation de l'algorithme de commande) que l'ensemble de l'activité demande une importante quantité de calculs, le plus souvent complexes, et la bonne connaissance de nombreuses théories.

Le concepteur a donc besoin de l'aide d'un outil informatique qui lui permet d'utiliser des résultats d'algorithmes complexes sans avoir à les programmer, le libérant ainsi des tâches qui sortent de son domaine de compétence. L'élaboration des systèmes de C S C A O nécessite en effet des compétences en automatique, en informatique et en mathématiques appliquées (analyse numérique, optimisation). Le développement de la C S C A O est récent (fin des années 70) et d'énormes progrès ont été enregistrés, aboutissant rapidement à une quasi-standardisation des possibilités et des outils.

2. HISTORIQUE ET CLASSE DES PROBLEMES TRAITES PAR C S C A O

Les automaticiens ont utilisé des calculateurs bien avant l'apparition des ordinateurs que nous connaissons aujourd'hui. Il s'agissait de simuler les modèles mathématiques des procédés qu'on voulait contrôler. Ses premiers outils pendant la période 1940-1970 furent des simulateurs analogiques construits autour d'amplificateurs opérationnels. Cependant, les calculs ou les graphiques effectués manuellement se révélaient insuffisants pour l'étude des systèmes complexes.

Avec l'évolution de l'électronique, le simulateur analogique a presque disparu pour laisser place à la simulation numérique. Cette dernière était souvent enrichie d'une synthèse de commande limitée à des algorithmes qui comprenaient généralement une seule méthode. Enfin, des progrès en flexibilité (possibilité de modifier des paramètres), en interactivité (utilisation de menus, interfaces conviviales de dialogue), etc... ont abouti aux systèmes de C S C A O.

2.1. EVOLUTION DES OUTILS DE C S C A O (voir annexe 1)

Les premiers programmes spécifiques à l'automatique datent du milieu des années 60. Ils reprenaient les techniques bien connues pour les systèmes monovariables (tracé automatique des diagrammes de Bode, de Nyquist, ...).

C'est au début de la décennie suivante [ROSENBROCK 74] que furent développés les premiers véritables outils de C S C A O incluant des outils de simulation et des techniques de commande comme la résolution de l'équation de Riccati matricielle. L'interactivité Homme/Machine n'était pas très grande et le fonctionnement était plutôt de type "BATCH".

Vers 1974, les programmes interactifs firent leur apparition avec l'utilisation de terminaux graphiques. Depuis, le développement de la C S C A O s'est accéléré, grâce aux progrès des outils informatiques (algorithmes plus robustes et plus efficaces). [POWER 76]

2.2. CLASSE DES PROBLEMES TRAITES

Il n'existe pas de logiciel universel de C S C A O pouvant résoudre les différents problèmes par toutes les techniques applicables. Le champ d'investigation est très large ; les méthodes sont nombreuses et les programmes reflètent souvent les spécialités des équipes qui les ont conçus. Les modèles des systèmes traités peuvent être continus ou discrets, linéaires

ou non linéaires, monovariables ou multivariables, opérationnels ou d'état, etc. Suivant les spectres des problèmes à résoudre, un ou plusieurs progiciels seront nécessaires.

3. CARACTERISTIQUES GENERALES DES SYSTEMES DE C S C A O

3.1. PRINCIPALES CARACTERISTIQUES [ASTRÖM 84 a et c]

Les principales caractéristiques pour la conception d'un système de commande assistée par ordinateur sont les suivantes :

- une structure de données adaptée aux problèmes d'automatique pour les formaliser et les traiter de la façon la plus simple possible ; les matrices, vecteurs, polynômes et les fonctions (dans le cas de problèmes non linéaires) sont des structures élémentaires typiques ;

- une bibliothèque numérique adaptée et composée de méthodes robustes et éprouvées :

  * équations de Lyapunov, de Riccati, calcul de vecteurs propres, opérations classiques ;

  * optimisation (pour la synthèse paramétrique, et l'identification) ;

  * simulation (résolveur d'équations différentielles) ;

  * identification (combinaison de la simulation et de l'optimisation).

- des fonctions graphiques (de base) :

  * résultats de simulation ;

  * validation de contrôleurs ;

  * utilisation de méthodes graphiques.

3.2. MODES D'UTILISATION

Le mode d'utilisation est la nature de dialogue qui s'instaure entre le système et l'utilisateur. Le plus simple est le mode "**question-réponse**", où le programme garde l'initiative. Le plus puissant est celui où l'utilisateur

contrôle tout le déroulement de la tâche en la définissant par un ensemble de commandes (**langage de commande**). Entre ces deux modes, on trouve le dialogue par choix dans des menus déroulants, ou un mélange de langage de commande avec un mode question-réponse pour des tâches élémentaires [ASTRÖM 84 a et c].

On peut également citer le mode de **programmation par blocs** qui est lui aussi très utilisé. Dans TUTSIM, par exemple, on définit un système comme un réseau de blocs élémentaires (intégrateurs, gains, sommateurs, etc...). On peut ainsi simuler les comportements qui nous intéressent sans exprimer nécessairement le modèle d'état ou la fonction de transfert du système global.

Il n'y a pas de bon ou de mauvais choix en la matière ; chacun des modes s'adresse à un type d'utilisateur différent. L'utilisateur débutant ou occasionnel devra être guidé et préférera donc un mode de type "question-réponse" ou menu, avec possibilité d'obtenir de l'aide ou des explications de la part du programme (**HELP**). L'utilisateur confirmé connaît bien la logique d'enchaînement des différentes parties du programme et veut pouvoir exprimer de manière concise l'énoncé de la solution d'un problème complexe. Le langage de commande est alors le moyen de communication le plus approprié. Il permet même, pour de longues et coûteuses résolutions, un fonctionnement non interactif du type "**BATCH**".

De façon intermédiaire, le type de programmation le plus répandu dans l'industrie est la programmation par blocs (Micro Z de contrôle Bailey, Masterpiece de Asea). Ce mode de programmation est bien compris des techniciens en régulation et en conduite de processus. Initialement conçu pour des applications monovariables, la prise en compte de problèmes plus complexes a entraîné la possibilité de structurer les schémas par la définition de macro-blocs [FENET 86].

Le meilleur système est finalement celui qui permet tous ces modes et s'adresse ainsi au plus grand nombre d'utilisateurs [LAPORTE 85].

4. EXEMPLES DE SYSTEMES DE C S C A O

Sans être exhaustive, la liste suivante est significative :

4.1. CLADP (the Cambridge Linear Analysis and Design Programs)

Ce progiciel permet la synthèse, l'analyse essentiellement fréquentielle

de systèmes linéaires, ainsi que leur simulation [MACIEJOWSKI et MACFARLANE 82].

4.2. MATLAB (MATrix LABoratory)

C'est un logiciel éprouvé d'algèbre linéaire, il définit (presque) un langage de programmation interactif. Sa syntaxe, proche du langage mathématique usuel, le rend très attractif pour les ingénieurs en automatique. Sa structure de donnée est basée sur le type "Matrice à cœfficients complexes" [MOLER 82/86].

ARBRE GENEALOGIQUE DE LA FAMILLE DE MATLAB
[RIMVALL 88]

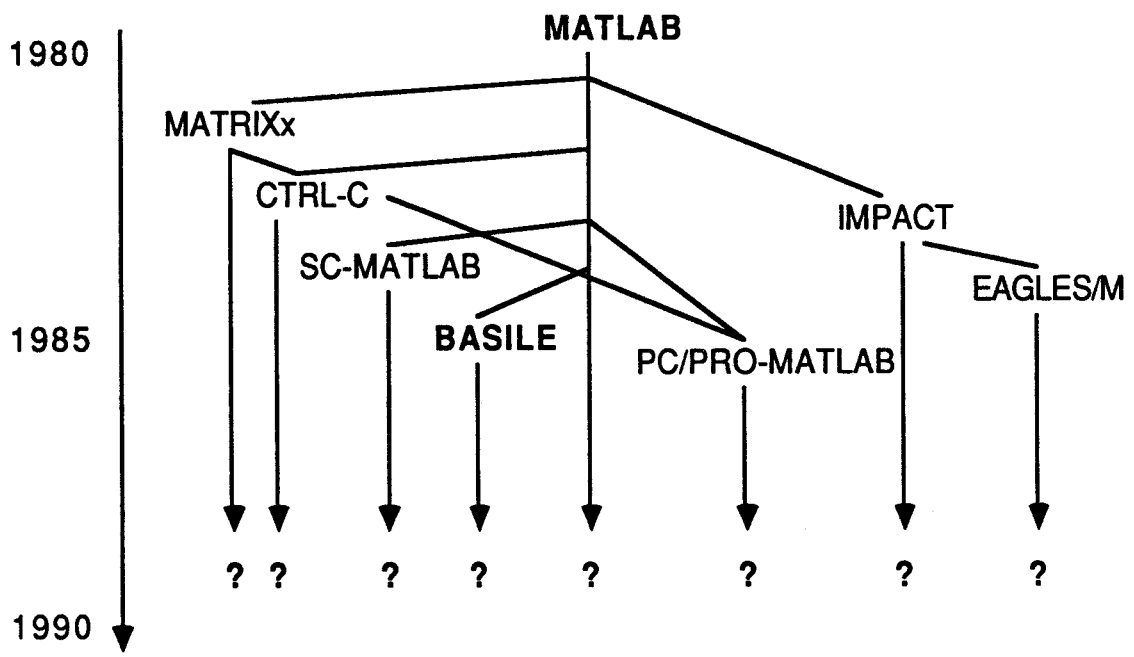


Fig. A.I.1.

Comme l'indique la figure A.I.1., MATLAB a donné naissance à de nombreux produits.

Nous proposons maintenant d'analyser l'un d'eux : BASILE d'INRIA/SIMULOG.

4.3. LE PROGICIEL BASILE [DELEBECQUE 85/87]

C'est un outil interactif de C A O en automatique. Ce système met en œuvre un langage de programmation complet travaillant sur des bases de données adaptées aux problèmes d'automatique. Il s'adresse donc plutôt à une classe d'utilisateurs possédant de bonnes connaissances en automatique.

Sa conception lui permet de satisfaire les besoins demandés à un système de C S C A O.

Il utilise un **langage de programmation de haut niveau.**

4.3.1. Principales caractéristiques du système BASILE

Il est construit autour de MATLAB.

Pour répondre aux besoins d'un système universel, les auteurs ont été amenés à l'améliorer avec :

- une base de données adaptée ;
- de puissants outils numériques supplémentaires à usage universel ;
- la définition de macro-instructions.

Ces choix ont été faits pour traiter efficacement l'énorme variété de problèmes et d'algorithmes d'automatique. Par exemple, les systèmes linéaires peuvent être décrits par différentes représentations (espace d'état, matrice de transfert ...).

Cette universalité nécessite que l'utilisateur potentiel ait une sérieuse expérience en automatique. Il doit choisir la présentation de son problème et un enchaînement de procédures pour le résoudre ; le système lui fournit tous les outils nécessaires.

4.3.1.1. *Structures de données*

Les concepteurs ont modifié complètement les structures de données de MATLAB dans le but de définir d'autres genres de variables que les matrices complexes. En particulier, ils ont introduit les types "matrice polynomiale ou rationnelle", et "macros" ... On utilise les opérateurs " + ", " - ", " * ", " / ", " exp. ", ... avec un sens qui dépend du type de variables.

La gestion des données s'articule autour de la notion de contexte (ou d'environnement). Un contexte est un ensemble d'objets (nom, type, valeur) connus par le système à un instant donné. Ce contexte est le résultat de l'exécution d'un travail. Il est donc particulièrement utile de le sauvegarder et de pouvoir le restaurer. On gagne ainsi du temps par rapport à une démarche complètement procédurale. Cela est possible en BASILE par ces commandes réciproques : SAVE & LOAD.

4.3.1.2. *Fonctions universelles*

Les fonctions disponibles en BASILE couvrent la plupart des algorithmes d'algèbre linéaire et les outils utilisés sont :

* les équations de Lyapunov et Riccati ;

* l'exponentielle de matrice ;

* les caractéristiques fréquentielles (lieux de Bode, Black, Nyquist) ;

* les calculs des zéros d'un polynôme ;

* les algorithmes de simulation (résolveurs d'équations différentielles) ;

* les algorithmes d'optimisation ;

* les graphiques.

Les auteurs ont pris beaucoup de soin à réaliser des fonctions à usage universel. Ainsi, l'utilisation de paramètres optionnels (ou de valeurs par défaut des paramètres) permet un usage simplifié des fonctions dans les cas les plus courants tout en gardant la richesse d'utilisation de ces fonctions. Par exemple l'utilisation simple de (plot (y)) trace le graphe {y(t), t variant de 1 à la taille de y} avec des échelles standards et sans aucune légende. 57 paramètres permettent par ailleurs d'obtenir tous les types de graphiques. La principale difficulté réside alors dans le bon choix des paramètres optionnels.

De plus, l'ensemble des fonctions est extensible par la notion de macro.

4.3.1.3. *Notion de macro*

Pour assurer l'évolution du langage BASILE, on a proposé la notion de macro qui a pour mission de définir par programmation une nouvelle

fonction. Ces macros admettent n'importe quel type de variable comme paramètre d'entrée/sortie et peuvent se référer à n'importe quelle variable définie préalablement. Pour l'utilisateur, il n'y a pas de différence entre une macro et une fonction de base du système.

L'esprit général de BASILE nécessite une large utilisation des macros, permettant ainsi une bonne structuration des applications en schémas hiérarchiques comme SADT (Fig. A.I.2.). Une tâche peut être scindée en plusieurs macros de même niveau hiérarchique et chacune de ces macros peut elle-même être décomposée.

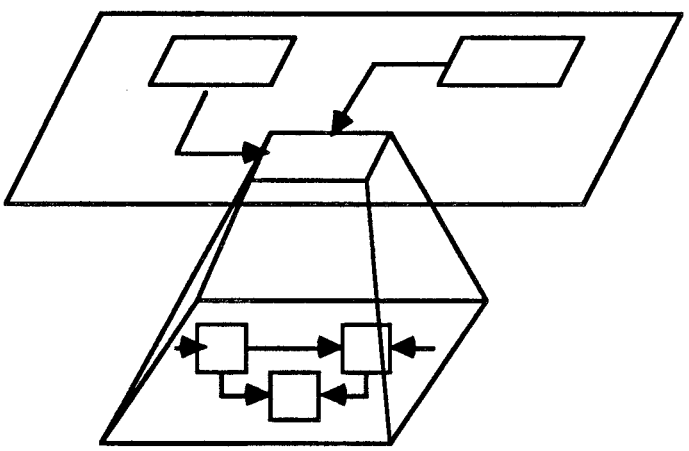


Fig. A.I.2.

4.3.2. Conclusion

Le système BASILE est bien adapté à la solution numérique de problèmes d'automatique dans un environnement d'utilisateurs avertis.

5. CONCLUSION

En général, un bon système de C S C A O doit être à la fois :

- suffisamment complet, pour permettre toutes les manipulations, des plus classiques à celles qui ne sont pas prévues lors de la conception.

- suffisamment simple, pour être utilisable par les différentes catégories d'utilisateurs même les "non experts".

Ces deux objectifs, contradictoires à priori, ne sont pas réalisés par un même système de C S C A O classique, on en déduit la nécessité d'une

nouvelle génération qui doit permettre :

- de guider l'utilisateur dans sa résolution d'un problème de commande en lui indiquant les divers méthodes possibles, en attirant son attention sur les résultats essentiels, ...

- d'offrir une grande variété de méthodes d'analyse et de synthèse, évoluant avec "l'état de l'art".

Pour satisfaire ce cahier des charges, on a utilisé la structure des systèmes à base de connaissances [GENTIL 87 b/88 a et b]. Les avantages potentiels de ce mode de programmation sont :

- la capacité à prendre des décisions de façon plus ou moins autonome puisque l'expertise nécessaire à l'utilisation des systèmes de C S C A O y est incluse. Par conséquent, l'utilisateur peut ainsi ne connaître que les grandes lignes de la théorie de commande et mettre en œuvre, sans embarras technique, des synthèses standards.

- l'ouverture de sa structure, puisque l'ajout d'une nouvelle méthode peut se réduire à l'addition de quelques règles supplémentaires sans remettre en cause l'ensemble du système.

- la possibilité de s'autoformer en interrogeant le système : "Pourquoi avez-vous utilisé la méthode **A** et non pas la méthode **B** ?" (Mais ceci n'est bien sûr possible que dans le cas où le système garde la trace du raisonnement).

II GENERALITES SUR LES SYSTEMES A BASE DE CONNAISSANCES ET LES SYSTEMES EXPERTS

Un système expert se compose des parties suivantes : les deux modules d'interface avec les deux sortes d'opérateurs que sont : l'utilisateur ou l'expert, la base de connaissances et le moteur d'inférence [FARRENY 85].

Chacun de ces modules est détaillé ci-dessous.

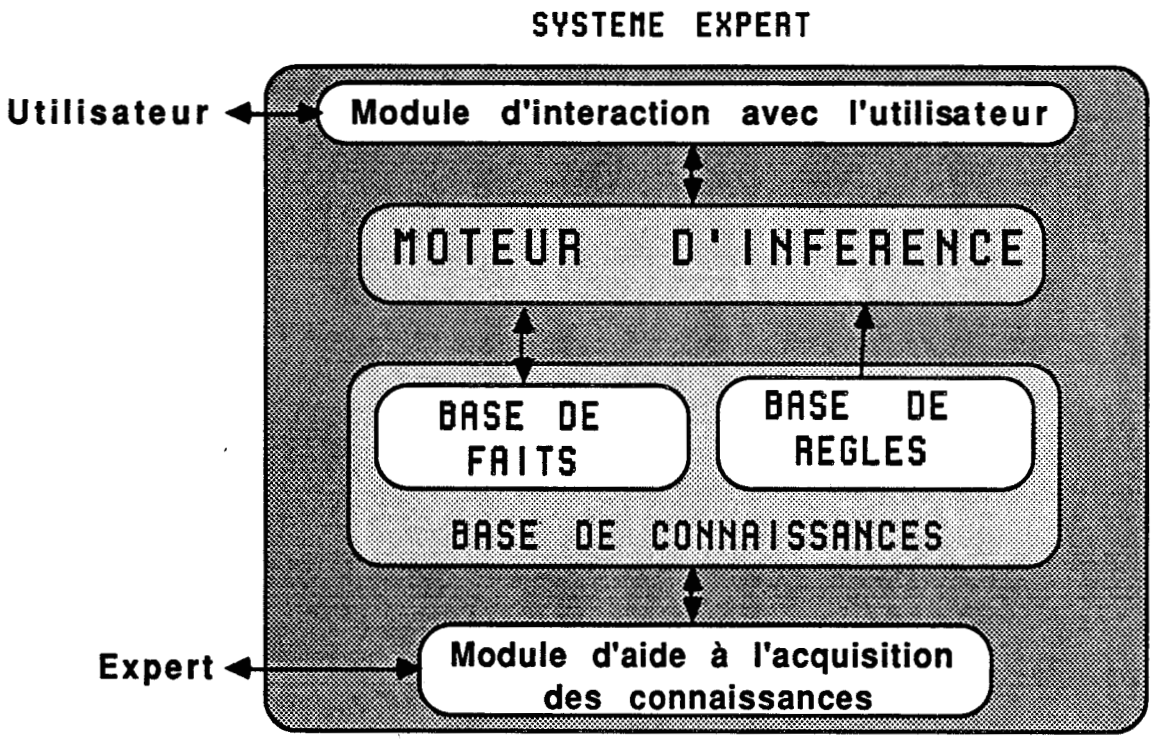


Fig. A.II.1.

1. LES MODULES D'INTERFACE AVEC L'EXTERIEUR

Ces deux modules présentent des interactions homme-machine les plus conviviales possible et les mieux adaptées à chaque type d'opérateur.

- Module d'aide à l'acquisition des connaissances :

  Il offre à **l'expert** le moyen d'introduire aisément ses connaissances. Il vérifie aussi la cohérence des donnés introduites pour éviter les problèmes de contradictions ou de redondances.

- Module d'interaction avec l'utilisateur : qui l'aide à

  - spécifier son problème ;

- répondre aux questions du système ;
- demander au système des explications (comment est-on arrivé à un tel résultat ?, ...).

2. LA BASE DE CONNAISSANCES

Dans ce module l'expert utilise deux sortes de représentations pour décrire :

- les connaissances descriptives, qu'on appelle aussi "faits" (on les trouve dans la base de faits), représentant des situations qui sont **connues** ou à **définir**. Elles **doivent** être déduites par le système ou **peuvent** être demandées à l'utilisateur ;

- les connaissances opératoires ou règles (on les trouve dans la base de règles) décrivant le savoir-faire dans le domaine considéré.

Les règles sont toujours du type :

<déclencheur><corps de la règle>

En général on représente le corps des règles sous forme de clauses de Horn :

Si [hypothèse$_1$, ^ hypothèse$_2$, ^...]
Alors [conclusion$_1$, ^ conclusion$_2$, ^...]

- les hypothèses peuvent définir une condition, une situation, ou un état du processus ;

- les conclusions sont des actions qui sont déclenchées si les hypothèses de la règle sont satisfaites.

Ces actions peuvent être :

- l'ajout d'un fait ;
- la suppression ;
- un appel à l'utilisateur.

Le déclencheur (qui sert à l'étape de filtrage) a une forme qui dépend du mode d'inférence utilisé. Par exemple, en chaînage avant il correspond aux hypothèses du corps de la règle [LAURENT 87].

3. LE MOTEUR D'INFERENCE

C'est un programme qui va tenter de résoudre le problème posé en exploitant la base de connaissances à partir de mécanismes qui combinent les connaissances **descriptives** et les connaissances **opératoires**. Le degré de généralité des connaissances manipulées caractérise l'ordre du moteur d'inférence.

3.1. ORDRE DU MOTEUR D'INFERENCE

Le formalisme autorisé pour l'écriture du corps de la règle caractérise l'ordre du système : 0, 0^+, 1 [RECHENMANN 88].

- moteur d'ordre 0 : les propositions (faits) ne peuvent prendre qu'une valeur binaire : "vrai" ou "faux" ;

- moteur d'ordre 0^+ : possibilité d'évaluer les propositions (faits) par des attributs : booléen, numérique ou symbolique. Le Système Propositionnel Type est utilisé dans ce travail et décrit plus loin (cf. C.I.1.), est un exemple d'un tel moteur ;

- moteur d'ordre 1 : possibilité d'utiliser des variables qu'on peut unifier avec des faits.

3.2. LES MODES D'INFERENCE

Nous pouvons disposer de trois modes d'inférences : chaînage en avant, en arrière et mixte.

3.2.1. Chaînage avant

Ce mode d'inférence opère en partant de la base de faits et en déclenchant des règles dont les prémisses (hypothèses) sont entièrement contenues dans celle-ci. On obtient ainsi une nouvelle base de faits et on poursuit jusqu'à ce qu'on atteigne le but recherché ou bien, jusqu'à saturation complète de la base de faits.

3.2.2. Chaînage arrière

Dans ce mode d'inférence, le système essaie d'atteindre une conclusion spécifique en identifiant la ou les règle(s) qui contient(-tiennent) l'objectif

désiré dans la ou leur conclusion et, en utilisant ses capacités d'inférence à prouver que la ou les prémisse(s) de la ou des règle(s) choisie(s) est (sont) satisfaite(s).

3.2.3. Chaînage mixte

Dans ce mode d'inférence on combine les deux algorithmes de chaînages précédents.

Le principe général est le suivant : on déduit grâce au chaînage avant tout ce qui peut être inféré à partir de la base de faits initiale. Ensuite, le but étant fixé et, grâce au chaînage arrière la "meilleure" question possible est posée à l'utilisateur. Une fois sa réponse formulée, le chaînage avant détermine, tout ce qui peut être déduit grâce à cette information supplémentaire, puis si c'est nécessaire, à nouveau la meilleure question est posée, etc... [DELAHAYE 87].

3.2.4. Exemple

On considère :

- la base de fait :

  A B C D E F G

- la base de règles :

R1 : A --> B	(si A est vrai, alors B est vrai)
R2 : C D --> E	(si C et D sont vrais, alors E est vrai)
R3 : F --> D	(si F est vrai, alors D est vrai)
R4 : B --> D	(si B est vrai, alors D est vrai)
R5 : G --> F	(si G est vrai, alors F est vrai)
R6 : G --> C	(si G est vrai, alors C est vrai)

- le but à atteindre : **E**

Chaînage avant :

- la base initiale de faits : A C Vrais

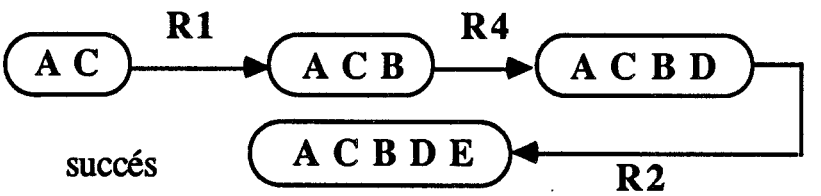


Fig. A.II.2. a

Chaînage arrière :

- la base initiale de faits : A C Vrais et F Faux

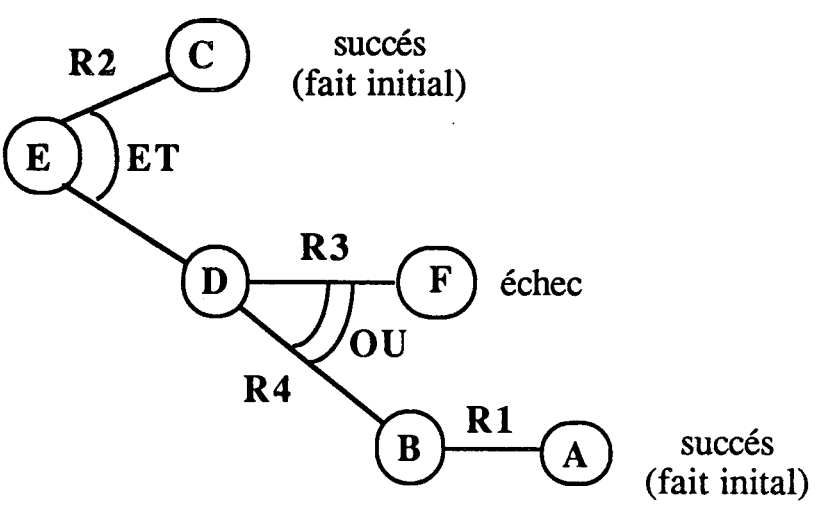
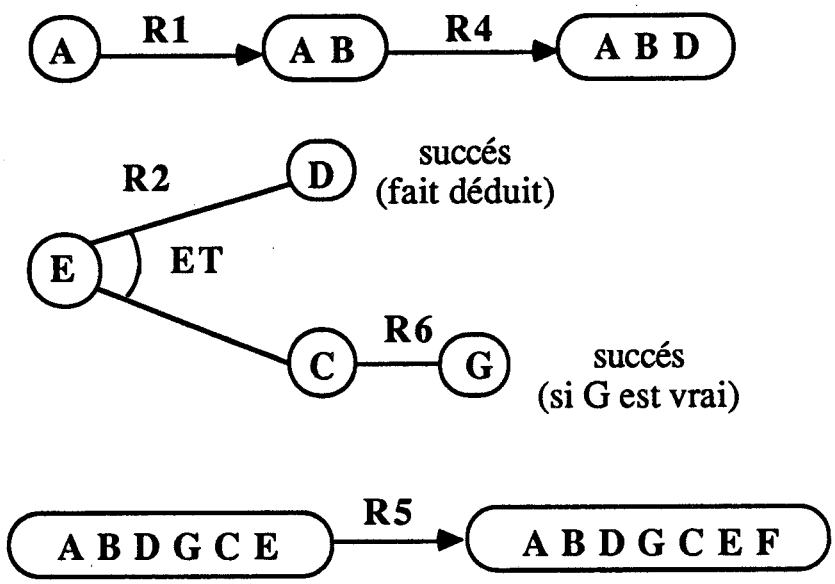


Fig. A.II.2. b

Chaînage mixte :

- la base initiale de faits : A Vrai



Arrêt car le but est atteint

Fig. A.II.2. c

3.3. CYCLE DE BASE D'UN MOTEUR D'INFERENCE

Le moteur d'inférence d'un système expert enchaîne des cycles de travail comportant chacun 4 étapes : restriction, filtrage, résolution de conflits et exécution des règles : [LAURENT 87][FARRENY 87].

3.3.1. Restriction

Elle consiste à déterminer à partir de la base de faits et de règles, deux sous-ensembles **faits sélectionnés** (BF1) et **règles possibles** qui peuvent être comparés lors de l'étape de Filtrage.

3.3.2. Filtrage

Dans cette étape le moteur d'inférence compare la partie déclencheur de chacune des **règles possibles** par rapport à l'ensemble BF1. D'où le

sous-ensemble des **règles déclenchables** jugées compatibles avec BF1.

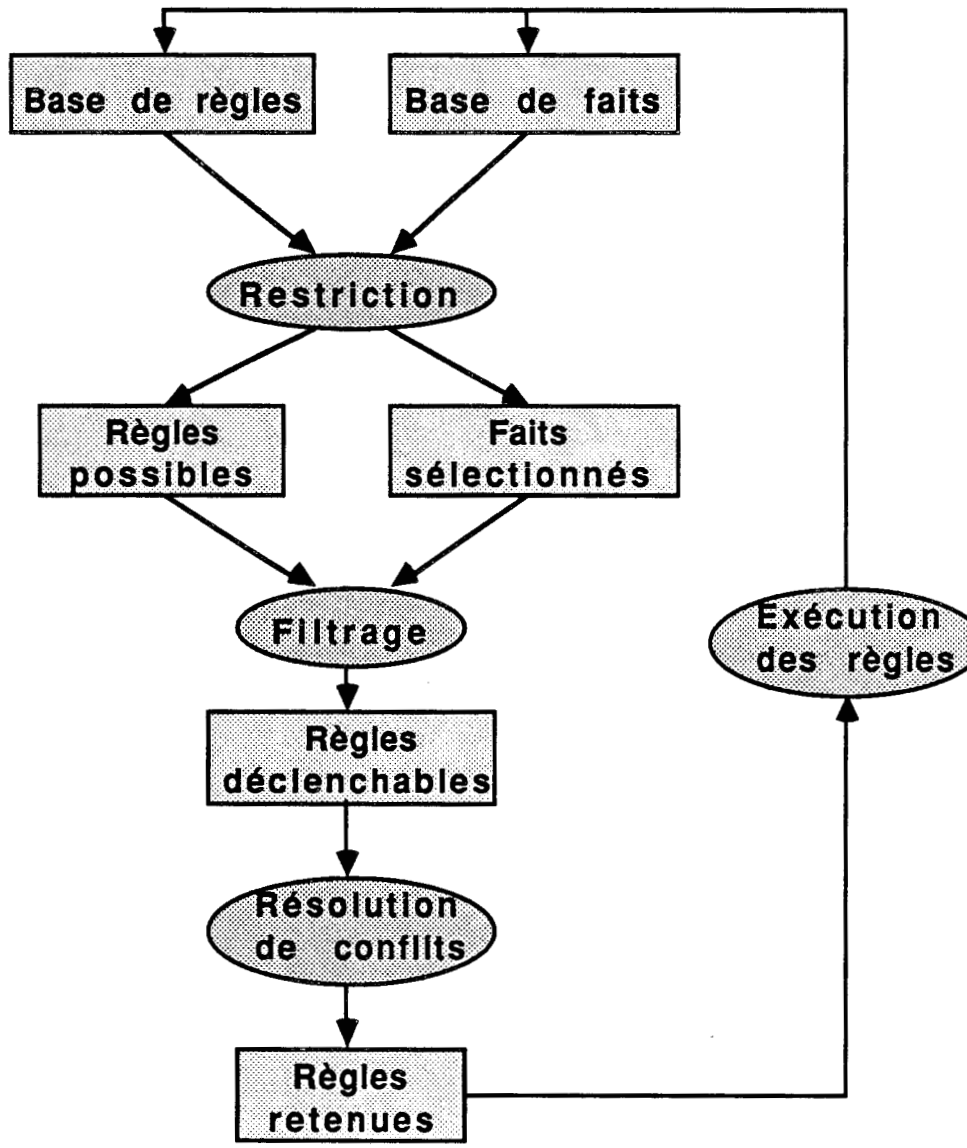


Fig. A.II.3.

3.3.3. Résolution de conflits

Dans cette phase du cycle le moteur doit choisir parmi les **règles déclenchables**, celles qui doivent être effectivement déclenchées. Le sous-ensemble **règles retenues** est le résultat de cette étape.

3.3.4. Exécution

Le moteur d'inférence enchaîne l'exécution des règles retenues selon

une logique bien précise : chaînage en avant, en arrière et mixte.

Remarque importante :

Il faut noter que dans le mode de représentation (faits et règles) décrit avant, les connaissances doivent être relativement indépendantes entre elles. Donc, il n'est pas adapté pour décrire une connaissance fortement structurée. D'où la nécessité d'utiliser d'autres modes de représentation.

4. AUTRES MODES DE REPRESENTATION DE CONNAISSANCES

4.1. REPRESENTATION CENTREE-OBJETS

C'est un mode de représentation bien adapté à la description de domaines fortement structurés, dans lesquels on peut attribuer diverses propriétés aux entités manipulées [RECHENMANN 84 a et b][PIERRET 88].

4.1.1. Notion de spécialisation et d'héritage

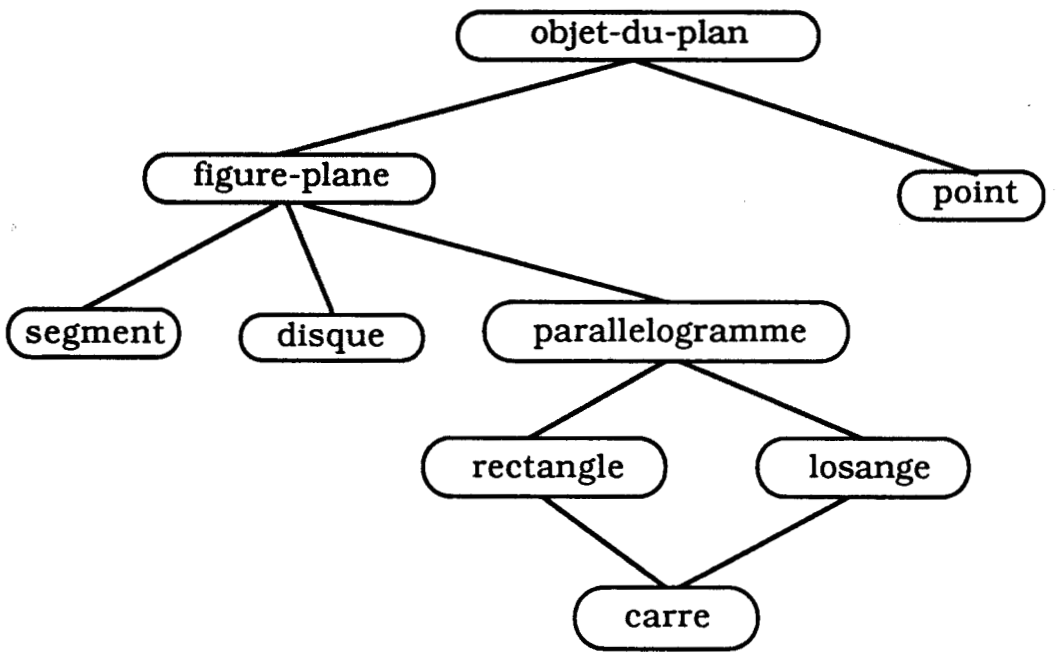


Fig. A.II.4.

On décompose un domaine en classes d'objets, elles peuvent être liées

entre elles par des liens hiérarchiques : une classe est la spécialisation d'une autre, dont elle affine la description. Une classe "hérite" automatiquement des propriétés de la classe supérieure : c'est un mode de raisonnement par défaut.

Dans l'exemple présenté sur la Fig. A.II.4., le **segment** hérite des propriétés de **figure-plane** qui elle même hérite de toutes les propriétés de **objet-du-plan**. Un cas particulier, la classe **carré** hérite des deux classes : **rectangle** et **losange**, c'est ce qu'on appelle l'héritage multiple.

4.1.2. Notion d'instance

Une instance décrit un objet particulier d'une classe, les propriétés de cette dernière y sont évaluées. Par exemple, un rectangle est caractérisé par une largeur et une longueur, ainsi l'instance rectangle_1 sera la description de l'objet rectangle qui a pour largeur la valeur 20 cm et pour longueur la valeur 60 cm.

4.1.3. Notion d'inférence

Les propriétés que l'utilisateur n'a pas évaluées directement peuvent être inférées par le système grâce aux attachements procéduraux ou aux filtres.

4.1.3.1. *Attachement procédural*

Cela consiste à attacher à l'une des propriétés d'une classe, une procédure de calcul grâce à laquelle, le système peut évaluer la valeur de la propriété en question à partir des valeurs d'autres propriétés.

Par exemple, la propriété surface d'un rectangle peut avoir comme procédure de calcul : Surface = Largeur * Longueur. Ainsi le système sera capable de calculer la surface du rectangle_1 qui est 1200 cm^2 = 60 * 20 .

4.1.3.2. *Filtre et filtrage*

Un filtre est un ensemble de conditions que doivent satisfaire des instances d'une classe donnée. Le mécanisme qui consiste à rechercher ces dernières est le filtrage.

C'est ainsi qu'on peut utiliser un filtre pour rechercher toutes les

instances de la classe rectangle qui ont une surface supérieure à 1000 cm^2.

4.2. REPRESENTATION HYBRIDE

La combinaison des objets structurés et des règles de production permet d'obtenir une bonne représentation de connaissances, nettement plus riche que celle qui se limite au triplet "objet-attribut-valeur". Ainsi, on utilise des règles de production pour décrire les connaissances dynamiques et des objets-structurés pour modéliser les connaissances statiques [COMYN 87][BEL 90].

4.3. CONCLUSION

L'utilisation de la représentation centrée-objet a permis à la programmation de style déclaratif (système expert) d'élargir son domaine d'application. Ceci est particulièrement vrai dans le domaine scientifique où la connaissance est souvent fortement structurée.

III UNE PREMIERE TENTATIVE D'UTILISATION DE SYSTEME A BASE DE CONNAISSANCES EN C S C A O

L'ajout de la couche système expert à un logiciel procédural de C S C A O doit donc permettre :

- d'assister la position du problème jusqu'à obtenir une formulation cohérente ;
- de faciliter la résolution d'un problème par tentatives successives ;
- de négocier sur l'adéquation entre les objectifs et les propriétés intrinsèques du processus à contrôler.

Dans ce qui suit, nous présentons une étude faite par [TAYLOR & Al 84/85] où ils donnent un exemple de système à base de connaissances en C S C A O.

1. LES ETAPES D'UNE ETUDE D'AUTOMATISATION

Les auteurs proposent la décomposition suivante des activités d'une étude d'automatisation :

1. Modéliser le processus qui va être contrôlé ; déterminer les caractéristiques du modèle du processus (nombre d'intégrateurs, gain statique, etc...) ;
2. Spécifier les objectifs de la synthèse ;
3. Essayer de voir si le problème de synthèse est bien posé (déterminer le réalisme des objectifs vis-à-vis du modèle du processus étudié et des contraintes de synthèse) ; faire des compromis sur les objectifs si c'est nécessaire ;
4. Choisir et exécuter les procédures de synthèse appropriées ;
5. Valider la synthèse (simulation) ;
6. Etablir une documentation complète de la synthèse retenue ;
7. Réaliser pratiquement la synthèse finale.

Le schéma de la figure suivante représente la séquence des tâches décrites précedement :

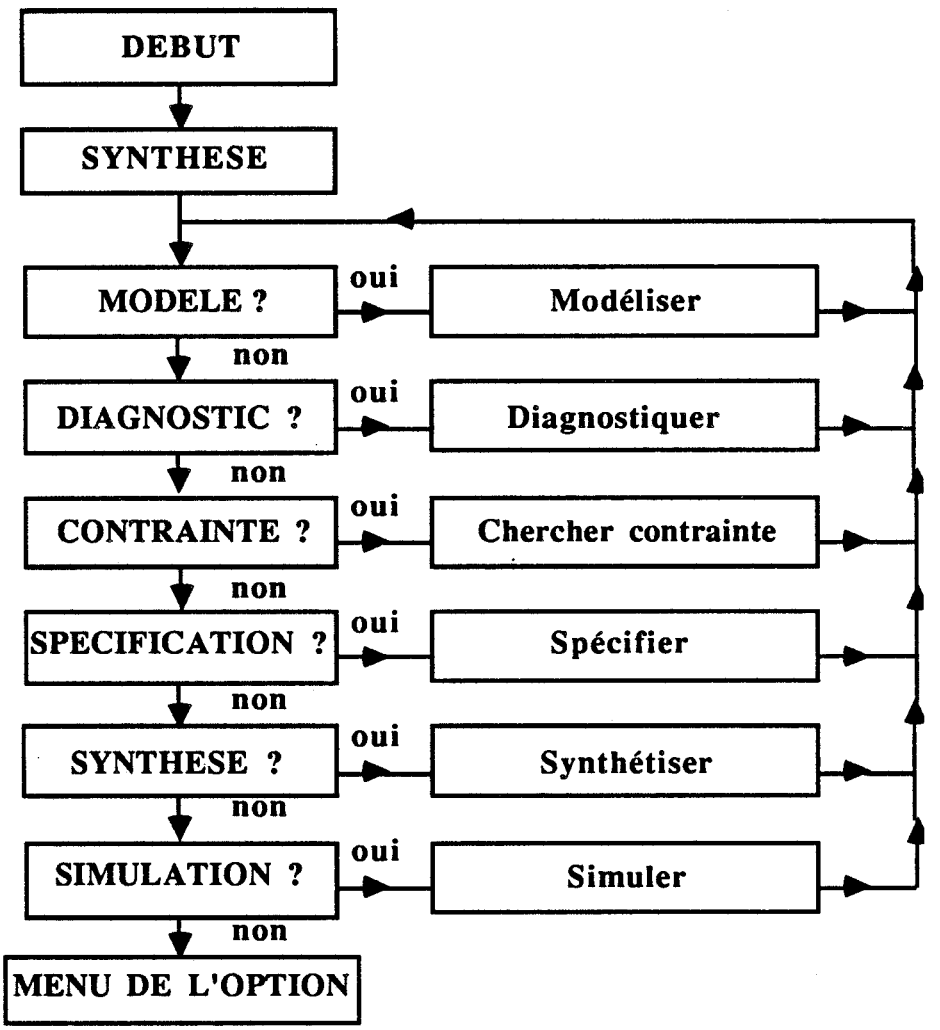


Fig. A.III.1.

2. STRUCTURE DU SYSTEME EXPERT

On retrouve les deux parties principales : La base de connaissances et le moteur d'inférence, celui-ci permet d'utiliser indifféremment le chaînage avant ou arrière. La communication avec l'utilisateur se fait via le moteur d'inférence.

La partie "C S C A O classique" est construite en regroupant les progiciels CLADP et SIMNON [ELMQVIST 77] (SIMNON simule les systèmes non linéaires et calcule les modèles linéaires tangents d'un système non linéaire). Le système expert communique avec ces progiciels en passant par le système d'exploitation via un système de boîte aux lettres.

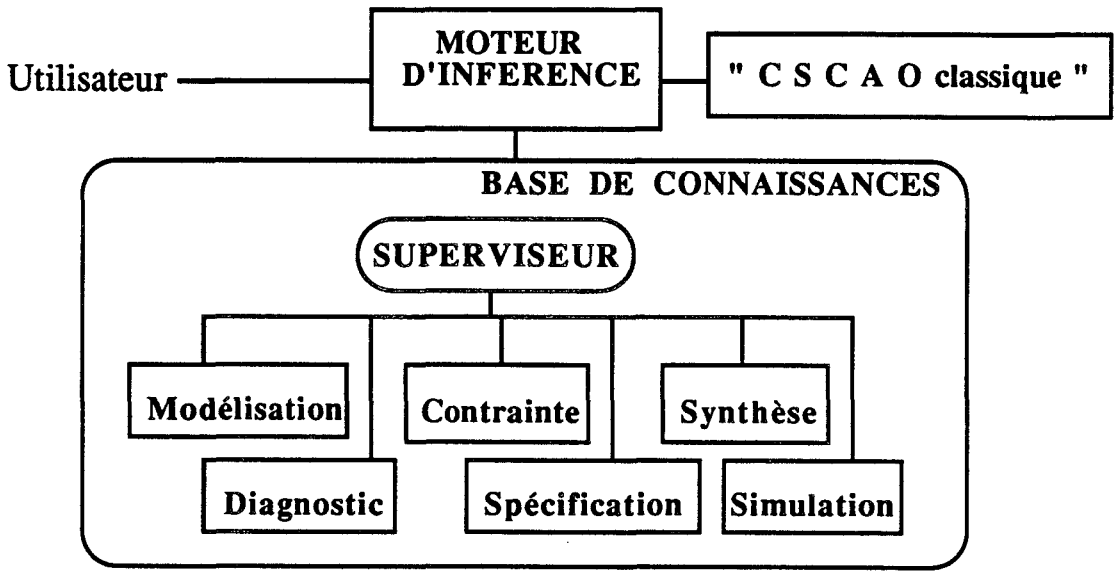


Fig. A.III.2.

2.1. LA STRUCTURE DE LA BASE DE CONNAISSANCES

Elle contient 7 modules (Fig. A.III.1.) conformément à la démarche énoncée dans le premier paragraphe.

Cette modularisation des connaissances rend le système expert plus facile à développer, à modifier et à maintenir.

Les six bases de règles opérationnelles (Fig. A.III.2.) sont gérées par un superviseur qui contrôle l'état courant de la solution du problème et décide des actions ultérieures.

Les fonctionnalités des différents modules du système (Fig. A.III.2.) sont les suivantes :

* SUPERVISEUR :

Les fonctions spécifiques sont :

- sélectionner la base de règles appropriée pour continuer la session quand une série de règles a accompli son travail (au cours de la session de synthèse) ;

- sauvegarder la liste courante de faits si l'utilisateur désire interrompre la session en cours et poursuivre plus tard ;

- permettre à l'utilisateur de réviser la formulation du problème, lui donner ainsi la possibilité d'ajouter ou de supprimer des faits (comme l'intégration de nouveaux résultats d'analyses et de synthèses).

* MODELISATION :

On traite les modèles linéaires en se servant de CLADP ; une extension de SIMNON permet de fournir les modèles linéaires tangents à un modèle non linéaire.

L'utilisateur peut définir/modifier son modèle au clavier et le sauvegarder sur fichier entre deux sessions.

* DIAGNOSTIC :

Ce module permet l'analyse du modèle du processus et l'élaboration des propriétés agrégées ou qualitatives comme par exemple le temps de réponse, le dépassement, etc...

* CONTRAINTE :

Cette base de règles permet à l'utilisateur d'entrer les contraintes telles que la saturation de l'entrée de commande.

* SPECIFICATION :

Le rôle de ce module est de spécifier les objectifs et d'estimer leur faisabilité, tout en considérant simultanément le modèle du processus, les contraintes et les objectifs.

* SYNTHESE :

Il s'agit d'aider l'utilisateur à choisir et à appliquer les procédures de CLADP pour synthétiser la loi de commande.

* SIMULATION :

La fonction de cette base de règles est de faciliter la simulation du système en boucle fermée (CLADP pour les systèmes linéaires, SIMNON pour les systèmes non linéaires).

2.2. LA REPRESENTATION DES CONNAISSANCES

Les connaissances sont représentées sous forme de règles et de faits qui

ont la forme suivante (voir table A.1 et A.2) :

2.2.1. Les règles

Les hypothèses et conclusions des différentes règles (cf. A.II.2.) ont la forme d'un triplet qui peut être vrai, faux, ou neutre (on peut avoir comme valeur +1, -1 ou 0). Ce triplet a la forme suivante :

(objet attribut valeur)

Exemple :

Le fait **(SPEC-CL-POLE MAX-REAL-PART UNASSIGNED) +1** implique qu'aucune valeur n'a pas été spécifiée pour le maximum de la partie réelle des pôles de la boucle fermée du système dont on fait la synthèse.

Une hypothèse peut être que le triplet
est vrai : EQ (objet attribut valeur)
ou faux : NE (objet attribut valeur).

Les conclusions de la règle peuvent être l'une des actions suivantes :

1. Affichage **(DISPLAY)** ;
2. Passage de contrôle à l'utilisateur **(U D O)** ;
3. Ajout d'un fait dans la base de connaissances **(WRITE (obj attr val))** ;
4. Suppression d'un fait de la base de connaissances **(CLR (obj attr val))** ;
5. Demande à l'utilisateur de valider un fait (triplet) **(ASK (obj attr val))** ;
6. Menu, choix d'un triplet à partir de la liste des options **(MENU)**.

Exemple :

En analysant les deux régles de la table A.1, il convient d'observer que la réponse "OUI" à la question dans la conclusion de la règle 106 entraîne que le triplet **(MAX-REAL-PART MODIFY VALUE)** sera écrit dans la base de faits, affecté de la valeur "VRAI" (+1). Cette action satisfait l'hypothèse de la règle 108, qui est alors déclenchée. Cette règle, à son tour, activera celle qui demande à l'utilisateur la valeur de **"SPEC-CL-POLE MAX-REAL-PART"**. Notons qu'on peut utiliser un "joker", qui indique n'importe quelle valeur.

TABLE A.1 Deux règles de ce système expert :

```
RULE 106 (Want to enter max Real part spec ; already assigned)
IF:
 EQ [SPEC-CL-POLE MAX-REAL-PART REQUESTED]
  (User has asked to enter max-real-part)
 EQ [MAX-REAL-PART VALUE]
  (A value is currently assigned)
THEN:
 CLR [SPEC-CL-POLE MAX-REAL-PART REQUESTED]
  (Reset-clear the fact that triggered this rule)
 DISPLAY
  (You want to enter a value for the maximum)
  (real part of the poles, but a value has been)
  (assigned previously)
 ASK [MAX-REAL-PART MODIFY VALUE]
  (Do you wish to replace the current value? (Y or N) ... ENTER :

RULE 108 (Current value of max Real part is to be replaced)
IF:
  EQ [MAX-REAL-PART MODIFY VALUE]
   (User wants to modify the current value)
THEN:
 CLR [MAX-REAL-PART MODIFY VALUE]
  (Reset-clear the fact that triggered this rule)
 CLR [MAX-REAL-PART VALUE]
    (Delete the old value)
 WRITE [SPEC-CL-POLE MAX-REAL-PART UNASSIGNED]
  (Reset-pave the way for a new assignment)
 WRITE [SPEC-CL-POLE MAX-REAL-PART REQUESTED]
  (Trigger the rule to request entry of new value)
```

2.2.2. Liste de faits

L'état de la session (quel que soit l'instant) ou son résultat, quand elle est terminée, est caractérisé par la liste de faits qui a été écrite au cours de l'opération.

Une telle liste est représentée dans la table A.2, qui contient la liste des faits pouvant exister dans la base de connaissances de ce système expert à la fin d'un DIAGNOSTIC et d'une session de SPECIFICATION.

Dans le but d'interpréter la table A.2, les auteurs donnent l'explication des notions de leurs listes de faits : **NL** --> non linéaire, **L** --> linéaire, **DIAG-DOMINANT** --> diagonalement dominant, **DIAGN** --> diagnostic, **FNAME** --> nom de fichier, **SS** --> régime permanent, $\mathbf{CH_i}$ --> canal_i.

TABLE A.2 Liste de faits après une session de diagnostic et de spécification :

PLANT MODEL NONLINEAR	User-asked
PLANT-NL-MODEL FNAME EXOREACT	User-asked
PLANT-NL-MODEL TIME-TYPE CONTINUOUS	Inferred
PLANT-NL-MODEL STATE-TYPE CONTINUOUS	Inferred
PLANT-NL-MODEL ORDER 2	Inferred
PLANT-NL-MODEL INPUTS 2	Inferred
PLANT-NL-MODEL OUTPUTS 2	Inferred
PLANT-NL-MODEL DIAGN-DATA-FNAME EXORNDATA	Inferred
PLANT-NL-MODEL NL-BEHAVIOR MILD	Inferred
PLANT-L-MODEL FNAME EXOREACTL	Inferred
PLANT-L-MODEL STABLE NO	Inferred
PLANT-L-MODEL CONTROLLABLE YES	Inferred
PLANT-L-MODEL OBSERVABLE YES	Inferred
PLANT-L-MODEL MINIMUM-PHASE YES	Inferred
MODEL DIAGNOSIS DONE	Inferred
SENSOR TIME-TYPE CONTINUOUS	User-asked
CONTROLLER TIME-TYPE CONTINUOUS	User-asked
CONTROLLER STRUCTURE DIAG-DOMINANT	User-asked
CONTROLLER CHANNEL1-IN U1	User-asked
CONTROLLER CHANNEL1-OUT Y1	User-asked
CONTROLLER CHANNEL2-IN U2	Inferred
CONTROLLER CHANNEL2-OUT Y2	Inferred
MAX-STEP-SS-ERR CH1-VALUE 0.25	User-asked
MAX-REAL-PART CH1-VALUE -1.4	User-asked
MIN-DAMPING-RATIO CH1-VALUE 1.0	User-asked
MAX-STEP-SS-ERR CH2-VALUE 0.5	User-asked
MAX-REAL-PART CH2-VALUE -1.4	User-asked
MIN-DAMPING-RATIO CH2-VALUE 1.0	User-asked
CONTINUOUS-SPEC ENTRY DONE	Inferred
CONTINUOUS-SPEC ENTRY REALISTIC	Inferred
CONTINUOUS-SPEC ENTRY COMPLETE	Inferred
CONTINUOUS-SPEC ENTRY CONSISTENT	Inferred
SPEC-SESSION TERMINATION NORMAL	Inferred

3. CONCLUSION

L'apport des systèmes à base de connaissances en automatique dépasse la simple mode. Ceux-ci permettront de faciliter l'accès à certaines connaissances pointues pour des utilisateurs, qui ne sont en général pas des experts, et le développement de la commande moderne dans le monde industriel [BARRAUD 89][FAVIER 87].

I V CONCLUSION

Un système expert pour la C A O en automatique devrait être conçu avec les objectifs suivants :

- conforter la rigueur des synthèses d'asservissement et fournir à tout moment une documentation à jour ;

- renforcer plutôt que remplacer, l'habileté et le jugement de l'automaticien.

D'autre part, ce système serait utile dans l'enseignement de l'automatique, en permettant une meilleure compréhension des étudiants qui mettraient plus facilement en pratique ce qu'ils apprennent [GENTIL 87 a][BARRAUD 89].

Enfin, l'Intelligence Artificielle se révèle une technique actuelle, ceci peut être justifié par les deux point suivants :

- un système expert n'est pas figé, il doit pouvoir évoluer, grâce à ses concepteurs qui améliorent les solutions déjà existantes ou en découvrent de nouvelles. Pour faciliter la tâche de l'expert, on lui donne la possibilité de travailler avec un langage de haut niveau. C'est le rôle de l'interface expert.

- le système expert doit être à la disposition de l'utilisateur non spécialiste. Celui-ci (non expert) peut donc communiquer facilement des données au système, et lui poser des questions. Ce dernier doit pouvoir expliquer son raisonnement, ce qui donne plus de clarté qu'une liste de faits déduits. C'est le rôle de l'interface utilisateur .[GENTIL 88 a et b].

En se basant sur ces différentes conclusions, nous nous proposons d'apporter une nouvelle contribution, aussi modeste soit elle, aux efforts déjà entrepris dans ce domaine. Ainsi, et afin de réaliser un système à base de connaissances en C S C A O, nous présentons dans le chapitre suivant les outils automatiques nécessaires à la synthèse de lois de commande.

Chapitre - B -

FORMULATION DES PROBLEMES D'AUTOMATIQUE

Chapitre B

I NOTION DE SIGNAL 63

II CARACTERISATION DES SYSTEMES DYNAMIQUES 67

1. INTRODUCTION 67

2. INTERFACES 67

3. STRUCTURATION DES PROCESSUS 68

3.1. BLOCS ELEMENTAIRES 68

3.2. SYSTEMES COMPLEXES 73

3.2.1. Transformation élémentaire 73

3.2.2. Relation d'addition 74

3.2.3. Exemple 74

4. CALCUL FORMEL DES TRANSFERTS ENTREE-SORTIE 75

4.1. SPECIFICATIONS 75

4.2. ELEMENTS DE CALCUL 75

4.2.1. Substitution récursive 75

4.2.2. Problème de bouclage 77

4.3. CONCLUSION 80

5. CONCLUSION 80

III SYNTHESE DE LOIS DE COMMANDE 81

1. INTRODUCTION 81

2. STRUCTURE DE COMMANDE 82

2.1. COMMANDE PAR RETRO-ACTION 82

2.2. COMMANDE CASCADE 83

2.3. LA COMPENSATION DE PERTURBATIONS MESURABLES EN BOUCLE OUVERTE 84

3. CONTROLE EN TEMPS DISCRET OU TEMPS CONTINU 86

3.1. CHOIX DU PAS D'ECHANTILLONNAGE 87

3.2. MODELES DISCRETS 87

4. PERFORMANCES EN REGIME PERMANENT 88

4.1. PRINCIPE DU MODELE INTERNE 88

4.2. NOMBRE D'INTEGRATEURS D'UN RESEAU CORRECTEUR 90

4.3. COMMANDE PAR MODELE INTERNE 91

4.4. CORRECTION DES SYSTEMES A RETARD PUR 93

5. COMPORTEMENT DYNAMIQUE 94

5.1. INTRODUCTION 94

5.2. LES COMPORTEMENTS DYNAMIQUES NON COMPENSABLES 95

5.2.1. Les retards 95

5.2.2. Les déphasages non-minimaux 96

5.2.3. Saturation et limitation de la dynamique 96

5.3. LES COMPORTEMENTS DYNAMIQUES COMPENSABLES 96

5.3.1. Stabilité des systèmes bouclés 97

6. ROBUSTESSE 102

7. CONCLUSION 103

IV CONCLUSION 104

Nous présentons dans ce chapitre les définitions des différents outils d'automatique que nous utilisons pour créer notre base de connaissances. C'est ainsi que le premier paragraphe est consacré à la notion de signal, le second à la caractérisation des systèmes dynamiques et le dernier à la présentation de la synthèse de lois de commande.

I NOTION DE SIGNAL

Un signal est une fonction caractéristique d'un phénomène variable dans le temps. Il est implicitement défini comme une fonction d'un domaine τ dans $\mathbf{R}$ ou, plus généralement dans $\mathbf{R}^m$ (signaux multidimensionnels).

Le domaine τ dans lequel on définit le temps amène à créer des classes de signaux.

Si τ est dénombrable $\{\tau = \{..., t_1, t_2, ..., t_i, t_{i+1} ...\}\}$ le signal est en **temps discret** sinon $\{\mathbf{R} \supset \tau\}$ le signal est défini en **temps continu.**

De manière pratique une commande par calculateur amène une analyse en temps discret alors que la donnée des processus et la synthèse des contrôleurs de bas niveau s'opèrent en temps continu.

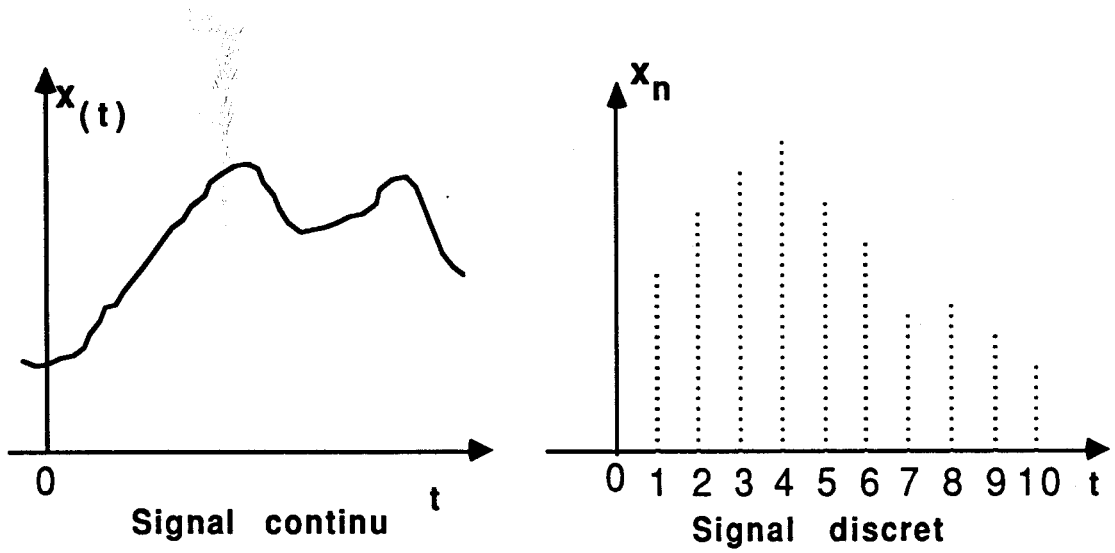


Fig. B.I.1.

Pour un signal en temps discret, nous considérons uniquement le cas où l'échantillonnage est périodique :

$$t_{i+1} - t_i = T \quad \forall i \quad \text{avec } T = \text{période d'échantillonnage.}$$

D'autre part, il convient de noter qu'un signal continu ou discret est aléatoire ou déterministe :

- les signaux aléatoires sont ceux dont l'évolution est en partie, due au hasard ; par exemple la majorité des signaux de perturbations tels les bruits de fond d'un amplificateur opérationnel.

- les signaux déterministes, quant à eux, sont ceux dont on peut connaître parfaitement l'évolution ; ainsi tous les signaux de commande tel le débit de combustible injecté dans une chaudière (Fig. B.II.1.) en sont un exemple.

Enfin, il ne faut pas oublier de citer deux autres caractéristiques très importantes : la mesurabilité du signal (vraie, quand on a accès à la mesure du signal) et son unité de mesure. Cette dernière doit être contrôlée pour valider la somme de plusieurs signaux. Ainsi, un dispositif de contrôle d'unités de mesure doit être présent pour additionner des grandeurs de même type (par exemple : x inches + y mètres devra entraîner la conversion de x inches en x' mètres).

Les formes de signaux les plus couramment utilisés sont : (Fig. B.I.2.)

a) L'échelon : $E(t) = 0$ pour $t < t_0$
$E(t) = a$ pour $t > t_0$

b) La rampe : $R(t) = 0$ pour $t < t_0$
$R(t) = pt$ pour $t > t_0$, p = pente de la droite

L'échelon et la rampe sont souvent utilisés pour étudier les problèmes d'asservissement en présence de consigne constante ou variable.

c) Bruit : c'est un signal aléatoire dans le temps.

d) Impulsion : $I = 1/a$ pour $-a/2 < t < a/2$
$I = 0$ pour $|t| > a/2$

L'impulsion de Dirac correspond au cas limite $a \rightarrow 0$.

Le bruit et l'impulsion sont utilisés surtout pour l'étude de problèmes de garde ou de rejet de perturbation (régulation).

e) Sinusoïde : $S = A\sin(\omega t+\Phi)$
avec A = amplitude, ω = pulsation et Φ = phase.
($\Phi = 0$ sur la figure B.I.2.)

On s'en sert aussi pour l'étude des problèmes de poursuite et de régulation.

f) Constante : $C = a \quad \forall t$

La valeur moyenne d'une grande majorité de signaux de perturbations a le même comportement qu'une constante.

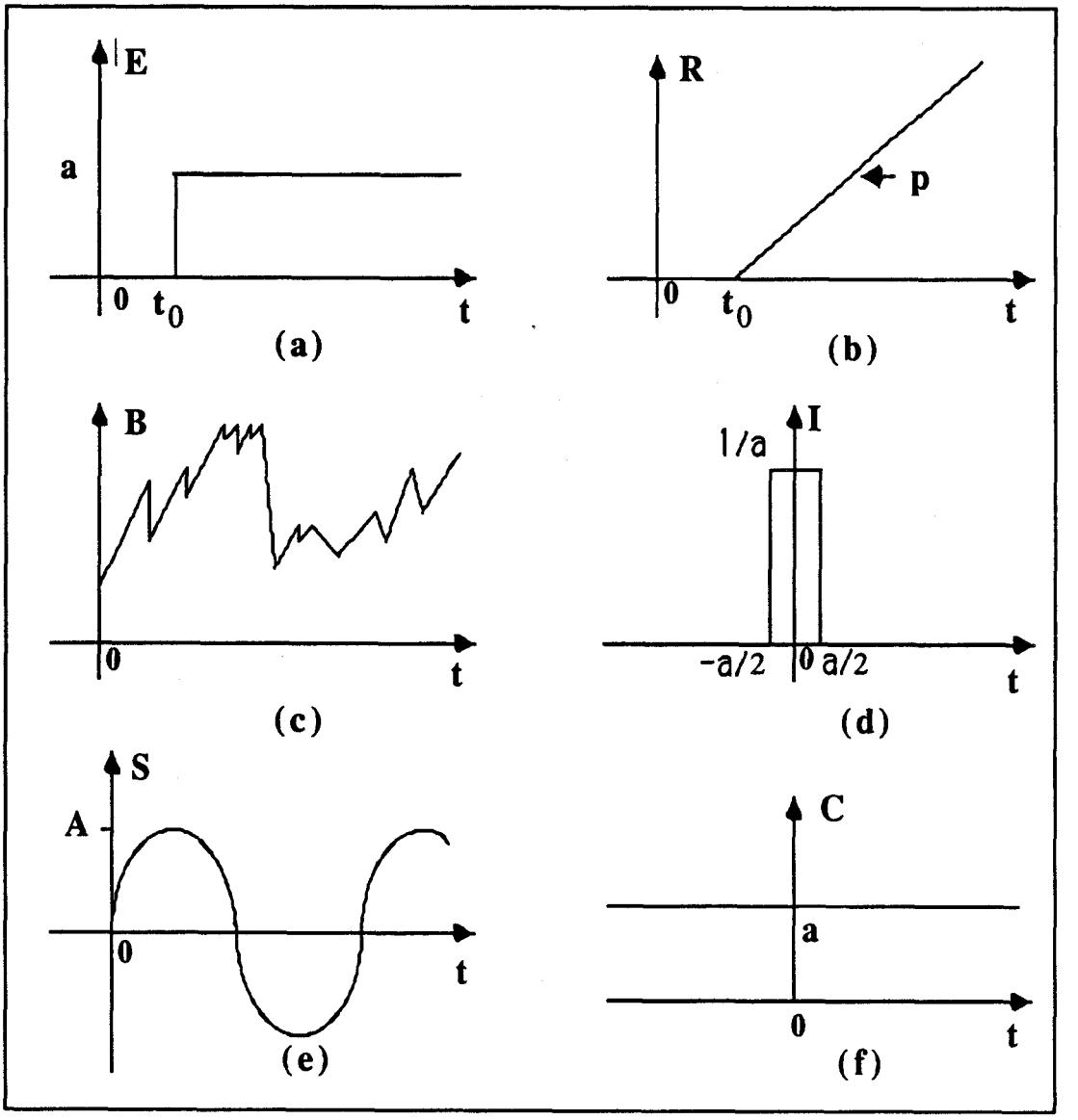


Fig. B.I.2.

En conclusion, on rappelle que le calcul opérationnel offre un formalisme pratique pour l'analyse de processus et de signaux.

Le calcul opérationnel utilise la transformée de Laplace en p pour le

temps continu, en z pour le temps discret.

Le calcul de la transformée d'un signal défini dans le domaine temporel, ou réciproquement celui de la transformée inverse d'une expression opérationnelle s'effectue en utilisant des tables (cf. Annexe 2).

Quelques exemples usuels :

Fonction temporelle	Transformée de Laplace	Transformée en z
échelon : $u(t) = 1$ pour $t > 0$	$u(p) = \frac{1}{p}$	$u(z) = \frac{z}{z - 1}$
rampe : $u(t) = t$ pour $t > 0$	$u(p) = \frac{1}{p^2}$	$u(z) = \frac{Tz}{(z - 1)^2}$
sinusoïde : $u(t) = \sin(\omega_0 t)$ pour $t > 0$	$u(p) = \frac{\omega_0}{p^2 + \omega_0^2}$	$u(z) = \frac{z\sin(\omega_0 T)}{z^2 - 2z\cos(\omega_0 T) + 1}$
T est la période d'échantillonnage ω_0 est la pulsation		

Dans ce cadre, on s'intéresse plus particulièrement aux signaux polynomiaux en la variable temps (par exemple les constantes et les rampes). Ces derniers peuvent être générés à partir de n intégrateurs en série. On définit alors le **type** d'un signal polynomial par le nombre n d'intégrateurs nécessaires pour le générer.

II CARACTERISATION DES SYSTEMES DYNAMIQUES

1. INTRODUCTION

Un système est essentiellement caractérisé par son **interface** avec l'extérieur (ses entrées et ses sorties) et, à un niveau supérieur d'analyse, par sa décomposition en sous-systèmes (constituants).

Il est considéré, à travers la notion de transfert entrée-sortie, comme un transformateur de signaux (un filtrage).

2. INTERFACES

- Les grandeurs de sortie sont caractéristiques de ce qui est produit par le système, elles correspondent aux mesures ou observations qu'il est possible de faire ;

- Les grandeurs d'entrées influencent les sorties, elles peuvent être des commandes choisies librement dans certaines limites par l'opérateur ou par le mécanisme qui a la charge de gouverner le système.
  Sinon ce sont des perturbations que subit le système. On distinguera les perturbations mesurables et non mesurables.

Exemple :

La régulation de température d'une habitation :

Le système est constitué par la chaudière et l'habitation. Le débit du combustible, qui alimente la chaudière, est le signal d'entrée tandis que la température intérieure T est la grandeur de sortie. La température extérieure peut être considérée comme un signal de perturbation.

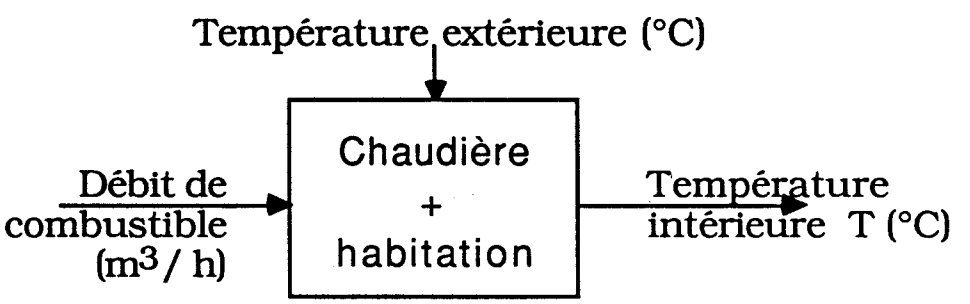


Fig. B.II.1.

3. STRUCTURATION DES PROCESSUS

Un bloc caractérise une transformation dynamique (Fig. B.II.1.). Par définition, il a une sortie (vectorielle de dimension m) et plusieurs entrées (vectorielles). Il est représenté comme un opérateur qui peut décrire soit une connaissance de la physique du procédé, soit un comportement qu'on ne cherche pas forcément à expliquer. Dans ce dernier cas, le bloc est noté "Boîte Noire".

Dans ce travail on se limite à l'étude des blocs avec une seule sortie et une seule entrée de commande. On étudiera aussi bien les blocs élémentaires que les blocs complexes.

3.1. BLOCS ELEMENTAIRES : DESCRIPTION

Ces blocs ont une seule entrée et une seule sortie. Nous les définissons sous la forme y = F * u avec y est le produit à gauche de F par u. La signification de ce produit dépend à la fois de F et de u.

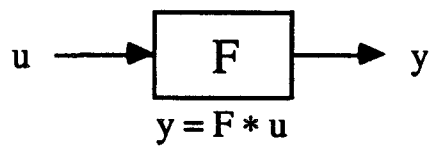
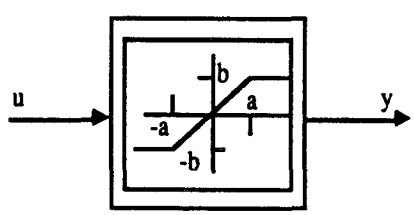


Fig. B.II.2.

On distingue deux catégories de blocs élémentaires : les blocs statiques comme les saturations (Fig. B.II.3.), ou les systèmes dynamiques (Fig. B.II.2.). Pour cette dernière classe nous restreignons notre étude aux processus linéaires stationnaires (fonctions de transfert).



on a $y = u$ pour $|u| \leq a$
$y = b$ pour $u > a$
$y = -b$ pour $u < -a$

Fig. B.II.3.

Un processus élémentaire est caractérisé par ses facettes externes (son entrée et sa sortie) et ses facettes internes. Ces dernières peuvent être plus ou

moins bien précisées, en fonction du degré de connaissance de l'utilisateur.

Idéalement, l'utilisateur connait parfaitement une représentation d'état (A, B et C) ou la fonction de transfert du processus, ou de façon équivalente, ses pôles, ses zéros et son gain statique.

Cette connaissance peut être partielle et se restreindre à des aspects qualitatifs ou à des données numériques macroscopiques, qui sont des informations plus faciles à recueillir.

* Les aspects qualitatifs sont :

Q1) la stabilité asymptotique {stable ou instable}.

Dans la suite, nous assimilons **STABILITE** à **STABILITE ASYMPTOTIQUE**

Q2) le caractère "minimum de phase" (stabilité de l'inverse) {minimum de phase ou non minimum de phase}.

Q3) le type de comportement qu'on peut classer en :

- système d'ordre 0 (le système est comparable à une constante {$Y(p)/U(p) = K$, le seul paramètre est K}).

- système de 1^{er} ordre (le système est assimilable à un système de 1^{er} ordre {$Y(p)/U(p) = K/(1 + \tau p)$, les paramètres sont K et τ}).

- système Ziegler-Nichols (le système est comparable à un modèle de Ziegler-Nichols {$Y(p)/U(p) = K(e^{-pT_r})/p$, les paramètres sont K et T_r}).

- système oscillant (le système est assimilable à un système de $2^{ème}$ ordre {$Y(p)/U(p) = K/(1 + 2(\zeta/\omega_0)p + (p/\omega_0)^2)$, les paramètres sont K, $\zeta<1$ et ω_0}).

* Les aspects numériques macroscopiques (Fig. B.II.4.) :

Ce sont des données faciles à obtenir comme :

- le temps de montée : c'est le temps pendant lequel la réponse du système passe de 0 à 98% de son régime permanent, cette propriété n'a de sens que dans le cas de système stable. On peut la définir sans être obligé de connaître la fonction de transfert (et on peut la

calculer si on dispose de cette dernière). Cet attribut est caractérisé numériquement de façon approximative (± 10%).

- le temps de retard (un réel) qui est une donnée essentielle du problème. Néanmoins, elle est mesurée dans la pratique avec une marge d'incertitude importante.

- le temps de réponse ($T_{montée} + T_{retard}$) peut être directement issu de l'expérience avec une tolérance de l'ordre de celle observée sur $T_{montée}$ et T_{retard}.

- le gain statique (un réel non nul). Lui aussi, ne peut être mesuré avec une grande précision.

- le nombre d'intégrateurs (un entier) bien connu.

Pour des raisons de cohérence de déduction le gain statique de K/p^n est par convention K et non l'infini. De même un processus dont le nombre d'intégrateurs est (-1) est un filtre dérivateur.

En résumé un processus est décrit par le catalogue de caractéristiques suivantes :

* entrée de commande ;
* sortie ;
* temps {continu, discret} ;
* matrices d'état {n, A, B, C} (avec n = l'ordre du système) ;
* numérateur de la fonction de transfert ;
* dénominateur de la fonction de transfert ;
* numérateur de la fonction de transfert continue discrétisée ;
* dénominateur de la fonction de transfert continue discrétisée ;
* type de comportement {1^{er} ordre, $2^{ème}$ ordre (oscill., amort.), Ziegler-Nichols, ...etc} ;
* temps de retard ;
* zéros de la fonction de transfert ;
* pôles de la fonction de transfert ;
* nombre d'intégrateurs de la fonction de transfert ;
* gain statique de la fonction de transfert ;
* temps de montée ;
* temps de réponse ;
* stabilité {stable, instable} ;
* déphasage minimum {minimum, non minimum}.

Certaines de ces caractéristiques peuvent être difficiles à connaître comme l'ordre et les paramètres d'un modèle d'état (n, A, B, C), ou les

cœfficients d'une fonction de transfert (N(p) et D(p)). Les mécanismes de raisonnement doivent permettre la méconnaissance de ces aspects, de même leur pleine utilisation lorsqu'ils sont disponibles.

Par exemple, considérons un processus dont la réponse a un changement de point de consigne est la suivante :

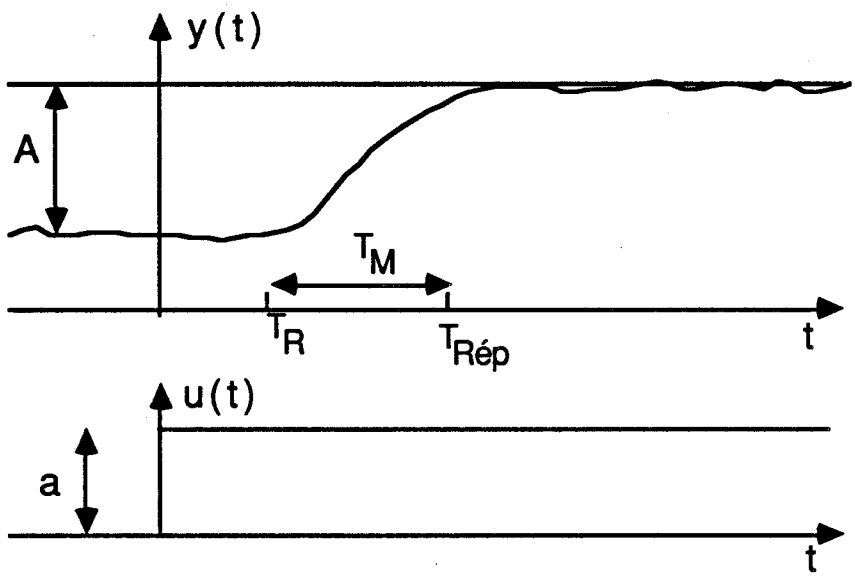


Fig B.II.4.

On peut le caractériser par :

- type de comportement = système de 1er ordre
- un temps de retard = T_R
- un nombre nul d'intégrateurs (option par défaut)
- un gain statique = A/a
- un temps de montée = T_M
- un temps de réponse = $T_{Rép}$
- le fait qu'il soit stable

A l'inverse, considérons un processus défini en temps continu dont on connait parfaitement l'équation d'état (Fig. B.II.5.) :

- temps = continu
- matrices d'état = (2 , -2 0.5 1 -3 , 0 1 , 2 0)

On peut déduire :

- numérateur de la fonction de transfert = 1
- dénominateur de la fonction de transfert = 5.5 + 5p + p^2
- les pôles de la fonction de transfert = (-3.37, -1.63)

- nombre d'intégrateurs de la fonction de transfert = 0
- gain statique de la fonction de transfert = 0.18
- stabilité = stable
- déphasage minimum = minimum
- temps de montée = 2.72

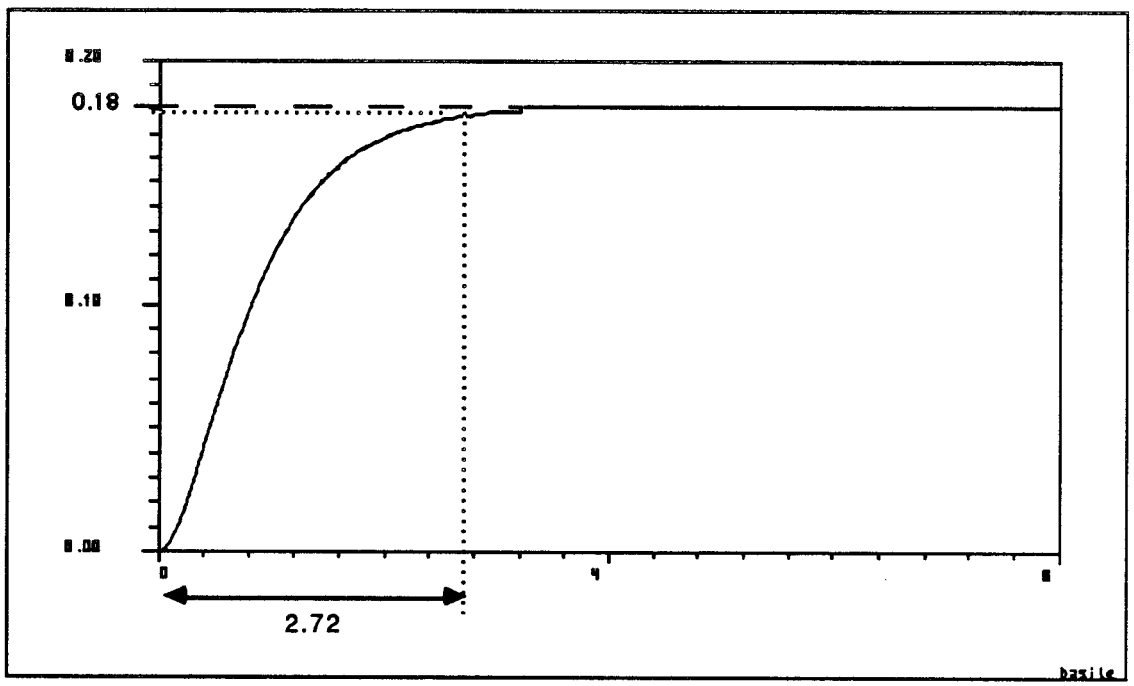


Fig. B.II.5.

Enfin, considérons un processus défini en temps discret dont on connait parfaitement la fonction de transfert F(z) :

- temps = discret
- numérateur de la fonction de transfert = $5 + z$
- dénominateur de la fonction de transfert = $3 + 2z + 5z^2$

On peut déduire :

- les zéros de la fonction de transfert = (-5)
- les pôles de la fonction de transfert = (-0.2 - 0.75i, -0.2 + 0.75i)
- nombre d'intégrateurs de la fonction de transfert = 0
- gain statique de la fonction de transfert = 0.6
- stabilité = stable
- déphasage minimum = non minimum
- temps de montée = 15.66

La réalité physique des processus que nous étudions est bien sûr plus riche que la simple relation fonctionnelle entre une entrée et une sortie que

nous venons de caractériser.

Dans le paragraphe suivant, nous définissons un processus (complexe) à partir de ses composants (élémentaires).

3.2. SYSTEMES COMPLEXES

L'analyse d'un bloc (ou processus) complexe conduit à le définir par plusieurs constituants, et par sa structure formée de liens reliant ces derniers.

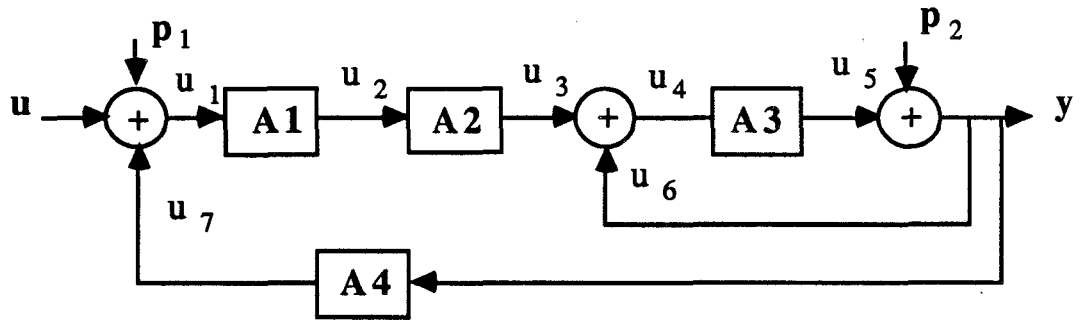


Fig. B.II.6.

On se limitera dans ce travail à un processus complexe qui a une seule sortie contrôlée (**y**), une seule entrée de commande (**u**) et plusieurs entrées de perturbations ($\mathbf{p_1}$, $\mathbf{p_2}$).

Les sorties des blocs internes (**A1 A2 A3 A4**) sont des variables internes (u_1 à u_7) du système, qui peuvent être mesurées ou non.

Deux classes de relations élémentaires peuvent être définies :

- transformation élémentaire ;
- relation d'addition.

3.2.1. Transformation élémentaire (1 sortie, 1 entrée)

- Celle-ci pourra être un filtrage par un processus élémentaire ou complexe. Comme elle pourra être n'importe quelle opération que l'on peut définir par un multiplicateur à gauche, par exemple une saturation (Fig. B.II.3).

Exemple 1 :

Un premier ordre avec $F(p) = k/(1 + \tau p)$

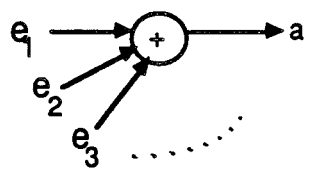
d'où $Y(p) = (k/(1 + \tau p))U(p)$

ici k = gain statique ;
τ = constante de temps.

Exemple 2 :

Un intégrateur avec $F(p) = 1/p$ d'où $Y(p) = U(p)/p$.

3.2.2. Relation d'addition (1 sortie, n entrées n ∈ N)



$$a = e_1\left[+ e_2\left[+ e_3 \ldots \left[+ e_n\right] \ldots\right]\right]$$

Fig. B.II.7.

3.2.3. Exemple

On considère le processus complexe suivant :

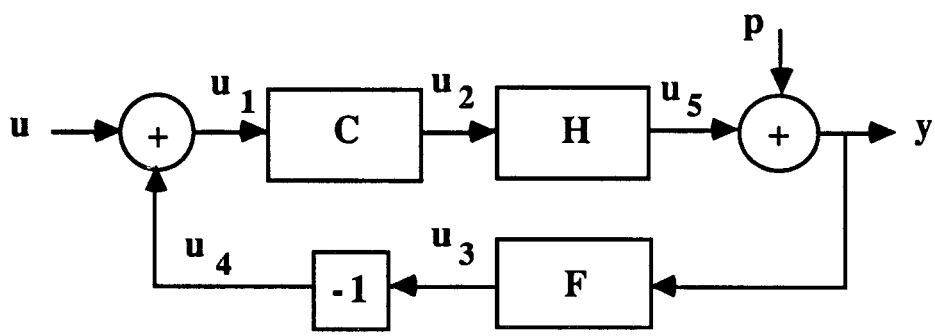


Fig. B.II.8.

Les différentes données du système sont présentées dans des listes, ainsi on a :

- la sortie du système : (y)
- la liste des entrées du système : (u, p)
- la liste des états internes du système : (u_1, u_2, u_3, u_4, u_5)
- la liste des différents blocs élémentaires qui forment le système dynamique : (C, H, F)
- enfin, la liste des différentes relations qui existent entre ces signaux : ($y = u_5 + p$, $u_5 = H * u_2$, $u_2 = C * u_1$, $u_1 = u + u_4$, $u_4 = -1 * u_3$, $u_3 = F * y$)

4. CALCUL FORMEL DES TRANSFERTS ENTREE-SORTIE

Nous avons été amené à développer un module de calcul formel pour déterminer la fonction de transfert du processus complexe en fonction de celles de ses constituants.

4.1. SPECIFICATIONS

Ce calcul est basé sur la substitution récursive des antécédents du signal de sortie étudié. Cette procédure continue jusqu'à ce que la liste finale des antécédents soit réduite à des signaux d'entrée et/ou de sortie.

Une attention particulière doit être portée aux relations comportant une structure bouclée.

Nous exposons dans un premier temps l'algorithme de substitution itérée puis nous examinons la résolution des structures bouclées.

4.2. ELEMENTS DE CALCUL

Pour faire ce calcul formel, nous avons besoin des différentes listes caractérisant le système (cf. B.II.3.2.) dont on veut calculer la fonction de transfert, et les noms des deux signaux (entrées, sortie ou états internes) entre lesquelles on veut déterminer la valeur de la fonction de transfert.

4.2.1. Substitution récursive

Dans une relation de filtrage le signal de sortie a un seul antécédent c'est le signal filtré : $y = F * x$, l'antécédent de y est x.

Dans une relation d'addition le signal de sortie a au moins deux antécédents ce sont les signaux additionnés : $y = x_1 + x_2 + ... + x_n$, x_1, x_2, ..., x_n sont les antécédents de y.

La liste des antécédents (l.a.) du signal de sortie étudié, initialisée avec le(s) premier(s) antécédent(s) du signal de sortie, est une liste où on remplace après chaque substitution, le signal substitué par son ou ses antécédents.

La liste des signaux initiaux (s.i.) est celle où on met au début le signal de sortie, les signaux d'entrées et ceux entre lesquels on cherche à déterminer la fonction de transfert (s'ils ne sont pas des signaux d'entrées ou de sortie).

On appelle relation principale celle obtenue après substitutions itérées des antécédents du signal de sortie. Il convient de noter qu'on prend comme première relation principale, l'égalité qui définit le signal de sortie dans la description initiale.

Exemple :

La description initiale est :

$y = F_3 * u_3$, $u_3 = u_1 + u_2$, $u_2 = F_2 * u$, $u_1 = F_1 * p$, avec s.i. = {y u p}.

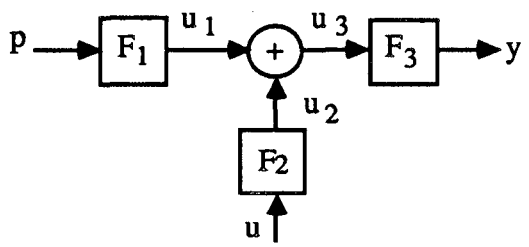


Fig. B.II.9.

L'algorithme fonctionnera alors comme suit :

	Relation principale	l.a.
	$y = F_3 * u_3$	$\{u_3\}$
1ère étape	$y = F_3 * u_1 + F_3 * u_2$	$\{u_1\ u_2\}$
2ème étape	$y = F_3 * F_1 * p + F_3 * u_2$	$\{u_2\ p\}$
3ème étape	$y = F_3 * F_1 * p + F_3 * F_2 * u$	$\{p\ u\}$

Le passage entre deux étapes consécutives consiste à prendre le premier élément (signal) de la liste l.a. et à chercher dans la liste des relations initiales celle où ce signal est à gauche du symbole égal. On remplace ainsi dans la relation principale ce signal par sa valeur.

La procédure de calcul s'arrête lorsqu'il ne reste dans l.a. que des signaux initiaux.

Pour notre exemple, la relation entre y et u est F_3*F_2 et entre y et p est F_3*F_1.

4.2.2. Problème de bouclage

Ce problème est détecté lorsqu'un signal déjà substitué réapparaît après d'autres substitutions.

Donc, il nous faut une liste où stocker tous les signaux que l'on substitue. Cette liste s'appelle s.v. (liste des signaux vus), initialisée à (), à laquelle on ajoute après chaque substitution le signal substitué.

Pour traiter le problème de bouclage, on est amené à tester avant chaque substitution l'appartenance du signal substitué à s.v.. Si le test est négatif, on peut continuer nos substitutions sans contrainte sinon, on caractérise un bouclage.

Pour résoudre un bouclage, on commence par relancer le calcul pour trouver une relation (R_1) entre le signal de sortie et le signal sur lequel on boucle ; puis on calcule la relation (R_2) entre ce dernier et le signal d'entrée ; enfin, on remplace dans la relation (R_1) le signal sur lequel on boucle par la relation (R_2). Ainsi, on a le signal de sortie en fonction du signal d'entrée et parfois de lui-même (dans le cas où on a un retour de la sortie).

Exemple : si y est le signal de sortie, u_1 le signal sur lequel on boucle et u le signal d'entrée.

Pour calculer $y = F(u)$, on cherche $y = G(u_1)$ et $u_1 = H(u)$.
On remplace dans y u_1 par sa valeur :

$$y = G(H(u)) \quad \text{d'où} \quad y = F(u).$$

Pour avoir la relation finale on doit résoudre encore un éventuel bouclage sur la sortie en effectuant l'opération suivante :

Si $y = F(u,y)$
$y = F_1 * u + F_2 * y$
$(1 - F_2) * y = F_1 * u$
d'où $y = (1 - F_2)^{-1} * F_1 * u$

Exemple 1

Description initiale :

(D1) $y = H * u_6$ (D5) $u_6 = u_5 + u_7$

(D2) $u_7 = L * u_2$ (D6) $u_5 = K * u_1$

(D3) $u_2 = G * u_1$ (D7) $u_1 = u + u_4$

(D4) $u_4 = -1 * u_3$ (D8) $u_3 = F * u_6$

s.i. = {y u}

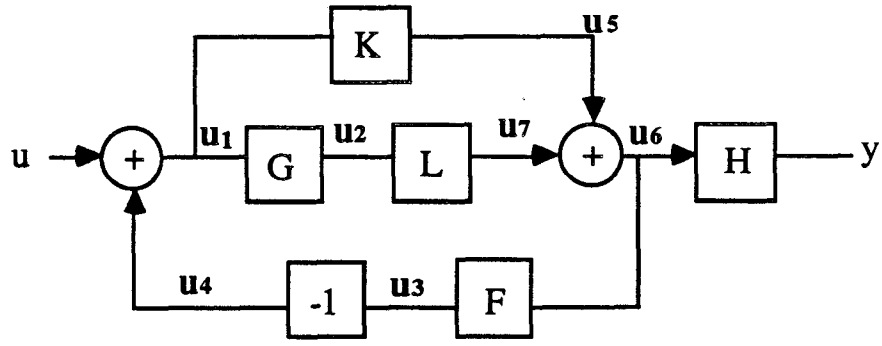


Fig. B.II.10.

Relation principale	règle utilisée	l.a.	s.v.
$y = H * u_6$	(D1)	$\{u_6\}$	$\{\}$
$y = H * u_5 + H * u_7$	(D5)	$\{u_5\ u_7\}$	$\{u_6\}$
$y = H*K * u_1 + H * u_7$	(D6)	$\{u_7\ u_1\}$	$\{u_6\ u_5\}$
$y = H*K * u_1 + H*L * u_2$	(D2)	$\{u_1\ u_2\}$	$\{u_6\ u_5\ u_7\}$
$y = H*K * u + H*K * u_4 + H*L * u_2$	(D7)	$\{u_2\ u_4\ u\}$	$\{u_6\ u_5\ u_7\ u_1\}$
$y = H*K * u + H*K * u_4 + H*L*G * u_1$	(D3)	$\{u_4\ u\ u_1\}$	$\{u_6\ u_5\ u_7\ u_1\ u_2\}$
$y = H*K * u + H*K*(-1) * u_3 + H*L*G * u_1$	(D4)	$\{u\ u_1\ u_3\}$	$\{u_6\ u_5\ u_7\ u_1\ u_2\ u_4\}$

u est un signal d'entrée, il n'a pas d'antécédent, donc rien à modifier.

u_1 appartient à s.v.. Il apparaît ainsi un phénomène de bouclage. Le calcul est relancé pour trouver la relation entre y et u_1.

Après calcul on aura : $y = H*K * u_1 + H*L*G * u_1$

donc $y = (H*K + H*L*G)* u_1$

Reste à calculer u_1 en fonction de u :

Relation principale	règle utilisée	l.a.	s.v.
$u_1 = u + u_4$	(D7)	{u u_4}	{ }
$u_1 = u + (-1)* u_3$	(D4)	{u_3}	{u_4}
$u_1 = u + (-1)*F * u_6$	(D8)	{u_6}	{u_4 u_3}
$u_1 = u + (-1)*F * u_7 + (-1)*F * u_5$	(D5)	{u_7 u_5}	{u_4 u_3 u_6}
$u_1 = u + (-1)*F*L * u_2 + (-1)*F * u_5$	(D2)	{u_5 u_2}	{u_4 u_3 u_6 u_7}
$u_1 = u + (-1)*F*L * u_2 + (-1)*F*K * u_1$	(D6)	{u_2 u_1}	{u_4 u_3 u_6 u_7 u_5}
$u_1 = u + (-1)*F*L*G * u_1 + (-1)*F*K * u_1$	(D3)	{u_1}	{u_4 u_3 u_6 u_7 u_5 u_2}

donc $u_1 = (1 + F*L*G + F*K)^{-1}* u$

d'où $\mathbf{y = (H*K + H*L*G)*(1 + F*L*G + F*K)^{-1}* u}$

Exemple 2

Exemple de session de travail :

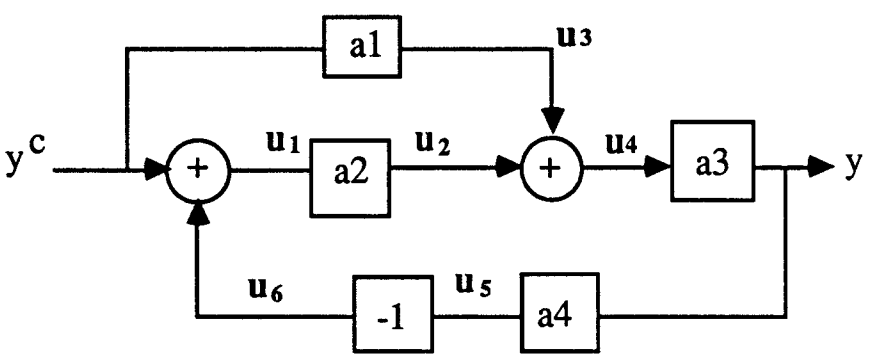


Fig. B.II.11.

```
Entrez la liste des signaux d'entrees : (yc)
Entrez la liste des signaux de sortie : (y)

Entrez la liste des relations initiales :
((y = a3 * u4)(u4 = u2 + u3)(u3 = a1 * yc)(u2 = a2 * u1)
 (u1 = yc + u6)(u6 = -1 * u5)(u5 = a4 * y))

Entre quels signaux voulez-vous la fonction de transfert ?
Signal d'entree choisi : yc
Signal de sortie choisi : y
y = (+ a3* u4)
y = (+ a3* u3 a3* u2)
y = (+ a3* u3 a3*a2* u1)
y = (+ a3*a1* yc a3*a2* u1)
y = (+ a3*a1* yc a3*a2* yc a3*a2* u6)
y = (+ a3*a2* u6 coef0* yc)     coef0 = a3*a1+a3*a2
y = (+ a3*a2*(-1)*a4* y coef0* yc)
y = (+ coef1*coef0* yc)         coef1 = 1/(1-1*a3*a2*(-1)*a4)
```

4.3. CONCLUSION

Ce module est capable de nous calculer la fonction de transfert entre n'importe quels points d'un processus complexe.

Il est écrit en Le_lisp de l'INRIA et tourne sur Vax/Vms et sur IBM & Compatibles.

5. CONCLUSION

En résumé, un système dynamique est caractérisé par :

- l'entrée de commande
- les entrées de perturbations :
  - mesurables
  - non mesurables
- les états internes :
  - mesurables
  - non mesurables
- la sortie
- les constituants
- les liens entre constituants

Cette définition est récursive (un constituant est un système dynamique).

III SYNTHESE DE LOIS DE COMMANDE

1. INTRODUCTION

En introduction de ce paragraphe, il convient de rappeler le but premier de l'automatique. Le rôle de cette discipline consiste à déterminer la **décision** qu'il faut appliquer au système étudié pour obtenir les **performances imposées** et ce, compte-tenu des **informations** que l'on possède sur le système.

Dans le cas présent, le terme "décision" désigne ici le signal (ou signaux) de commande, tandis que les informations sur le système sont de deux ordres : son modèle et ses sorties.

- le modèle du système nous permet de prévoir le comportement de sa sortie en fonction des entrées qui peuvent lui être appliquées ;

- les mesures effectuées sur le système (ses sorties) nous fournissent l'état dans lequel il se trouve au moment de la prise de décision.

Il reste à préciser les performances que l'utilisateur impose. Nous allons les spécifier en terme d'objectifs, elles sont de deux types :

- Les spécifications "statiques" :

  Elles consistent à chercher la structure de lois de commande à appliquer au système pour avoir une erreur consigne/sortie nulle ou qui converge vers zéro. Ceci doit être valable pour toute la classe des signaux de consigne qu'on souhaite poursuivre, ainsi que pour toute la classe des signaux de perturbations que l'on doit rejeter .

- Les spécifications "dynamiques" :

  Elles sont essentiellement liées à la dynamique naturelle du système. Ainsi, la stabilité constitue une exigence minimale imposée par défaut. Le caractère non-oscillant représente une propriété généralement souhaitée. Enfin, le processus corrigé doit contenir les dynamiques non-compensables du système initial comme les retards.

D'autre part, on ne peut pas parler d'objectifs ou de performances sans citer le terme contrainte.

La prise en compte de contraintes de réalisations, liées essentiellement aux caractéristiques du processus qu'on se propose de commander amène

naturellement à contraindre les objectifs visés pour les rendre réalisables.

Comme exemple de contrainte, on peut citer celles qui correspondent à une saturation de l'entrée, c'est à dire à une limitation physique du processus. Cette contrainte est présente par défaut. L'utilisateur devra fixer les bornes min et max de la commande ou déclarer que cette contrainte n'existe pas.

En conséquence, d'une limitation de l'entrée de commande, on sait qu'on ne pourra pas accélérer arbitrairement la dynamique de la sortie. Le temps de réponse du système asservi à concevoir et celui du processus contrôlé doivent donc être du même ordre.

2. STRUCTURE DE COMMANDE

Un problème de contrôle commence toujours par spécifier un comportement souhaitable pour une ou plusieurs sorties. Ce comportement doit être explicité comme par exemple $y^c(t)$ (Fig. B.III.1.). Pour le réaliser nous associons le processus à un contrôleur. La poursuite asymptotique de ce comportement amène des spécifications de structure vis à vis du contrôleur.

2.1. COMMANDE PAR RETRO-ACTION

L'asservissement d'un processus consiste à appliquer sur ses entrées, une loi de commande de manière à satisfaire les critères imposés par l'utilisateur malgré l'action des entrées de perturbations.

La structure de commande par rétro-action est directement issue de cette dernière formulation puisque les lois de commande, pour contrebalancer l'effet des entrées de perturbations, doivent disposer des informations relatives à celles-ci. Lorsque les perturbations ne sont pas mesurées (mesurables) ces informations ne sont disponibles que dans les sorties mesurables du processus, elles sont donc injectées dans le bloc définissant la loi de commande du procédé à partir des critères imposés (Fig. B.III.1.).

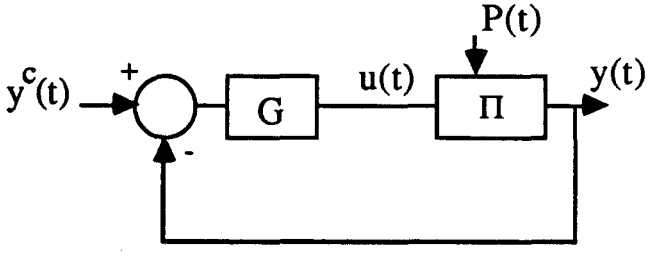


Fig. B.III.1.

Le cas des systèmes que nous envisageons, celui des systèmes monovariables, n'induit pas une très grande complexité structurelle des applications traitées.

On peut quand même évoquer un cas ou le contrôle d'une seule variable peut et doit être abordé par la solution de problèmes emboîtés : c'est le cas de commande **CASCADE**.

2.2. COMMANDE CASCADE

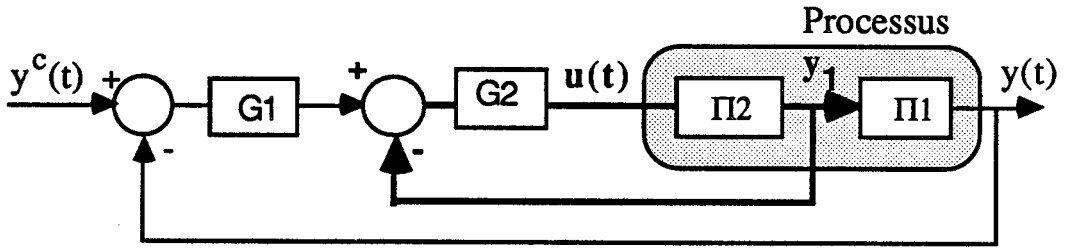


Schéma générique de la commande cascade
Fig. B.III.2.

L'étude d'un tel système s'opère :

- en détectant l'existence d'une mesure intermédiaire y_1 permettant de réaliser une boucle interne avec le régulateur G2. Cette dernière sera emboîtée entre la variable contrôlée y(t) et son comportement désiré $y^c(t)$.

- on fait alors la synthèse du contrôleur de la boucle interne en ne se préoccupant que des spécifications statiques et dynamiques de celle-ci (sans tenir compte de la compensation des perturbations externes à cette boucle interne).

Cette procédure se généralise d'abord dans le cas où il existe plusieurs cascades possibles. On aboutit alors à un schéma de commande par boucles emboîtées.

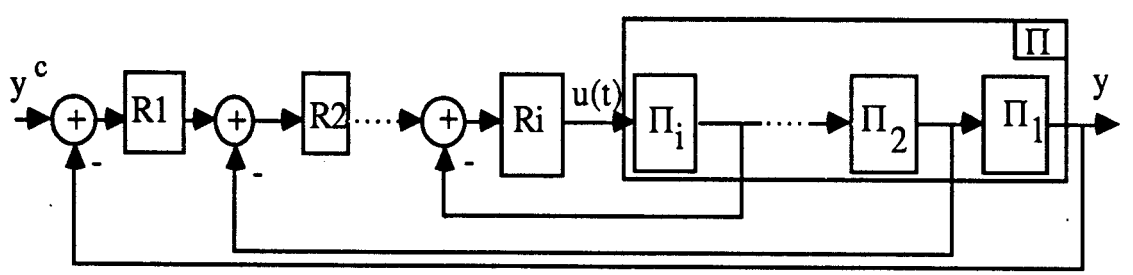
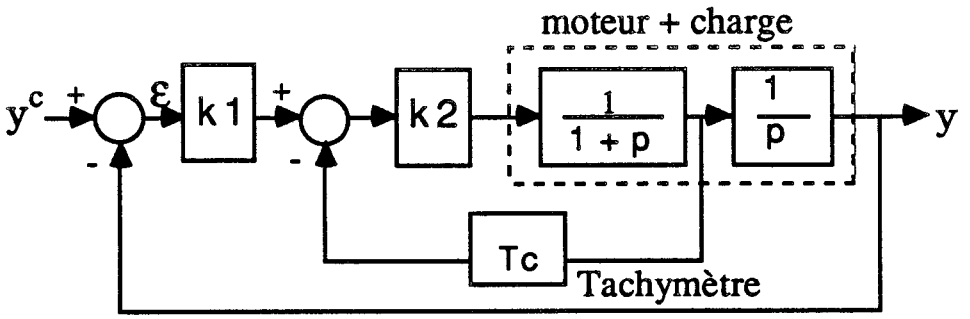


Fig. B.III.3.

Enfin, il faut noter que ce type de réglage voit sa généralisation dans la commande par **RETOUR D'ETAT** [FOULARD 87].

Exemple 1 :



Correction tachymétrique d'un asservissement de position
Fig. B.III.4. a

Dans cette exemple, on asservit la vitesse du moteur grâce à la boucle interne. Tandis que la boucle externe sert d'asservissement de la position.

Exemple 2 :

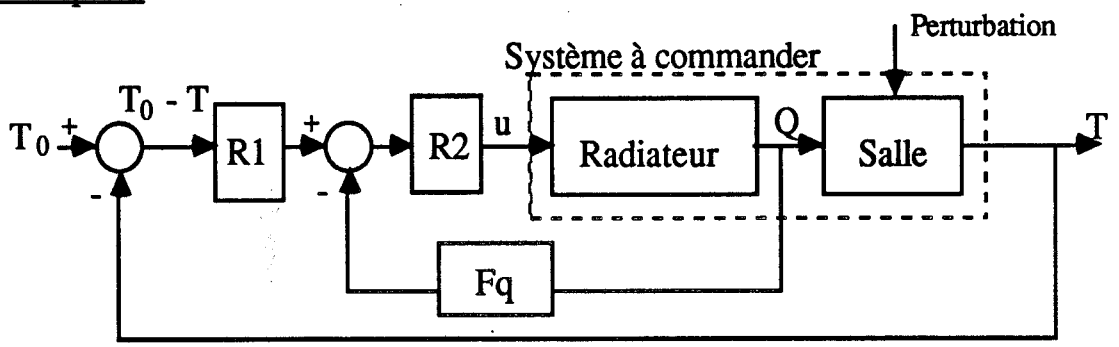


Fig. B.III.4. b

Q débit en calories du radiateur
u commande du radiateur

Comme dans l'exemple 1, la boucle interne a pour rôle de contrôler le débit calorifique du radiateur alors que la boucle externe asservit la température de la salle.

2.3. LA COMPENSATION DE PERTURBATIONS MESURABLES EN BOUCLE OUVERTE

Cette compensation peut et doit être faite a posteriori de la synthèse

d'une loi de commande en boucle fermée (Fig. B.III.5.), permettant ainsi une action anticipée qui doit réduire l'écart entre le comportement nominal du système et son comportement perturbé.

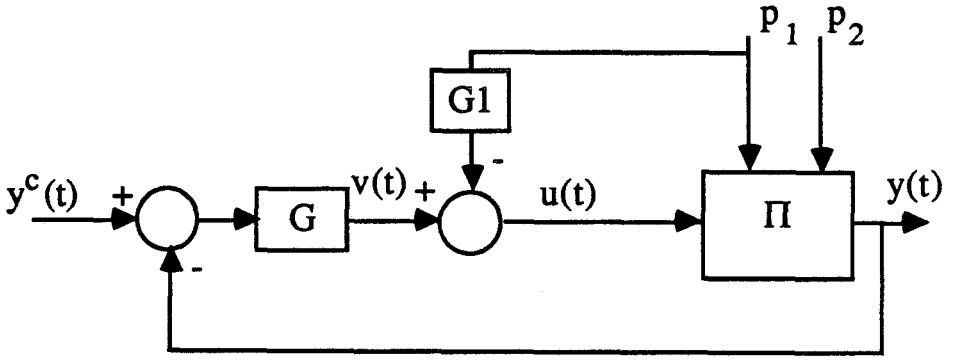


Fig. B.III.5.

- p_1 et p_2 sont des perturbations respectivement mesurable et non mesurable ;
- u est la commande, y la sortie à commander, y^c la consigne ;
- G et G1 sont des régulateurs.

Le calcul du correcteur G va être détaillé dans le paragraphe III.4. du présent chapitre. Nous présentons ici la procédure de détermination de G1 qui est effectuée indépendamment de celle de G.

D'après la figure B.III.5. on peut prendre :

$$y = H*u + H1*p_1 + p_2 \quad (1)$$

et $u = v - G1*p_1$ (2)

avec H et H1 caractéristiques de Π

En remplaçant dans (1) u par sa valeur dans (2), on a :

$$y = H*v - HG1*p_1 + H1*p_1 + p_2 \quad (3)$$

On souhaite éliminer l'influence de p_1 : $y = H*v + p_2$ (4)

ceci est vérifié si $- HG1*p_1 + H1*p_1 = 0$

donc il faut prendre $G1 = H^{-1}H1$.

On pose $H^{-1} = H^-H^+$, H^- étant à la fois stable et réalisable et H^+ étant de gain statique unité. Ainsi, pour que G1 soit stable et réalisable il faut prendre **$G1 = H^-H1$.**

Exemples :

- si $H = \frac{k}{1 + \tau p} \Rightarrow H^{-1} = \frac{1 + \tau p}{k}$ d'où $H^- = \frac{1}{k}, H^+ = 1 + \tau p$

- si $H = k\frac{1 + p}{1 + 2p} \Rightarrow H^{-1} = \frac{1 + 2p}{k(1 + p)}$ d'où $H^- = \frac{1 + 2p}{k(1 + p)}, H^+ = 1$

- si $H = k\frac{1 - p}{1 + 2p} \Rightarrow H^{-1} = \frac{1 + 2p}{k(1 - p)}$ d'où $H^- = \frac{1}{k}, H^+ = \frac{1 + 2p}{1 - p}$

3. CONTROLE EN TEMPS DISCRET OU EN TEMPS CONTINU

L'utilisation d'un asservissement en temps discret ou temps continu peut être un choix de l'utilisateur ; comme elle peut être une contrainte d'ordre technologique. Si on utilise des éléments analogiques (pneumatiques, électroniques, ...) on réalise la commande en temps continu. Par contre, la mise en œuvre de composants informatiques amène à se poser le problème de la discrétisation des signaux donc à travailler en temps discret.

Dans ce paragraphe, nous allons développé le cas de la synthèse de loi de commande en temps discret. Cela suppose :

- le choix d'un pas d'échantillonnage T compatible avec les possibilités du système à contrôler et avec les objectifs fixés pour le système asservi ;

- la connaissance d'un modèle discret du système à commander, c'est-à-dire la connaissance de la relation entre les entrées $\{u_k\}$, $\{p_k\}$ et la sortie $\{y_k\}$ (Fig. B.III.6.). Celle-ci est immédiate si le processus muni de ses convertisseurs a été directement identifié en temps discret. Dans le cas où l'on ne dispose que de la relation continu entre u(t), p(t) et y(t), on devra effectuer une transformation que nous expliciterons.

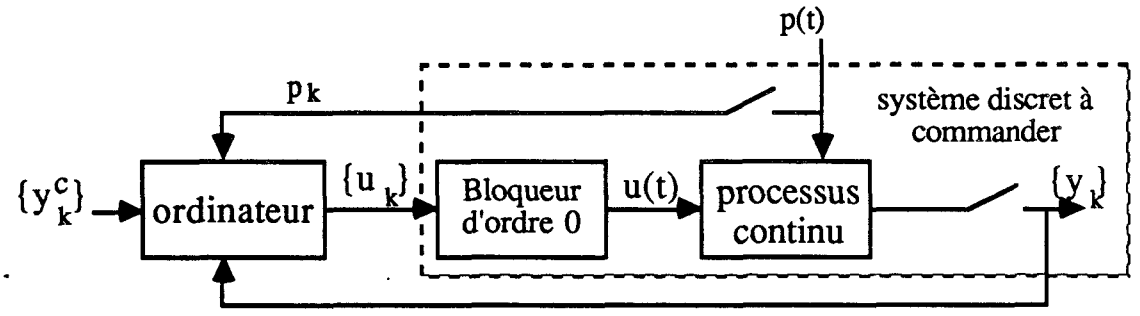


Fig. B.III.6.

3.1. CHOIX DU PAS D'ECHANTILLONNAGE

Ce choix doit être compatible avec les performances naturelles du système et celles de l'asservissement.

Cette sélection s'opère à partir de conditions issues du théorème de Shannon, sur la base des temps de réponses considérés pour l'ensemble des processus composant le système étudié.

Une période d'échantillonnage raisonnable, entre le 1/4 et le 1/10 du temps de réponse du système [ASTRÖM 84 b] peut être ainsi suggérée au concepteur qui l'acceptera ou en proposera une autre. Le calcul du temps de réponse d'un système simple ou composé est détaillé dans le paragraphe C.III.1.2.1.

Exemple 1 :

Considérons le système monovariable de fonction de transfert G(p) :

$$G(p) = \frac{2}{(0.5 + p)(0.4 - 0.2i + p)(0.4 + 0.2i + p)}$$

On calcule conventionnellement son temps de réponse (§ C.III.1.2.1), $T_{rép} = 12.81$ s.

On peut donc proposer raisonnablement $T = 0.1T_{rép} = 1.28$ s.

Exemple 2 :

Dans ce second exemple, on ne connaît que les propriétés numériques macroscopiques du processus à commander : Gain statique = 3, Temps de réponse = 4 s ==> T = 0.4 s est donc un choix possible.

3.2. MODELES DISCRETS

Nous disposons de deux méthodes permettant d'obtenir un modèle discret à partir d'un modèle continu : une solution à base de transformées et une utilisant les équations d'état [SEVELY 73].

Pour les besoins de l'étude nous ne détaillerons que la 2ème méthode. Supposons que dans le schéma de la figure B.III.6., on ait :

,

$$\dot{x} = Ax + B_1 u + B_2 p$$
$$y = Cx$$

On sait alors [Delarminat 75, pp. 304-305], que :

$$x_{k+1} = A' x_k + B'_1 u_k + B'_2 p_k$$
$$y_k = C x_k$$

avec

$$A' = e^{AT}$$

$$B'_1 = \left[\int_0^T e^{At} dt\right] B_1 \qquad B'_2 = \left[\int_0^T e^{At} dt\right] B_2$$

avec T = période d'échantillonnage.

Exemple :

$$G(p) = \frac{a}{p+a} \qquad \text{c'est-à-dire } \dot{y} = -ay + au$$

donc $y_{k+1} = e^{-aT} y_k + (1 - e^{-aT}) u_k$

d'où : $$\frac{Y(z)}{U(z)} = \frac{1 - e^{-aT}}{z - e^{-aT}}$$

4. PERFORMANCES EN REGIME PERMANENT

Dans ce paragraphe, on décrit ce qui va constituer le cahier des charges d'une synthèse d'asservissement en termes d'objectifs de poursuite (erreur entrée-sortie nulle) et/ou de régulation (rejet de perturbation). Le principe du modèle interne de Francis & Wonham constitue l'outil de base de cette étude.

4.1. PRINCIPE DU MODELE INTERNE [FRANCIS & WONHAM, 79]

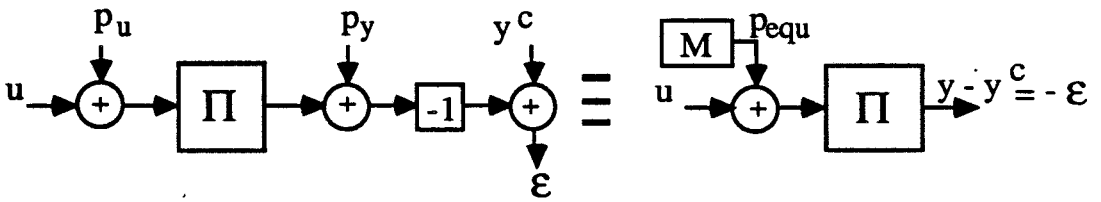
C'est une procédure qui permet, dans le cadre de la théorie des systèmes linéaires, de fixer la structure minimale de correction à adopter pour suivre une classe de trajectoires de consigne sans erreur. Ceci est valable même en présence de perturbations dont on connait le processus

générateur [cf. Annexe 3].

Dans le cas monovariable ce principe s'exprime de façon constructive comme suit :

1 - on ramène sur l'entrée du processus toutes les perturbations et l'entrée de consigne : on obtient un modèle **M** sans entrée mais dont les conditions initiales génèrent une sortie (Fig. B.III.7.) ;

2 - on intègre dans une structure de commande en réseau correcteur un système $\mathbf{M}^+$ qui contient toutes les parties non stables de **M**. Cette partie de la loi de commande doit être commandable par l'erreur ($\varepsilon = y^c - y$) et observable par la commande u.



On ramène les perturbations sur l'entrée
Fig. B.III.7.

3 - on stabilise le système contenant à la fois $\mathbf{M}^+$ et le processus.

L'effort préalable à l'application de cette méthode consiste à modéliser les signaux de perturbations, et les différentes composantes du système pour pouvoir les représenter comme une perturbation de l'entrée de commande. Le problème restant à régler est un problème de stabilité.

Exemple1 :

Soit le processus A (Fig. B.III.8.) soumis à une perturbation p_0 sur la sortie, à aucune perturbation sur l'entrée et dont la consigne y^c est constante par morceaux. La structure minimale de commande peut être un gain statique (unitaire) puisque la perturbation constante $p_0 - y^c$ ramenée sur l'entrée est nulle.

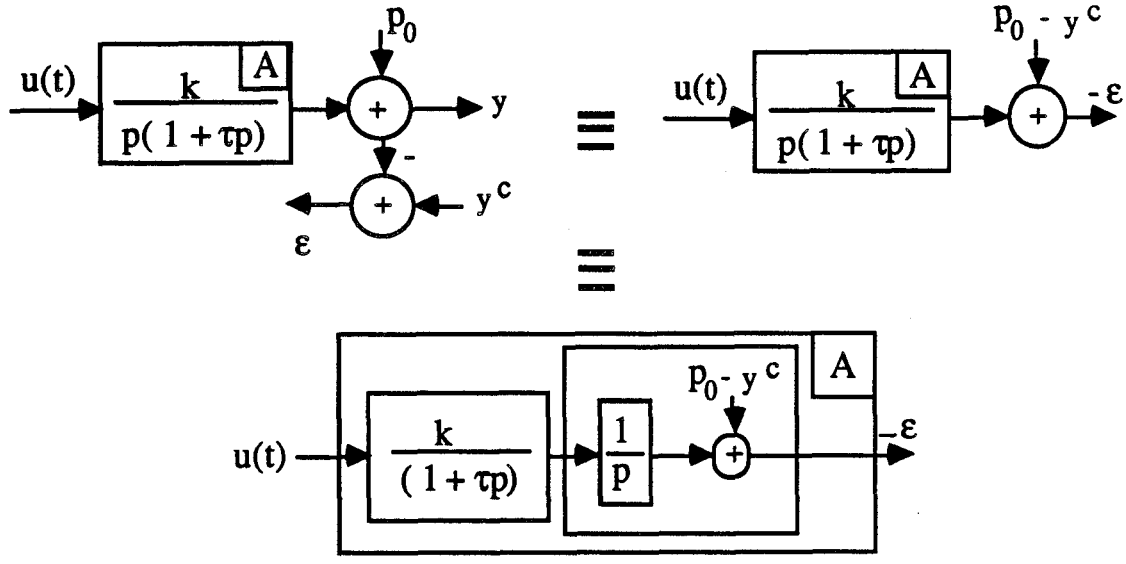


Fig. B.III.8.

Exemple2 :

Pour contrôler le processus B on doit placer un intégrateur en amont de u, car la perturbation constante $p_0 - y^c$ ramenée sur l'entrée reste toujours constante.

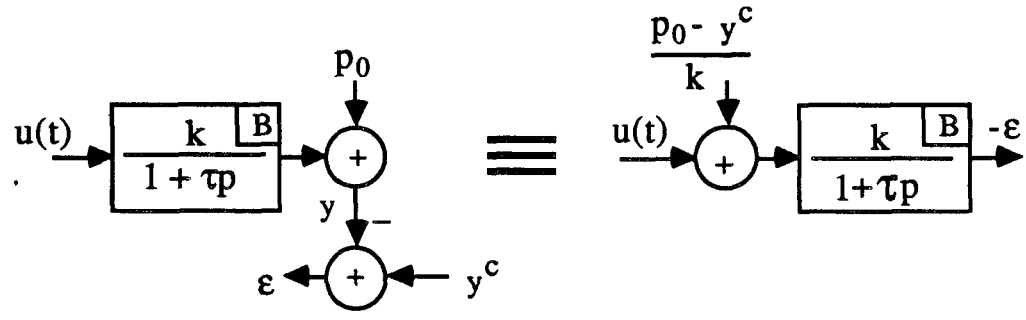


Fig. B.III.9.

Dans le cas particulier où les signaux de consigne et de perturbations sont polynomiaux. On aboutit au résultat classique du paragraphe suivant.

4.2. NOMBRE D'INTEGRATEURS D'UN RESEAU CORRECTEUR

Considérons la figure B.III.10. et examinons comment choisir G afin d'avoir des écarts statiques nuls pour des consignes et des perturbations variant en échelon ou en rampe. Les transmittances de la figure B.III.10. sont définies en temps continu ou discret.

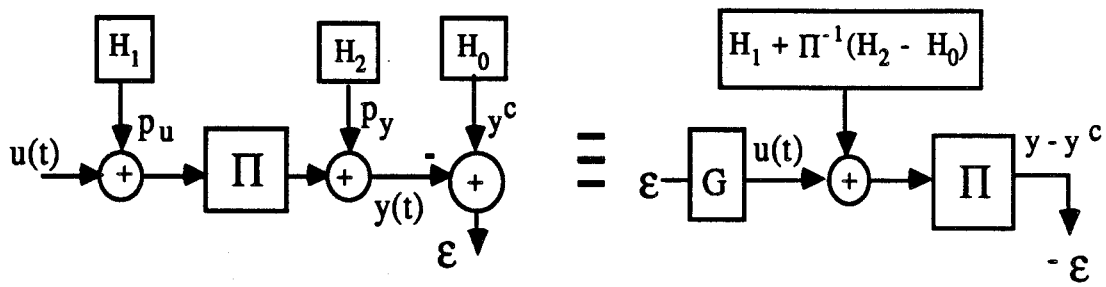


Fig. B.III.10.

D'après le principe du modèle interne, le bloc G doit pouvoir générer le signal opposé à celui issu de $M = H_1 + \Pi^{-1}(H_2 - H_0)$. Donc, le nombre d'intégrateurs de G est fonction du type de y^c, p_u, p_y et du nombre d'intégrateurs de Π (processus à contrôler), d'où la formule suivante :

$$\text{Nbre\_Integ.(G)} = \text{Max}\ \{\text{type}(p_u)\ ;\ (\text{Max}\ \{\text{type}(y^c)\ ;\ \text{type}(p_y)\ \}$$

$$- \text{Nbre\_Integ.}(\Pi))\}$$

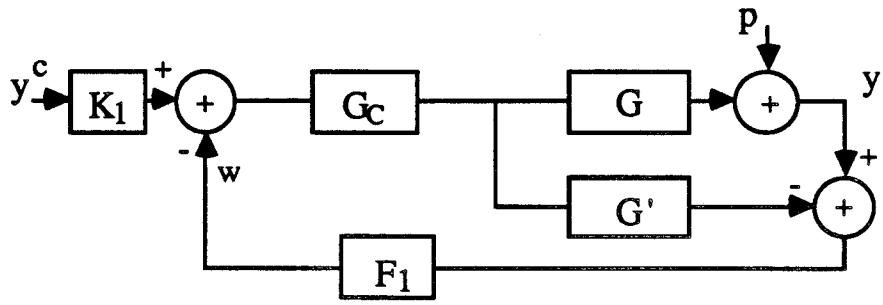
Une commande qui possède un intégrateur et un seul est dite commande de type intégral.

La commande par modèle interne est un cas particulier intéressant de commande de type intégral [DUFOUR 86].

4.3. COMMANDE PAR MODELE INTERNE

Cette technique s'applique à des procédés naturellement stables ou préalablement stabilisés.

Du point de vue de la synthèse, la méthode liée à la structure de commande par modèle interne (CMI) est très simple puisqu'elle consiste à faire d'abord la commande en boucle ouverte du modèle puis mettre en contre-réaction l'erreur de comportement processus-modèle à travers le filtre dit de **robustesse**. Le choix de ce dernier permet en effet d'ajuster le compromis souhaité entre performance et robustesse aux erreurs de modélisation [GARCIA & MORARI 82/85]. La synthèse en boucle ouverte est généralement abordée par le concept d'inversion de modèle. Le contrôleur obtenu en inversant le modèle du procédé est qualifié de **parfait** puisqu'il assure une copie parfaite de la trajectoire désirée, lorsque cette inversion est possible.



Structure de CMI
Fig. B.III.11.

- G est le processus à contrôler ;
- G' est le modèle du processus ;
- G_c est le régulateur en cascade ;
- K_1 est le modèle de référence qui permet le lissage de la commande u
- F_1 est le filtre qui rend la structure de commande robuste quandG' représente mal G.

Les propriétés essentielles de la commande par modèle interne sont issues de la considération du cas nominal où la modélisation est parfaite G = G'. Dans ce cas, le signal de feedback w est égal à p et n'introduit donc aucune rétroaction. On est donc dans un cas de commande en boucle ouverte. $G_c = (G')^{-1}$ est le régulateur parfait qui est irréalisable dans la plupart des cas.
Pour remédier à ce problème, on factorise G' sous la forme : $G' = G_- G_+$. Dans cette expression G_+ contient le retard et les zéros instables de G qui peuvent rendre G_c instable et irréalisable. En choisissant le gain statique de G_+ égal à 1, on est amené à prendre $G_c = (G_-)^{-1}$.

Dans le cas où G'(z) a n pôles stables et (n - 1) zéros stables, on peut adopter $G_c = z^{-1}(G'(z))^{-1}$ qui est stable et réalisable [ZAFIROU 85].

Si on considère le schéma suivant correspondant à une structure classique de boucle fermée (réseau correcteur) :

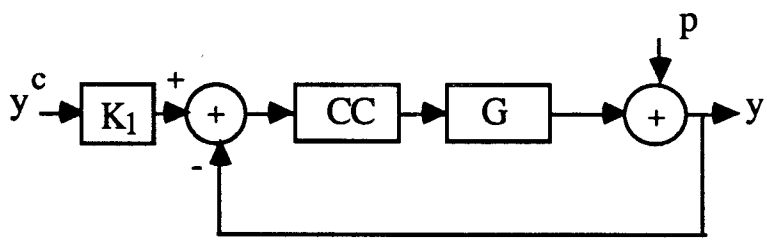


Fig. B.III.12.

L'équivalence de ce schéma avec celui de la figure B.III.11. nous donne dans le cas où $F_1 = 1$:

$$CC = \frac{G_c}{1 - G'G_c} \Longrightarrow \text{Gain\_statique (CC)} = \frac{\text{Gain\_statique (} G_c)}{1 - \text{Gain\_statique } (G'G_c)}$$

Dans la CMI on prend toujours le Gain_statique de $G'G_c$ égal à 1, ce qui nous donne un Gain_statique de CC infini. D'où l'effet intégrateur de la CMI.

Nous pouvons conclure que pour un système stable dont la loi de commande doit être de type intégral, la CMI constitue une solution intéressante par rapport à un PI classique. Elle permet d'avoir une erreur statique nulle en utilisant une procédure de synthèse simple et systématique.

La commande des systèmes à retard pur par prédicteur de Smith constitue un cas particulier de ce type de commande.

4.4. CORRECTION DES SYSTEMES A RETARD PUR

Les processus présentant des retards purs ont des transmittances de la forme :

$$G(p) = G_1(p)e^{-pT_r}$$

Si l'on était capable de réguler G(p) selon le schéma de la figure B.III.13., la mise au point de la boucle pourrait s'effectuer en considérant $R_1(p)G_1(p)$ comme fonction de transfert à retour unitaire.

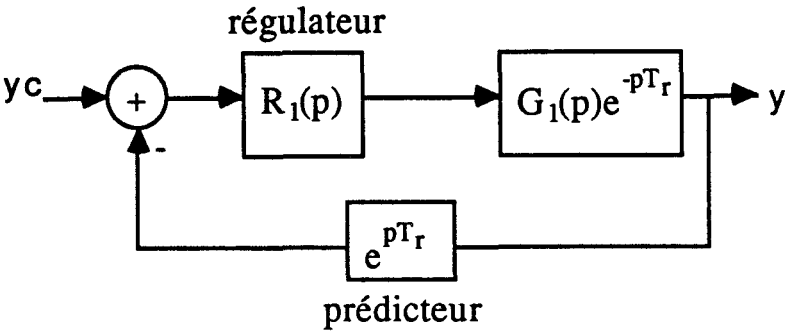


Fig. B.III.13.

Malheureusement, la prédiction e^{pTr} est impossible à réaliser. On peut cependant vérifier que le schéma de la figure B.III.14. est équivalent à celui de la figure B.III.13. avec, cette fois, un régulateur prédicteur parfaitement réalisable.

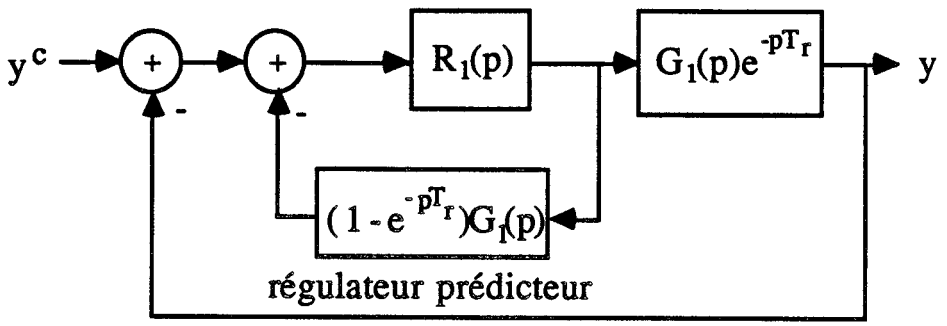


Fig. B.III.14.

La figure B.III.14. donne en effet :

$$\frac{y(p)}{y^c(p)} = \frac{R_1(p)G_1(p)e^{-pT_r}}{1 + R_1(p)G_1(p)}$$

On obtient le même résultat que celui de la figure B.III.13.

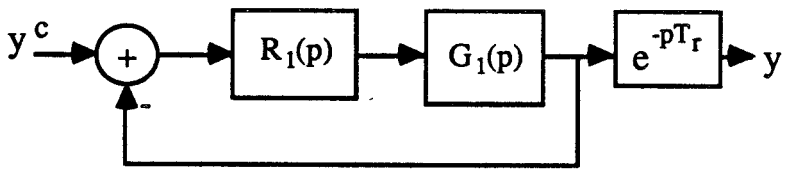


Fig. B.III.15.

On détermine donc $R_1(p)$ comme si la boucle était celle de de la figure B.III.15.

Ainsi la synthèse d'un processus à retard pur se fait en deux étapes. On commence par l'étude du système sans retard pur, puis on intègre le correcteur correspondant ($R_1(p)$) dans une structure de prédicteur de Smith R(p) [DELARMINAT 77].

$$R(p) = \frac{R_1(p)}{1 + (1 - e^{-pT_r})G_1(p)R_1(p)}$$

5. COMPORTEMENT DYNAMIQUE

5.1. INTRODUCTION

Le paragraphe précédent nous a permis de montrer comment

introduire une structure de commande telle qu'aux états stationnaires de l'asservissement construit, correspondent des erreurs identiquement nulles en poursuite et en régulation.

Le problème de précision posé a donc été déplacé en un problème de stabilité de ce comportement nominal.

Dans le cas des systèmes linéaires, cette propriété s'étudie globalement sur le système dont tous les comportements asymptotiques ont la même propriété de stabilité.

La caractérisation de la stabilité asymptotique s'opère dans le domaine fréquentiel par le critère de Nyquist et les notions de marge de stabilité ou, de façon équivalente, par la localisation des pôles de l'asservissement.

Chacune de ces deux approches peut être utilisée dans son domaine d'application :

- Les méthodes fréquentielles sont peu sensibles à l'ordre d'un système mais ne sont pas associées à des méthodes de synthèse directe.

- Les méthodes raisonnant sur les pôles et les zéros réclament une bonne connaissance du modèle et de la dynamique du système, mais elles permettent des synthèses directes de contrôleurs.

En utilisant ces techniques on peut modifier, pour les améliorer, certains comportements mais pas d'autres. Il convient donc de faire le tri entre les comportements dynamiques non-compensables qui sont des contraintes, et les autres qui font l'objet de l'étude.

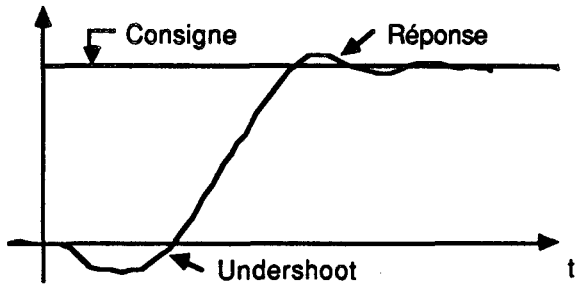
5.2. LES COMPORTEMENTS DYNAMIQUES NON COMPENSABLES

5.2.1. Les retards

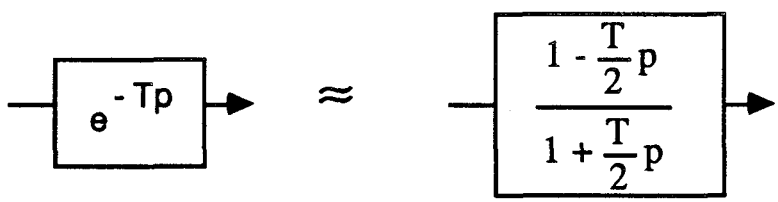
Ils ne peuvent se compenser de façon causale dans une structure en boucle fermée. Une commande en boucle ouverte avec un rôle d'anticipation, peut dans certains cas être envisagée. Par exemple, lorsqu'on utilise les prévisions météorologiques pour compenser de façon anticipée une brusque variation de la température extérieure.

5.2.2. Les déphasages non-minimaux

Ce sont des systèmes d'inverse instable. D'un point de vue externe, on peut les caractériser par des comportements proches du retard ou même par un "under-shoot" lors d'un changement de point de consigne.



Comportement typique d'un système à déphasage non-minimal
Fig. B.III.16.



Approximation d'un retard par un système rationnel à déphasage non-minimal
Fig. B.III.17.

Ces phénomènes sont non-compensables puisque la commande qui les éliminerait, contiendrait l'inverse du système et serait donc instable.

5.2.3. Saturation et limitation de la dynamique

Rappelons qu'une contrainte sur la puissance de commande se traduit par le fait qu'on ne peut imposer une dynamique arbitrairement rapide au système étudié.

5.3. Les comportements dynamiques compensables

D'après le paragraphe précédent et le principe du modèle interne (cf. B.4.1.) on sait que la précision en régime permanent d'un système est obtenue au prix de l'ajout, dans le contrôleur, d'un système non stable correspondant à la partie non évanescente des signaux. On peut donc se demander si effectivement les régimes permanents, que l'on a supposés tout au long du paragraphe précédent, existent vraiment. Autrement dit, lorsque

l'on insère dans un régulateur un fort gain statique et des intégrateurs, le système bouclé reste-t-il **stable?**

5.3.1. Stabilité des systèmes bouclés

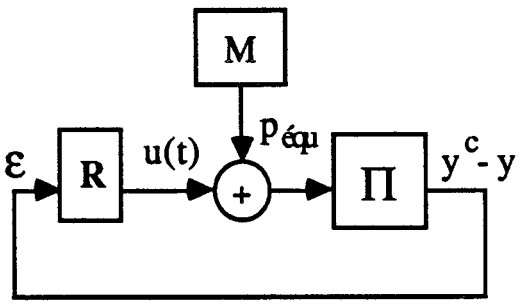


Fig. B.III.18.

- R est le régulateur ;
- M est le processus générateur de la consigne et des perturbations ;
- Π est le processus à contrôler.

Il est hors de question dans ce paragraphe de traiter de façon exhaustive la stabilité des systèmes bouclés, mais nous allons considérer quelques cas particuliers où la solution au problème de stabilité peut se formuler à partir de règles. Dans le cas d'un processus Π (Fig. B.III.18.) :

règle R1 :

Si le processus Π est stable en boucle ouverte et le nombre d'intégrateurs du régulateur (R) est égal à 0, il suffit de le boucler par un petit gain (R = G). Nous proposons l'heuristique suivante pour calculer une valeur admissible de G :

1ère étape : évaluer le domaine $[0 , G_{max}]$, qui assure la stabilité du système en boucle fermée avec un gain G, par dichotomie.

2ème étape : chercher le gain G qui assure le plus petit temps de réponse dans le domaine $]0 , G_{max}]$.

règle R2 :

Si le processus Π est stable en boucle ouverte et le nombre d'intégrateurs du régulateur (R) est égal à 1, il suffit de le boucler par une action proportionnelle intégrale ($R = G(1 + 1/pT_i)$).

Dans ce cas les étapes du calcul sont :

1ère étape : on cherche par tentative un G_0 stabilisant en utilisant R1.

2ème étape : on augmente le terme $T_i^{(-1)}$ de 0 jusqu'à $T_0^{(-1)}$ où on atteint la limite de stabilité, on peut alors adopter :

$$R = G_0(1 + \frac{1}{2pT_0})$$

règle R3 :

Si le processus Π est à déphasage minimum en boucle ouverte, qui ne présente pas de retard et le nombre d'intégrateurs du régulateur (R) est égal à 0, on peut remédier au problème de stabilité en le bouclant par un grand gain (R = G). Comme dans le cas précédent **R1**, on fait la recherche de G stabilisant par tentative

1ère étape : évaluer par dichotomie le domaine $[G_{min}, \infty[$, qui assure la stabilité du système en boucle fermée avec un gain G.

2ème étape : chercher le gain G qui assure le plus petit temps de réponse dans le domaine $[G_{min}, \infty[$.

règle R4 :

Si le processus Π est à déphasage minimum en boucle ouverte, qui ne présente pas de retard, qui peut être instable et le nombre d'intégrateurs du régulateur (R) est égal à 1, on peut remédier au problème de stabilité en lui mettant en feedback une action proportionnelle intégrale ($R = G(1 + 1/pT_i)$). Dans ce cas, les étapes du calcul sont :

1ère étape : on cherche par tentative un G_0 stabilisant en utilisant R3.

2ème étape : on augmente le terme $T_i^{(-1)}$ de 0 jusqu'à $T_0^{(-1)}$ où on atteint la limite de stabilité, on peut alors adopter :

$$R = G_0(1 + \frac{1}{2pT_0})$$

règle R5 :

Si le processus Π est de type Ziegler-Nichols ($k\frac{e^{-pT_r}}{p}$) et le nombre d'intégrateurs du régulateur (R) est égal à 0, il suffit de le boucler avec un gain (R = G), avec :

- $G = \frac{1}{kT_r}$ dans le cas continu ;

- $G = \frac{1}{k(T_r + T)}$ dans le cas discret ;

où T = période d'échantillonnage.

règle R6 :

Si le processus Π est de type Ziegler-Nichols et dont la loi de commande doit être de type intégral, on peut lui mettre en feedback une action proportionnelle intégrale :

- dans le cas continu $R = G(1 + \frac{1}{pT_i})$

avec $G = \frac{0.9}{kT_r}$ et $T_i = 3.3T_r$;

- dans le cas discret $R = G + \frac{K_I}{1 - z^{-1}}$

avec $G = \frac{0.9}{k(T_r + 0.5T)} - 0.5K_I$ et $K_I = \frac{0.27T}{k(T_r + 0.5T)^2}$;

il faut noter que ces formules supposent que $T_r/T \geq 0.5$.

règle R7 :

Si le processus Π est composé d'un intégrateur en série avec un système stable et le nombre d'intégrateurs du régulateur (R) est égal à 0, nous pouvons lui mettre en feedback une action proportionnelle (R = G).

règle R8 :

Si le processus Π est composé d'un intégrateur en série avec un système stable et le nombre d'intégrateurs du régulateur (R) est égal à 1, nous pouvons lui mettre en feedback une action proportionnelle intégrale ($R = G(1 + 1/pT_i)$).

règle R9 :

Si le processus Π est composé d'un intégrateur en série avec un système de type premier ordre (Fig. B.III.19.) et le nombre d'intégrateurs du régulateur (R) est égal à 0, nous pouvons lui mettre en feedback une action proportionnelle (R = G) qu'on calcule de

façon à ce que le système asservi soit critique. Un système critique est un système qui se comporte comme un système de second ordre (5.1) dont le cœfficient d'amortissement est $\zeta = 1$.

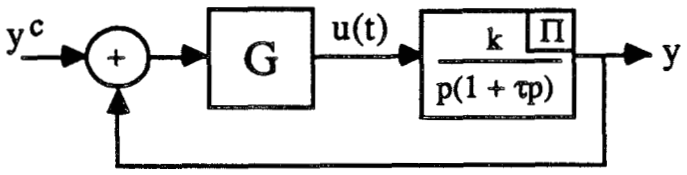


Fig. B.III.19.

$$F(p) = \frac{\omega^2}{\omega^2 + 2\zeta\omega p + p^2} \qquad (5.1)$$

$$\frac{y}{y^c} = \frac{Gk}{Gk + p + \tau p^2} \qquad (5.2)$$

Le calcul est tout simple, en comparant (5.2) à (5.1) on a la valeur de G qui nous donne un système asservi critique :

$$\boxed{G = \frac{1}{4\tau k}}$$

règle R10 :

Si le processus Π est composé d'un intégrateur en série avec un système de type premier ordre (Fig. B.III.20.) et le nombre d'intégrateurs du régulateur (R) est égal à 1, nous pouvons lui mettre en feedback une action proportionnelle intégrale ($R = G(1 + 1/pT_i)$) qu'on calcule de façon à ce que le système asservi ait 3 pôles confondus (5.3).

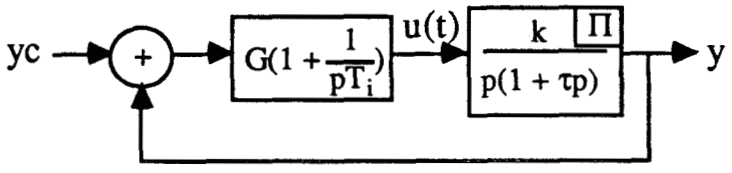


Fig. B.III.20.

$$F(p) = \frac{1 + T_i p}{(1 + \alpha p)^3} \qquad (5.3)$$

$$\frac{y}{y^c} = \frac{Gk(1+T_ip)}{T_ip^2(1+\tau p)+Gk(1+T_ip)} \qquad (5.4)$$

En comparant (5.3) à (5.4) on peut déduire les valeurs des paramètres (G ,T_i) du régulateur. D'où :

$$\boxed{\begin{aligned} T_i &= 9\tau \\ G &= \frac{1}{3\tau k} \end{aligned}}$$

règle R11:

Si le processus Π a toutes ses variables d'état mesurables et son modèle d'état connu, on peut utiliser une commande par retour d'état stabilisant. Comme exemple de calcul d'une commande par retour d'état, on a la commande linéaire quadratique [FOULARD 87][DELARMINAT 77].
Dans le cas de système linéaire discret à n variables d'état, 1 entrée, 1 sortie, et défini par sa représentation d'état discrète :

$$\begin{aligned} x(i+1) &= Ax(i) + Bu(i) \\ y(i) &= Cx(i) + Du(i) \end{aligned}$$

la commande recherchée, qui est donc une suite de u(i), doit minimiser un critère quadratique que nous noterons :

$$J = \sum_{i=0}^{N-1} \{Ru^2(i) + Q\varepsilon^2(i)\}$$

avec ε(i) comme écart entre la consigne y^c et la sortie y(i). R et Q sont deux scalaires positifs.

règle R12 :

Si du processus Π, on ne connaît que le modèle d'état (sans avoir de précision sur la mesurabilité de ses variables d'état), on ne peut faire que du placement de pôles [ASTRÖM 84 c, pp. 221-253][DELARMINAT 89]. Si on considère le schéma de la figure B.III.18. avec $R = R_1/R_2$, Π = N/D et H la fonction de transfert de la boucle fermée ($H = NR_1/(DR_2 + NR_1)$).
Le but est d'avoir les racines de $DR_2 + NR_1$ dans le domaine de stabilité (1/2 plan gauche en p, cercle unité en z).
Pour cela, la technique du placement de pôles consiste à spécifier

arbitrairement un polynôme stable E(q) et à calculer R_1 et R_2 en sorte que :

$$DR_2 + NR_1 = E$$

règle R13 :

Si le correcteur du processus Π doit être d'ordre 2 ou plus, on ne sait faire que du placement de pôles ou de la commande linéaire quadratique. Il faut alors disposer d'un **bon modèle** (expression exacte de la fonction de transfert) et/ou de la mesurabilité de l'état.

Remarque importante :

On a vu dans le paragraphe B.III.4.3. que pour un système stable en boucle ouverte et dont la loi de commande est de type intégral, il vaut mieux utiliser la C.M.I. Nous insistons sur ce choix car l'utilisation de ce type de commande ne déstabilise pas le système corrigé.

règle R14 :

Si le processus Π est stable en boucle ouverte et le nombre d'intégrateurs du régulateur (R) est égal à 1, il suffit d'utiliser une structure de Commande par Modèle Interne (C.M.I).

Avant de conclure ce paragraphe, il faut signaler que nous ne prétendons pas résoudre tous les problèmes de stabilité avec les solutions proposées ci-dessus. Ceci dit, l'utilisation de la programmation déclarative nous permet facilement d'intégrer d'autres méthodes au fur et à mesure qu'elles deviennent d'une mise en œuvre simple.

6. LA ROBUSTESSE

C'est une notion qu'on ne sort du flou qu'au prix d'une étude nécessitant, outre la connaissance d'un modèle, celle des incertitudes concernant ce modèle.

Ce concept important ne sera pas traité dans cette étude hormis dans le cadre de la commande par modèle interne, dans laquelle on dispose d'un résultat dont l'utilisation est immédiate.

La Commande par Modèle Interne d'un processus stable tolère des erreurs de modélisation arbitraires, à condition d'augmenter suffisamment la constante de temps du filtre de régulation (Fig. B.III.11.) [GARCIA &

MORARI 82/85].

7. CONCLUSION

Nous pouvons conclure ce paragraphe en résumant les différentes étapes d'une synthèse d'asservissement. Ainsi nous devons :

- désigner la sortie à contrôler y ;
- désigner la consigne y^c de cette sortie ;
- désigner l'entrée de commande u ;
- trouver les signaux mesurables $\{x_i\}$ (en série entre u et y) en cascade avec y ;
- pour chaque x_i de la cascade, dresser la liste des entrées de perturbations $\{p_j\}$ s'exerçant entre x_i et u et donc compensables à ce niveau ;
- choisir entre commande analogique ou échantillonnée. Dans ce dernier cas, proposer une période d'échantillonnage à partir du calcul du temps de réponse du système {u -> y} ;
- pour chaque boucle $\{u \rightarrow x_i\}$, déterminer le processus générateur des perturbations et consigne ramenées sur l'entrée ;
- déterminer la structure de commande minimale, qui correspond à ajouter dans chaque boucle B_i le modèle de la partie non asymptotiquement stable des perturbations ;
- stabiliser la boucle B_i ;
- compenser les perturbations mesurables en boucles ouvertes (§ B.III.2.3.)

IV CONCLUSION

Nous avons rappelé dans ce chapitre les différents outils nécessaires à l'élaboration d'une synthèse de loi de commande.

Dans le but de réaliser un système à base de connaissances en C S C A O, nous allons traduire dans le chapitre suivant la connaissance décrite précédemment sous une forme propre à son exploitation par un outil logiciel.

Chapitre - C -

SPECIFICATION ET REALISATION D'UN SYSTEME EXPERT EN C S C A O

Chapitre C

I INTRODUCTION 109

II LES OUTILS INFORMATIQUES POUR LA REALISATION DU SYSTEME EXPERT 110

1. DESCRIPTION DE SPT 110

1.1. MOTEUR D'INFERENCE 111

1.2. BASE DE CONNAISSANCES 111

1.2.1. Faits 111

1.2.2. Règles 112

1.3. INTERFACE UTILISATEUR 113

1.4. CONCLUSION 114

2. DESCRIPTION DE SHIRKA 114

2.1. STRUCTURE DU SCHEMA 114

2.2. FACETTES DISPONIBLES 115

2.3. HERITAGE ET SPECIALISATION 116

2.3.1. Héritage 116

2.3.2. Spécialisation 118

2.4. LES MODES D'INFERENCE 118

2.4.1. Attachement procédural 119

2.4.2. Filtre et filtrage 119

2.5. L'INTERFACE UTILISATEUR 121

2.6. CONCLUSION 122

3. CONCLUSION 122

III DESCRIPTION DE LA BASE DE CONNAISSANCES DU SYSTEME EXPERT 123

1. MODULE DE MODELISATION 123

1.1. CLASSE SIGNAL 123

1.2. CLASSE PROCESS 124

1.2.1. La classe processus-élémentaire 125

1.2.1.1. *Gain* 130
1.2.1.2. *Processus* 131

1.2.2. La classe Blocs statiques non linéaires 137

2. MODULE DE SYNTHESE 138

2.1. OBJET SYNTHESE 138

2.2. LA BASE DE REGLES POUR LA SYNTHESE DE LOIS DE COMMANDE 142

2.3. L'OBJET REGULATEUR 144

2.4. L'ALGORITHME DE DETECTION DE CASCADE 145

3. TRANSFERT DES DONNEES DANS LA BASE DE CONNAISSANCES 149

3.1. TRANSFERT DES DONNEES SHIRKA <-> SPT 150

3.2. TRANSFERT DES DONNEES LE_LISP <-> BASILE 150

4. EXEMPLES ET MODE D'EMPLOI 151

IV CONCLUSION 163

I INTRODUCTION

Nous nous proposons d'effectuer la synthèse de commande de processus à partir d'informations **partielles** ou **complètes**.

Ainsi, cela doit permettre à l'utilisateur peu averti d'obtenir une solution standard et robuste à partir de peu d'informations. Une solution plus performante nécessite bien évidemment, plus d'informations.

En suivant les conclusions du chapitre A on peut justifier le choix de systèmes à base de connaissances (utilisant la représentation hybride pour décrire nos connaissances) afin de réaliser un module logiciel capable de satisfaire les hypothèses ci-dessus.

II LES OUTILS INFORMATIQUES POUR LA REALISATION DU SYSTEME EXPERT

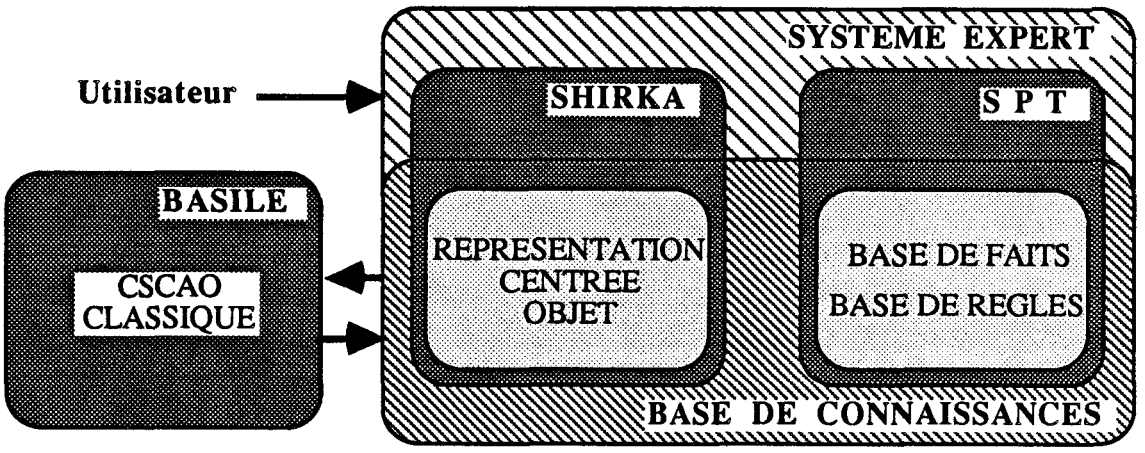


Fig. C.II.1.

Les constituants sont :

- un système expert d'ordre 0^+
  SPT : Système Propositionnel Type [J.P. DELAHAYE]

- un système de gestion de bases de connaissances centrées objets
  SHIRKA : [F. RECHENMANN]

- un progiciel de C S C A O
  BASILE : [INRIA/SIMULOG]

L'assemblage des modules est assuré par des programmes écrits en LE_LISP de l'INRIA. Nous avons fait ce choix pour deux raisons principales, d'une part les modules (SPT, SHIRKA) intégrés dans notre maquette sont écrits en Le_LISP. D'autre part ce langage est facilement portable, d'une machine à une autre, ce qui nous a permis d'installer facilement sur VAX/VMS, SPT conçu sur IBM PC et SHIRKA développé sur MacIntosh.

1. DESCRIPTION DE SPT

Il est formé d'un moteur d'inférence d'ordre 0^+, qui gère une base de connaissances (faits et règles), et d'une interface utilisateur pour éditer et/ou utiliser la base de connaissances [DELAHAYE 87][DUBOIS 87].

1.1. MOTEUR D'INFERENCE

Il nous offre le choix de faire du chaînage avant ou du chaînage mixte.

1.2. BASE DE CONNAISSANCES

1.2.1. Faits

On dispose de quatre types de faits :

- type booléen : La valeur de ce fait est : vrai ou faux.

Exemple : stable ----> le fait "*stable*" est vrai
non mesurable ----> le fait "*mesurable*" est faux

- type symbolique : prend comme valeur une chaîne de caractères qui appartient à un ensemble fini (domaine) décrit par l'utilisateur.

Exemple : le fait type_de_model peut avoir un domaine = {ordre-0, ordre-1, ordre-2, ordre-2-oscillant, Ziegler-Nichols}

type_de_model = Ziegler-Nichols

- type réel : prend comme valeur un nombre réel. On peut faire des comparaisons et des calculs sur les faits réels en utilisant les opérateurs : <, >, >=, <=, <>, =, *, /, +, -, $\sqrt{-}$, max.

Exemple : nbre_intégrateur = (n_c - n_p)
nbre_intégrateur, n_c et n_p sont des faits réels.

- type act_ex : on a créé ce quatrième type pour pouvoir exécuter des fonctions lisp, à partir de la base de connaissances. Ce fait peut être vrai ou faux. Quand il est vrai, le système doit évaluer la liste définie par l'expert, correspondante à une action donnée.

Exemple : calcul_racines est un fait act_ex, la liste d'action est : (racines_équation). Quand le fait calcul_racines est vrai, on aura l'activation de la fonction racines_équation.

A chaque instant d'une session un fait peut être :

- connu si une valeur lui a été attribuée par l'utilisateur ou déduite

par le système ;

- inconnu si on ne lui a pas attribué de valeur et si le système n'a pas cherché à la déduire ;

- indéterminé si la réponse de l'utilisateur à la question posée par le système est "je ne sais pas".

Enfin, le concepteur de SPT a prévu aussi la notion de fait "demandable" et "non demandable". Ce qui permet à l'utilisateur de choisir les faits qui **doivent** être déduits par le système, ce sont les "non demandables" et les faits qui **peuvent** être demandés à l'utilisateur s'ils ne peuvent pas être déduits par le système : ce sont les "demandables".

1.2.2. Règles

Elles sont de la forme :

si < condition > alors < conclusion >

<condition> = <premisse$_1$>^...^<premisse$_i$>^...^<premisse$_n$>
<conclusion> = <action$_1$>^...^<action$_j$>^...^<action$_m$>
Dans ce formalisme <premisse$_i$> peut prendre les formes suivantes :

<Fait booléen> (vrai ou faux)
ou <Fait symbolique> (= ou <> d'une valeur symbolique)
ou <Fait réel> (comparé avec une valeur réelle)
ou <Fait act_ex> (vrai ou faux).

La valeur symbolique appartient au domaine défini par l'utilisateur.

La valeur réelle peut être un nombre réel ou le résultat d'une opération sur des réels.

Enfin, on peut aussi avoir comme <premisse$_i$> : un fait (booléen, symbolique, réel ou act_ex) qui peut être égal à une métavaleur (connue, inconnue, indéterminée).

<action$_j$> peut être <Fait booléen> (vrai ou faux)
ou <Fait symbolique> (= une valeur symbolique)
ou <Fait réel> (= une valeur réelle)
ou <Fait act_ex> (vrai ou faux).

Exemple de règle :

si nbre_intégrateur_réseau_correcteur = 1
alors réseau_correcteur = Type_1

1.3. INTERFACE UTILISATEUR

Elle nous permet de choisir entre 2 options : **TRAVAIL** et **EDITION**.

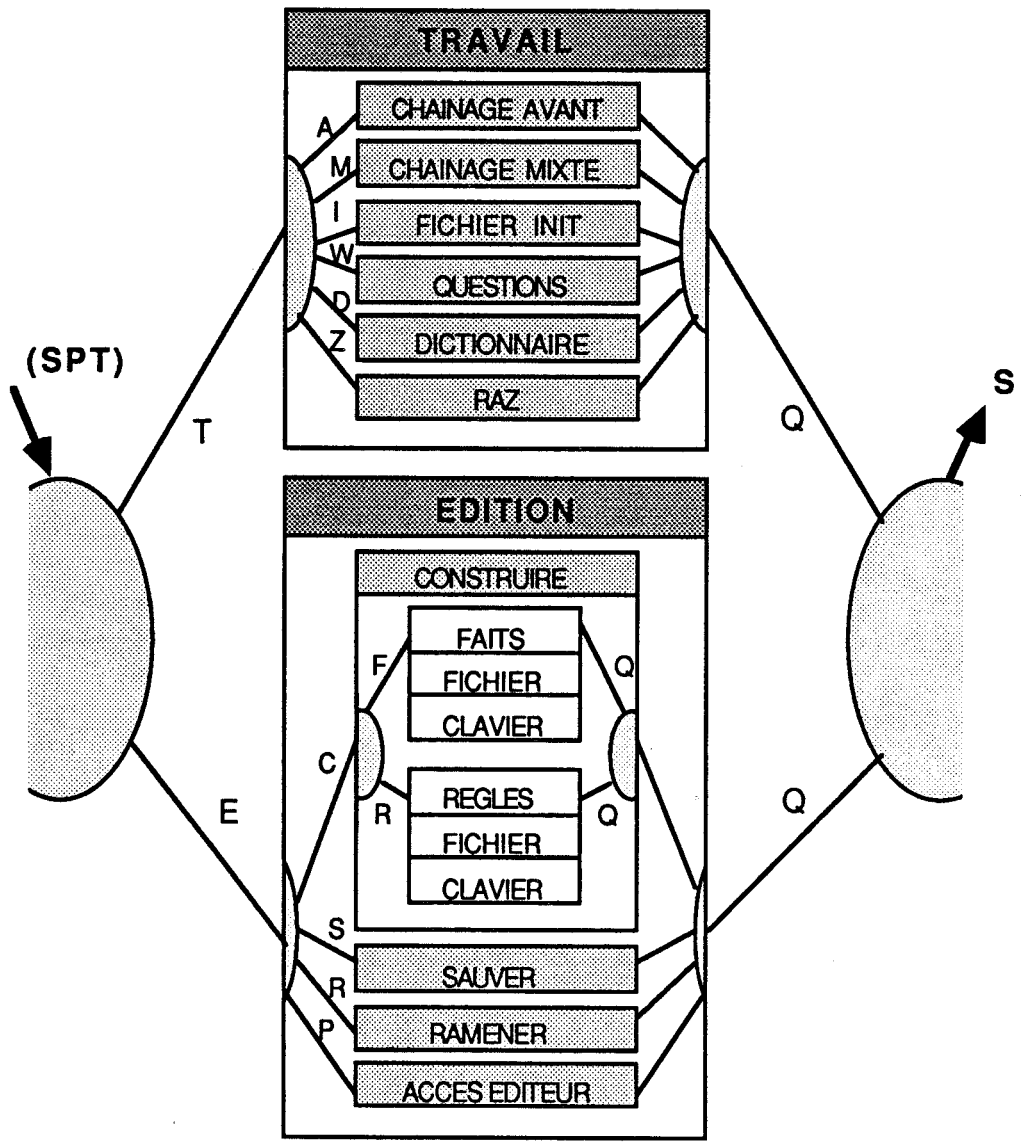


Fig. C.II.2.

TRAVAIL : Sur une base de connaissances chargée ou construite il est possible d'effectuer une inférence en :

- chaînage avant d'après les valeurs accordées à certains faits ;

- chaînage mixte à partir d'un fait (but) ; en laissant le choix à l'utilisateur de voir ou non s'afficher les valeurs affectées à des faits intermédiaires.

Grâce à l'option :

- "FICHIER INIT", un fichier d'initialisation permet de démarrer plusieurs sessions de travail avec les mêmes conditions initiales.

- "QUESTIONS" on peut demander au système les raisons d'affectation d'un fait.

- "DICTIONNAIRE" on peut consulter la mémoire de travail.

- "RAZ" on peut réinitialiser tous les faits avec la métavaleur, "inconnu".

EDITION permet de construire, de sauver et de ramener une base de connaissances. Il donne aussi accès à un éditeur de texte.

1.4. CONCLUSION

SPT est un noyau de système expert que nous avons pu facilement modifier pour l'adapter à nos besoins. Nous avons ainsi rajouté la possibilité de déclencher une procédure comme résultat d'activation d'une règle.

2. DESCRIPTION DU SHIRKA

C'est un système de gestion de bases de connaissances centrée-objet. Dans ce système l'entité conceptuelle "objet" est représentée par un schéma.

2.1. STRUCTURE DU SCHEMA

Un schéma est défini par son nom et décrit sous forme d'un ensemble d'**attributs** qui caractérisent toutes les propriétés de l'objet créé. Par exemple, une date est caractérisée par un jour, un mois et une année. On décrit les attributs par des facettes. Il y a plusieurs types de facettes. Par exemple, nous avons les facettes de restriction pour définir l'ensemble des valeurs que peut prendre l'attribut. A chaque facette correspond une valeur (Fig. C.II.3. a et b).

```
{ objet
    attribut1
        $facette1.1  valeur1.1
        $facette1.2  valeur1.2
        .......
    attribut 2
        $facette2.1  valeur2.1
        $facette2.2  valeur2.2
        ......                    }
```

Fig. C.II.3. a

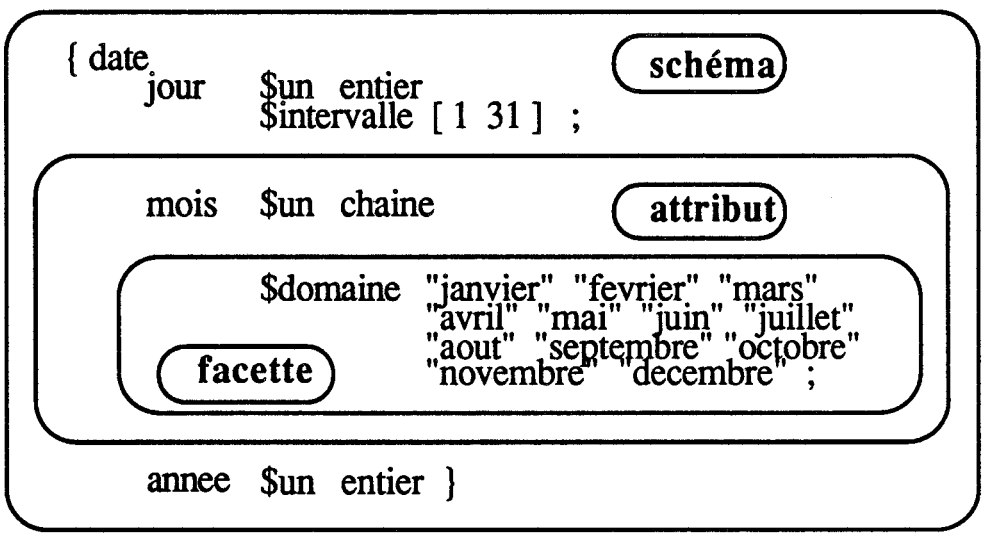


Fig. C.II.3. b

2.2. FACETTES DISPONIBLES

Dans SHIRKA l'utilisateur dispose de plusieurs catégories de facettes :

- facettes de typage :

$un, suivie du nom d'un type simple (entier, réel, chaîne, booléen ou symbole), ou du nom d'une classe. Dans ce dernier cas, les valeurs de l'attribut correspondant sont des instances de cette classe ;

$liste-de, joue le même rôle que **$un**, mais indique un attribut multi-valué ;

- facettes de restriction de type :

$domaine, suivie d'une liste des valeurs possibles de l'attribut ;

$sauf, suivie d'une liste de valeurs que l'attribut correspondant ne peut prendre ;

$intervalle, est suivie d'un ou de plusieurs intervalles. La valeur de l'attribut correspondant doit être située dans l'un de ces intervalles. Cette facette ne peut être associée qu'à un attribut de type simple ;

- facettes de définition et de manipulation des variables :

$var-nom permet d'associer explicitement une variable à un attribut ;

$var-> indique que la valeur de l'attribut peut être transmise à la variable dont le nom suit ;

$var<- indique que l'attribut peut recevoir la valeur de la variable dont le nom suit ;

$var-liste->** semblable à **$var->, utilisée dans le cas d'attribut multi-valué ;

$var-liste<-** semblable à **$var<-, utilisée dans le cas d'attribut multi-valué.

Remarques :

1. Les valeurs affectées à ces facettes sont de même type que celui de l'attribut correspondant ;

2. Nous n'avons défini dans ce paragraphe que les facettes les plus utilisées dans notre étude (pour plus de détails, on se reportera à l'Annexe 4).

2.3. HERITAGE ET SPECIALISATION

2.3.1 Héritage

Dans SHIRKA, un schéma de classe hérite des attributs des schémas de classe qui le dominent à travers le lien **sorte-de**.

Un attribut hérité d'une classe supérieure ne peut être que précisé et non redéfini. Préciser un attribut, c'est lui adjoindre de nouvelles

restrictions ou de nouveaux moyens de détermination de ses valeurs, voire même fixer ses valeurs.

L'utilisateur dispose de deux modes d'héritage : simple et multiple.

Exemple d'héritage simple :

```
{ point
  sorte-de    =            objet ;
  xcoord      $un          reel ;
  ycoord      $un          reel }

{ point-premier-quadrant
  sorte-de    =             point ;
  xcoord      $intervalle  [0.0  +inf] ;
  ycoord      $intervalle  [0.0  +inf] }
```

Exemple d'héritage multiple :

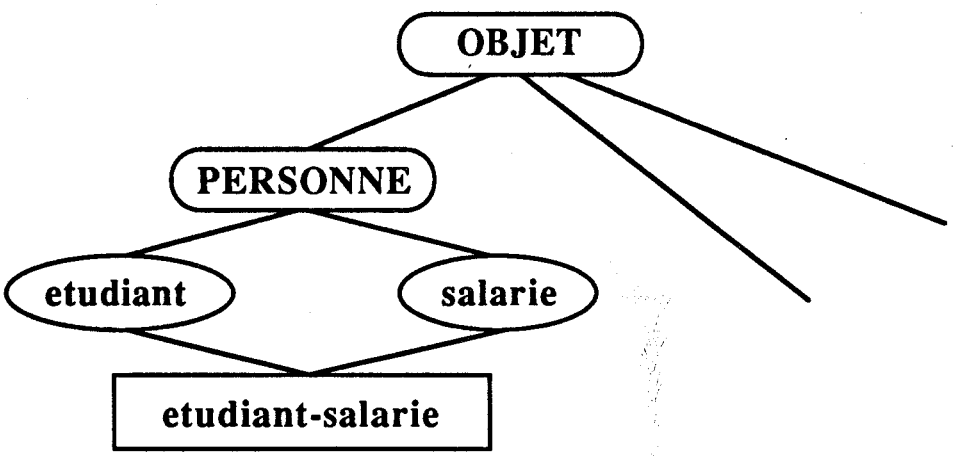


Fig. C.II.4.

```
{ etudiant
  sorte-de    =            personne ;
  age         $intervalle  [17 77] ;
  no-carte    $un          carte ;
  universite  $liste-de    universite }

{ salarie
  sorte-de    =             personne ;
  age         $intervalle   [18  65] ;
  salaire     $un           reel ;
  activite    $un           activite }

{ etudiant-salarie
  sorte-de                  =   etudiant  salarie }
```

```
{ Dupond
  est-un      =        etudiant-salarie ;
  nom         =        "Dupond" ;
  age         =        20 ;
  no-carte    =        carte-20 ;
  universite  =        Lille_1 ;
  salaire     =        4500.00 }
```

2.3.2. Spécialisation

Ce terme désigne à la fois, une classe qui en spécialise une ou plusieurs autres par restriction, et l'action qui consiste à déterminer si l'instance d'une classe donnée peut être considérée comme instance de l'une de ses sous-classes.

Exemple :

```
{ disque-premier-quadrant
  sorte-de              =     disque ;
  centre                =     point-premier-quadrant }

{ pt1
  est-un      =        point ;
  xcoord      =        1.23 ;
  ycoord      =        0.65 }

{ d5
  est-un      =    disque-premier-quadrant ;
  centre      =    pt1 }
```

2.4. LES MODES D'INFERENCE

Dans SHIRKA, l'inférence consiste à déterminer les valeurs manquantes d'attributs dans des instances, en utilisant la connaissance contenue dans les classes, plus particulièrement grâce aux facettes :

- **$valeur**, fournit une valeur d'attribut valable pour toutes les instances de la classe ;
- **$défaut**, la valeur qui suit est retenue comme valeur d'attribut si aucun autre moyen d'inférence n'a fonctionné auparavant ;
- **$sib-exec** pour les attachements procéduraux ;
- **$sib-filtre** pour les filtrages.

2.4.1. Attachement procédural

Il consiste à attacher, par la facette **$sib-exec**, à un attribut d'une classe une procédure, ici une fonction Lisp. A cette fonction est associée une méthode, celle-ci est une spécialisation de la classe méthode dont elle hérite l'attribut **nom-fct**.

Exemple :

```
{ rectangle
   sorte-de    =            objet ;
   longueur    $un          reel
               $intervalle  [0.0  +inf] ;
   largeur     $un          reel
               $intervalle  [0.0  +inf] ;
   surface     $un          reel
               $sib-exec    { produit
                              x      $var<-     longueur ;
                              y      $var<-     largeur ;
                              z      $var->     surface }}

{ produit
   sorte-de    =            methode ;
   nom-fct     $valeur      produit ;
   x           $un          reel ;
   y           $un          reel ;
   z           $un          reel }

(de produit  (inst)
   (affect 'z  (* (val? 'x)(val? 'y))))

{ rectangle_2
   est-un    = rectangle ;
   longueur  = 3.55 ;
   largeur   = 6.20 }
```

Remarque : Pour inférer la valeur de surface de rectangle_2, il suffit d'utiliser la commande **val?** qui cherche la valeur d'un attribut en mode intéractif :

Val? rectangle_2 surface ---> 22.01

2.4.2. Filtre et filtrage

Un filtre est un ensemble de conditions que doivent satisfaire des instances. Il est décrit par un schéma de classe qui porte le nom de celle à

laquelle se rattachent les instances cherchées. Sa structure en est identique, mais chacun de ses attributs peut posséder des restrictions supplémentaires par rapport à celles figurant dans le schéma de classe.

Le filtrage est le mécanisme qui consiste à rechercher les instances de cette classe qui satisfont le filtre et à en extraire, par des facettes **$var->** ou **$var-liste->**, les valeurs de l'attribut auxquelles le filtre est attaché par la facette **$sib-filtre**.

Exemple :

```
{ processus
     sorte-de          =          objet ;
     a-pour-pere       $un        processus ;
     pere-de           $liste-de  processus }
```

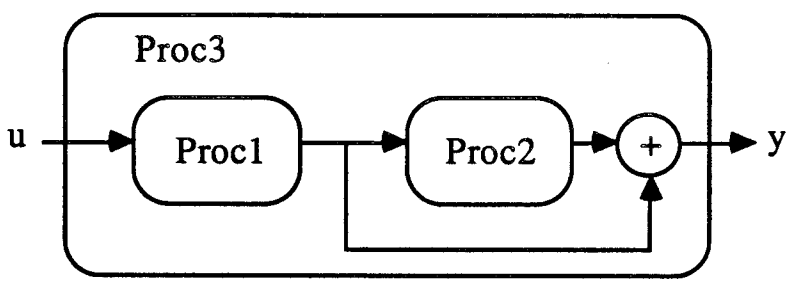


Fig. C.II.5.

```
{ processus_1
     est-un            =    processus ;
     a-pour-pere       =    processus_3 }

{ processus_2
     est-un            =    processus ;
     a-pour-pere       =    processus_3 }

{ processus_3
     est-un            =    processus }
```

```
{ processus
   sorte-de        =          objet ;
   lui-meme        $var-nom   lui ;
   a-pour-pere     $un        processus ;
   pere-de         $liste-de  processus
                   $sib-filtre
                         { processus
                                lui-meme     $var->   pere-de ;
                                a-pour-pere  $var<-   lui}}
```

Val? processus_3 pere-de processus_1 processus_2

Dans cet exemple, l'utilisateur demande au système la liste des instances de la classe processus dont la valeur de l'attribut a-pour-pere est processus_3. Pour résoudre ce problème, le système commence par affecter à la variable **lui** la valeur processus_3. Ensuite, il va chercher dans la liste des instances de processus celles qui ont pour valeur de l'attribut a-pour-pere la valeur lui. Enfin, il va affecter a l'attribut pere-de du filtre la liste des instances qui satisfont cette condition.

Remarque

- l'attribut **lui-meme** est implicitement défini dans tout schéma de classe. Pour une instance de cette classe, sa valeur est l'instance elle-même ;

- les filtres peuvent être imbriqués ;

- on a la possibilité d'avoir plusieurs filtres derrière **$sib-filtre.**

2.5. L'INTERFACE UTILISATEUR

L'utilisateur dispose des commandes suivantes :

charger charge une base de schémas et de fonctions Lisp ;

sauver sauvegarde les instances d'une liste de classes dans un fichier Lisp ;

l-att affiche la liste de tous les attributs, propres et hérités, instanciables et non instanciables, d'une classe ;

l-inst affiche la liste des instances d'une classe ;

l-spec affiche la liste des spécialisations d'une classe ;

cr-inst crée une instance de façon interactive ;

sup-inst supprime une instance ;

vi visualise à l'écran une instance à plusieurs niveaux d'imbrication des valeurs de ses attributs ;

val? cherche à déterminer la valeur d'un attribut, en utilisant le cas échéant les divers mécanismes d'inférence ;

aj-val permet d'ajouter une valeur à un attribut d'instance ;

mod-val permet de modifier une valeur dans un attribut en la remplaçant par une autre ;

sup-val permet de supprimer une valeur dans un attribut ;

exec crée une instance d'un objet méthode en interactif.

2.6. CONCLUSION

SHIRKA est bien adapté au développement d'applications dans lesquelles la connaissance est fortement structurée. Cette connaissance y est organisée autour d'objets et de méthodes, et permet leur appariement : retenir la bonne méthode pour un objet dans un contexte donné [RECHENMANN 87 a et b].

3. CONCLUSION

Il faut noter que la combinaison de connaissances décrites sous forme d'objet-structuré avec celles décrites par une base de faits et de règles (supervisée par un moteur d'inférence d'**ordre 0⁺**) est bien adaptée à la gestion de la connaissance utilisée dans notre étude. Aussi, cette combinaison est recommandée par un certain nombre de chercheurs dans les domaines des systèmes à base de connaissances et de la C A O en automatique [RECHENMANN 88].

III DESCRIPTION DE LA BASE DE CONNAISSANCES DU SYSTEME EXPERT

En nous reportant au chapitre B, on peut décomposer notre base de connaissances en deux modules principaux : **modélisation** et **synthèse**. Ce sont deux étapes indispensables dans une étude de synthèse d'asservissement de lois de commande.

1. MODULE DE MODELISATION

Dans ce module la connaissance est fortement structurée ce qui nous a amené à la décrire sous forme d'objet-structuré en utilisant **SHIRKA**.

On définit deux classes principales : **Signal** et **Process**.

1.1. CLASSE SIGNAL

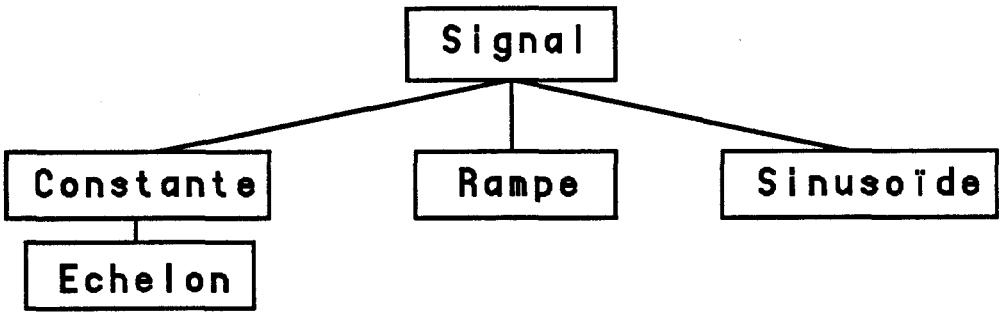


Fig. C.III.1.

Nous avons défini pour la classe signal trois sous-classes, retenues comme prototypes dans nos différentes études dé synthèse de lois de commande : **Constante, Rampe** et **Sinusoïde.**

La **mesurabilité** et l'**unité-de-mesure** sont les deux attributs définis pour la classe **signal.** Le premier est un attribut binaire **faux** par défaut. Le second est de type symbole et il a pour domaine : **volt, seconde, mètre, gramme, bar, degré celsius,** ... etc.

D'après les définitions du B.I, on décrit l'objet :

- **Constante** par l'attribut **amplitude** de type réel ;
- **Echelon** par l'attribut **amplitude**, qu'il hérite de l'objet **constante**, et l'attribut **instant-de-démarrage** de type réel lui aussi ;
- **Rampe** par les attributs **pente** et **instant-de-démarrage** tous

les deux de type réel ;

- **Sinusoïde** par les attributs **amplitude, pulsation** et **phase** tous les trois de type réel.

En exemple d'instance, on peut décrire les signaux constants : **Zéro_volt** et u_0 (déréglage). En langage **SHIRKA** on aura :

```
{ Zéro_volt
    est-un          = constante ;
    unité-de-mesure = volt ;
    mesurabilite    = vrai ;
    amplitude       = 0. }
```

Fig. C.III.2. a

```
{ u0
    est-un = constante }
```

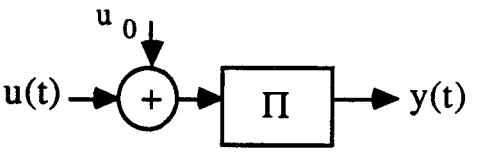


Fig. C.III.2. b

Remarque importante :

Les différentes valeurs des attributs relatifs à un signal sont nécessairement données directement par l'utilisateur et ne peuvent être déduites.

1.2. CLASSE PROCESS

La classe **process** décrit un système (cf. B.II.) à travers son interface avec l'extérieur qu'on caractérise par les attributs **entrée-de-commande** et **sortie**. Ces deux attributs sont monovalués car notre étude est limitée à des blocs mono-entrée/mono-sortie. Ils prennent comme valeur les instances de la classe **signal**.

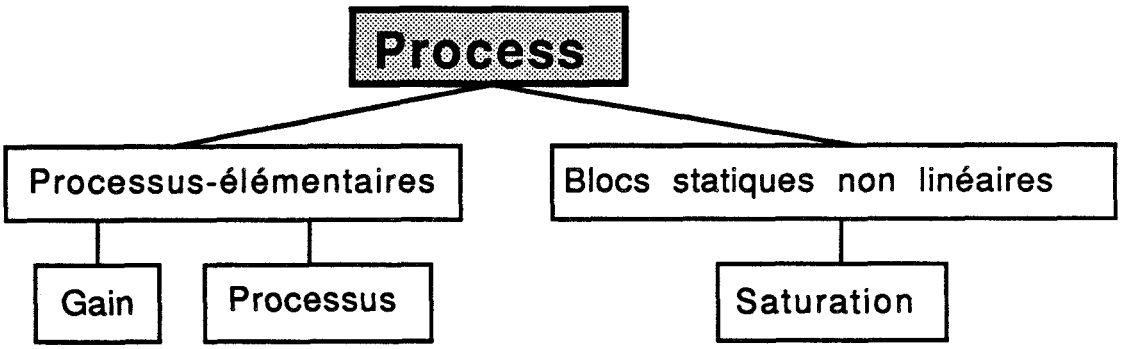


Fig. C.III.3.

Nous décrivons d'une part les process dynamiques grâce à la sous-classe **processus-élémentaires**, qui recouvre les systèmes dynamiques linéaires et stationnaires ; et d'autre part les non-linéarités statiques à travers la sous-classe **blocs statiques non linéaires.**

1.2.1. La classe processus-élémentaire

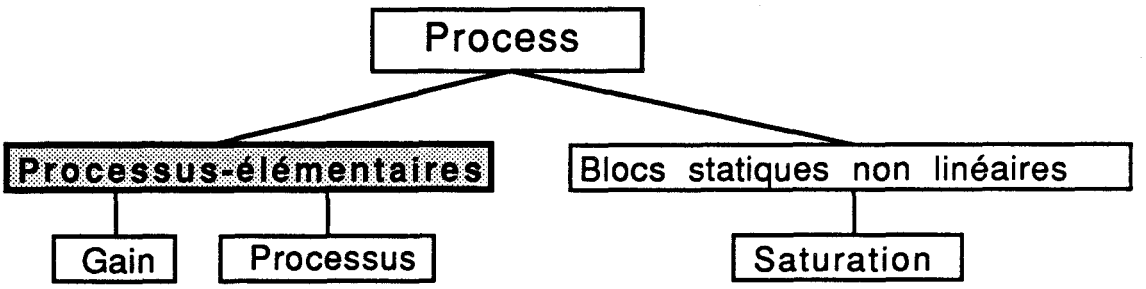


Fig. C.III.4.

Dans cette classe on représente un **processus-élémentaire** à travers les attributs suivants (a1 à a16) :

a1) **Temps** de type symbole peut prendre deux valeurs **continu** ou **discret.** S'il n'est pas instancié par l'utilisateur, il prend par défaut la valeur **continu** ;

a2) **Type-modele-procede** de type symbole a pour domaine { **ordre-0, Ziegler-Nichols, ordre-1, ordre-2-oscillant, ordre-2** }. Cet attribut ne peut être affecté que par l'utilisateur ou calculé à partir d'une description plus fine (Espace d'état ou Fonction de transfert), si elle existe ;

a3) **Matrices-etat** de type réel est évalué directement par l'utilisateur. Il prend comme valeur une liste de réels sous la forme suivante :

(n A_{11} A_{12} ... A_{21} ... A_{nn} B_1 B_2 ... B_n C_1 C_2 ... C_n) avec n, le degré du système, les A_{ij}, B_i et C_i sont respectivement les éléments des matrices A, B et C ;

a4) **Num-Ft** de type réel, prend pour valeur la liste des cœfficients réels du numérateur de la fonction de transfert ;

a5) **Den-Ft** de type réel, prend pour valeur la liste des cœfficients réels du dénominateur de la fonction de transfert ;

Si ces deux derniers attributs ne sont pas affectés par l'utilisateur on peut déduire leur valeur grâce à un attachement procédural mais ceci n'est possible que si l'utilisateur a entré la valeur de **Matrices-etat** (a3).

Cette procédure revient tout simplement à faire le calcul suivant :

Continu	Discret
$\dot{X} = AX + BU$	$X_{n+1} = AX_n + BU_n$
$Y = CX$	$Y_n = CX_n$
$(pI-A)X = BU$	$(zI-A)X_n = BU_n$
$Y = C(pI-A)^{-1}BU$	$Y_n = C(zI-A)^{-1}BU_n$

d'où dans le cas continu $\frac{\text{num-Ft}}{\text{den-Ft}} = C(pI-A)^{-1}B$

et dans le cas discret $\frac{\text{num-Ft}}{\text{den-Ft}} = C(zI-A)^{-1}B$

a6) **Num-Ftd** de type réel, prend pour valeur la liste des cœfficients réels du numérateur de la fonction de transfert discrétisée ;

a7) **Den-Ftd** de type réel, prend pour valeur la liste des cœfficients réels du dénominateur de la fonction de transfert discrétisée ;

Ces deux derniers ne sont pas affectés par l'utilisateur. Le système calcul leur valeur grâce à un attachement procédural qui fait appel à une fonction **Basile** qui discrétise le modèle continu. Ce calcul n'est possible que si le système dispose de la fonction de transfert du processus étudié c'est-à-dire que si **Num-ft** (a4) et **Den-ft** (a5) ont été affectés.

a8) **Temps-de-retard** de type réel, s'il n'est pas instancié par l'utilisateur, prend par défaut la valeur **0**. Un cas particulier se

présente lorsqu'on on étudie un système discret avec n pôles nuls on doit ajouter à la valeur du **Temps-de-retard** l'équivalent de n périodes d'échantillonnage ;

a9) **Zeros-Ft** de type réel, prend pour valeur la liste des racines du numérateur de la fonction de transfert. S'il n'est pas affecté par l'utilisateur, sa valeur peut être déduite grâce à une procédure de calcul mais ceci n'est possible que si la fonction de transfert du système est connue.

a10) **Poles-Ft** de type réel, prend pour valeur la liste des racines du dénominateur de la fonction de transfert. S'il n'est pas affecté par l'utilisateur, on peut déduire sa valeur grâce à une procédure de calcul et ceci n'est possible que si le système connait la fonction de transfert ;

a11) **Nbre-integrateur** de type entier, prend comme valeur le nombre d'intégrateurs de la fonction de transfert. S'il n'est pas attribué par l'utilisateur on peut le déduire au moyen d'un attachement procédural avec la fonction de transfert (si celle-ci est connue). Dans la cas contraire, il prend par défaut la valeur **0** ;

Le résultat du calcul correspondant est la différence entre :

- le nombre de racines nulles du dénominateur et du numérateur dans le cas continu ;

- le nombre de racines égales à 1 du dénominateur et du numérateur dans le cas discret.

a12) **Gain-statique** de type réel, reçoit la valeur du gain statique de la fonction de transfert. S'il n'est pas donné par l'utilisateur on peut le déduire au moyen d'une procédure de calcul à partir de la fonction de transfert :

- dans le cas continu, cela revient à faire le rapport entre le cœfficient non nul du plus faible degré du numérateur et celui du dénominateur ;

- dans le cas discret, cela revient à faire le rapport entre la somme des cœfficients du numérateur et celle des cœfficients du dénominateur.

a13) **Stabilité** de type symbolique, prend comme valeur {**stable**, **instable**}, et peut être affecté directement par l'utilisateur ou

déduit par le système grâce à un attachement procédural :

- dans le cas continu, on lui attribue la valeur **stable** si toutes les parties réelles des pôles sont inférieures à zéro, sinon on lui donne la valeur **instable** (rappelons que nous assimilons stabilité et stabilité asymptotique) ;

- dans le cas discret, on lui affecte la valeur **stable** quand tous les pôles sont en module inférieur à 1, sinon on lui attribue la valeur **instable.**

a14) **Déphasage-minimum** de type symbolique, prend comme valeurs {**mini, pas-mini**} que l'utilisateur peut donner ou que le système peut calculer grâce à un attachement procédural. On fait les mêmes tests que pour l'attribut **stabilité** seulement au lieu de les faire sur les pôles on les effectue sur les zéros de la fonction de transfert. Ainsi au lieu de conclure sur **stable** (**instable**) on conclue sur **mini** (**pas-mini**).

a15) **Temps-de-montée** de type réel. S'il n'est pas donné directement par l'utilisateur on peut le déduire par calcul. Ceci n'est possible que si d'une part on connait les pôles de la fonction de transfert et d'autre part le processus étudié est stable. Ce calcul est [DELARMINAT 75] :

- dans le cas continu, pour un processus :

c2) d'ordre 2 : $F(p) = \dfrac{k}{\omega_n^2 + 2\zeta\omega_n p + p^2}$

c21) si $\zeta = 1$: un pôle double $p = x$ $\qquad T_m = \dfrac{4\sqrt{2}}{|x|}$

c22) si $\zeta < 1$: deux pôles complexes conjugués $p = x \pm iy$ $T_m = \dfrac{4}{|x|}$

c23) si $\zeta > 1$: deux pôles différents $p_1 = x_1$, $p_2 = x_2$

$$T_m = 4\sqrt{\frac{1}{x_1^2} + \frac{1}{x_2^2}}$$

c3) d'ordre $n > 2$ est considéré comme une mise en série de m **processus-élémentaires** d'ordre 1 ou 2 :

$T_m = \sqrt{\sum_{i=1}^{m} T_{m_i}^2}$, T_{m_i} = temps de montée du **processus-élémentaires** i

- dans le cas discret, on fait les mêmes calculs qu'en temps continu (c1, c2 et c3) avec les équivalents continus des pôles discrets définis par :

$$pôle_c = -\frac{1}{T} Log(pôle_d)$$

avec T = période d'échantillonnage que l'utilisateur doit avoir donné au système ;

a16) **Temps-de-réponse** de type réel, sa valeur est affectée directement par l'utilisateur ou bien calculée par attachement procédural. L'opération correspondante revient à sommer la valeur du **Temps-de-retard** (a6) avec celle du **Temps-de-montée** (a15).

Exemples :

Si l'utilisateur crée l'instance Proc1 de la classe processus-élémentaires

```
( proc1
      est-un = processus-elementaire ;
      num-ft = (1.00) ;
      den-ft = (3.00 5.00 6.00) ;
      temps-de-retard = 0.10 ;
)
```

Le système est capable d'inférer ce qui suit :

```
( proc1
      est-un = processus-elementaire ;
      temps = continu ;
      num-ft = (1.00) ;
      den-ft = (3.00 5.00 6.00) ;
      temps-de-retard = 0.10 ;
      poles-ft = (-0.42 -0.57 -0.42 0.57) ;
      nbre-integrateur = 0 ;
      gain-statique = 0.33 ;
      stabilite = stable ;
      temps-de-montee = 4.40 ;
      temps-de-reponse = 4.50 ;
)
```

En partant d'une description minimale suivante :

```
( proc2
     est-un = processus-elementaire ;
     entree-commande = e ;
     sortie = s ;
     temps = discret ;
     matrices-etat = (2.00 -1.50 1.00 0.50 -1.00 1.00 0.00 0.50 1.00) ;
)
```

Le système peut déduire :

```
( proc2
     est-un = processus-elementaire ;
     entree-commande = e ;
     sortie = s ;
     temps = discret ;
     matrices-etat = (2.00 -1.50 1.00 0.50 -1.00 1.00 0.00 0.50 1.00) ;
     num-ft = (0.50) ;
     den-ft = (0.50 1.00) ;
     temps-de-retard = 0.00 ;
     poles-ft = (-0.50 0.00 ) ;
     nbre-integrateur = 0 ;
     gain-statique = 0.33 ;
     stabilite = stable ;
     dephasage-minimum = mini ;
     temps-de-montee = 5.77 ;
     temps-de-reponse = 5.77 ;
)
```

1.2.1.1. *Gain*

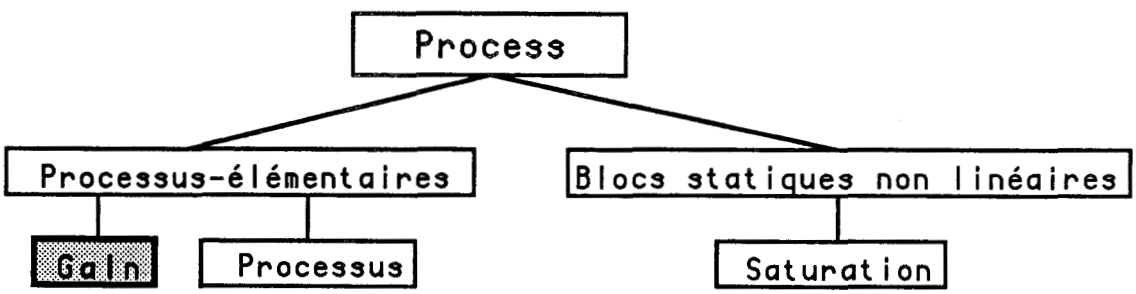


Fig. C.III.5.

Cette classe hérite de tous les attributs de sa surclasse **processus-élémentaires**. Sa particularité est l'existence d'attributs préaffectés de la manière suivante :

- **Type-modele-procede** a pour valeur **ordre-0** ;
- **Den-Ft** dont la valeur est la liste (**1.**) ;
- **Temps-de-retard** a la valeur **0.** ;
- **Nbre-integrateur** a la valeur **0** ;

- **Temps-de-montee** dont la valeur est **0.** ;
- **Stabilité** a la valeur **stable** ;
- **Déphasage-minimum** a pour valeur **mini** .

Les attributs non affectés sont : le **Gain-statique** qui doit être attribué par l'utilisateur et **Num-Ft** qui peut être déduit par le système s'il n'est pas donné par l'utilisateur. Il a pour valeur la liste dont le seul élément est la valeur du **Gain-statique.**

Exemple :

D'après les données suivantes :

```
{ g1
      est-un = gain ;
      gain-statique = 2.50 ;
}
```

Le système évalue les propriétés suivantes :

```
{ g1
      est-un = gain ;
      temps = continu ;
      type-modele-procede = ordre-0 ;
      num-ft = (2.50) ;
      den-ft = (1.00) ;
      temps-de-retard = 0.00 ;
      nbre-integrateur = 0 ;
      gain-statique = 2.50 ;
      stabilite = stable ;
      dephasage-minimum = mini ;
      temps-de-montee = 0.00 ;
      temps-de-reponse = 0.00 ;
}
```

1.2.1.2. *Processus*

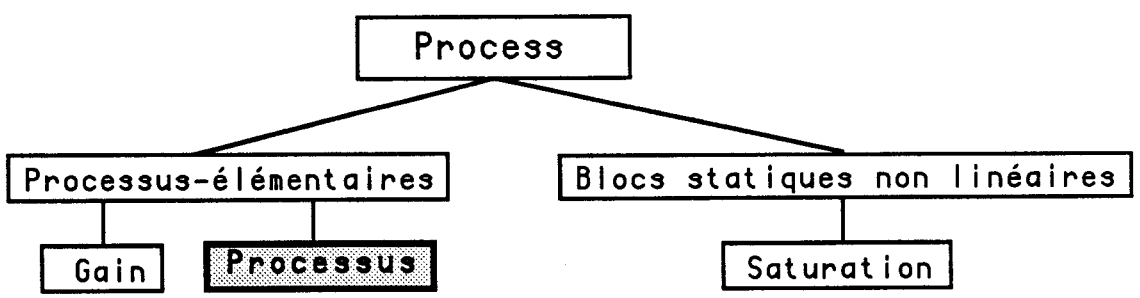


Fig. C.III.6.

Cette classe représente la famille des processus complexes définis dans

le paragraphe B.II.

Tout d'abord, de nouveaux attributs qui sont spécifiques aux processus complexes, sont définis et ne peuvent être affectés que par l'utilisateur. Ce sont :

a17) **Constituants** qui prend pour valeur la liste des **process** qui constituent le processus étudié. Dans cette liste on aura des noms d'instances de la classe **processus-élémentaires** et/ou celle de **processus** et/ou celle de **gain** ;

a18) **Structure** de type symbole, prend pour valeur la liste qui contient les différentes relations existantes entre les constituants du processus ;

a19) **Perturbations** prend comme valeur des instances de la famille **signal**. S'il n'est pas affecté par l'utilisateur il reçoit par défaut l'instance **Nulle**, c'est une instance de **constante** qui a pour amplitude **0**.

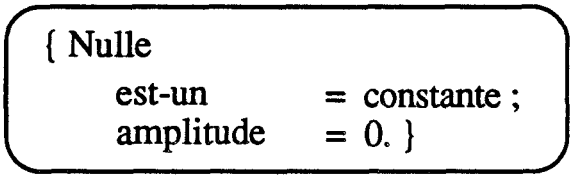


Fig. C.III.7.

Ensuite, grâce à ces nouveaux attributs (a17, a18 et a19) il est possible de définir de nouveaux attachements procéduraux au niveau des propriétés que **processus** hérite de sa surclasse **processus-élémentaires**. Ainsi en utilisant le module de calcul symbolique (décrit dans le paragraphe B.II.4.) on aura la relation qui existe entre **sortie** et **entrée-de-commande**. Par conséquent :

- si l'attribut **Num-Ft** (resp. **Den-Ft**) du **processus** n'est pas évalué le système peut inférer sa valeur. Ceci n'est possible que dans le cas où il connait **tous** les **Num-Ft** et **Den-Ft** des différents constituants du processus étudié. Pour cela, il suffit de substituer dans la relation **sortie/entrée-de-commande** les fonctions de transfert par leur valeur. Connaissant alors **Num-Ft** et **Den-Ft** le processus est aussi bien défini qu'un processus-élémentaire.

Exemple :

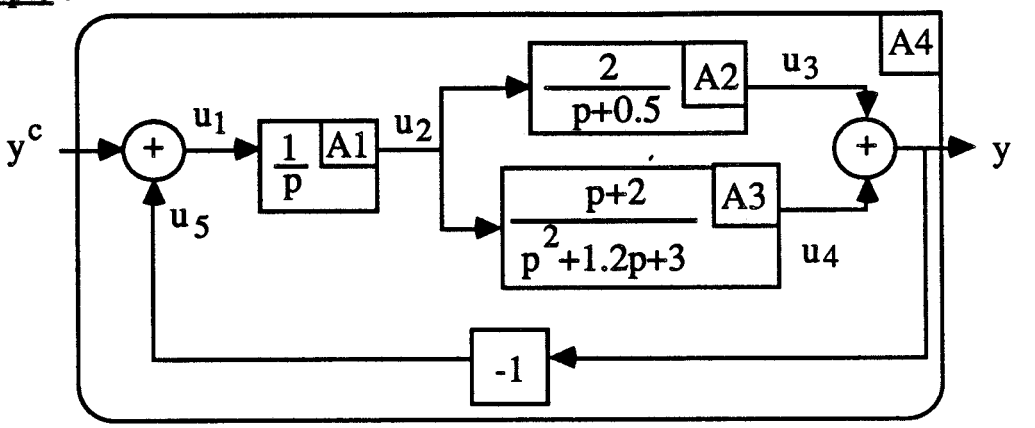


Fig. C.III.8.

Le module de calcul formel délivre :

$$\frac{Y}{Y^c} = Coef1*Coef0*A1 \quad \text{avec} \quad Coef0 = A2 + A3 \quad \text{et} \quad Coef1 = \frac{1}{1+Coef0*A1}$$

Le calcul numérique effectué sous **BASILE** donne :

$$\frac{Y}{Y^c} = \frac{3p^2+4.9p+7}{p^4+1.7p^3+6.6p^2+6.4p+7}$$

d'où **Num-Ft** = (7 4.9 3) et **Den-Ft** = (7 6.4 6.6 1.7 1).

Remarque : Dans ce qui suit on suppose que l'utilisateur ne connait pas toutes les fonctions de transferts des différents constituants ; cependant il est capable d'évaluer leurs aspects numériques macroscopiques (cf. B.II.3.1.)

- si on ne connait pas la valeur de **Temps-de-retard**, le système pourra la déduire en analysant la relation **sortie/entrée-de-commande**. Ainsi le **Temps-de-retard** de deux **process** :

  - en série est égal à la somme des deux **Temps-de-retard** ;

  - en parallèle est égal au minimum des deux **Temps-de-retard** ;

  - en feedback est égal à la valeur du **Temps-de-retard** du **process** qui se trouve en chaîne d'action.

Le **Temps-de-retard du process** A4 de l'exemple précédent sera la somme du **Temps-de-retard du process** A1 et du minimum entre celui de A2 et A3.

- pour déduire la valeur du **Nbre-integrateur** et celle du **Gain-statique**, il suffit de remplacer dans la relation **sortie/entrée-de-commande** les fonctions de transfert des différents constituants par leur pseudo-fonction de transfert. Ce que nous désignons par pseudo-fonction de transfert est une fonction de la forme :

  - $\frac{K_i}{p^{n_i}}$ dans le cas continu ;
  - $\frac{K_i}{(z-1)^{n_i}}$ dans le cas discret ;

avec K_i le **Gain-statique** et n_i le **Nbre-integrateur** du constituant i.
On justifie cette substitution par le fait qu'on ne s'intéresse dans cette partie d'étude qu'aux régimes permanents.
De ce fait en analysant la pseudo-fonction de transfert finale comme on le fait avec une fonction de transfert normale on aura la valeur du **Nbre-integrateur** et celle du **Gain-statique.**

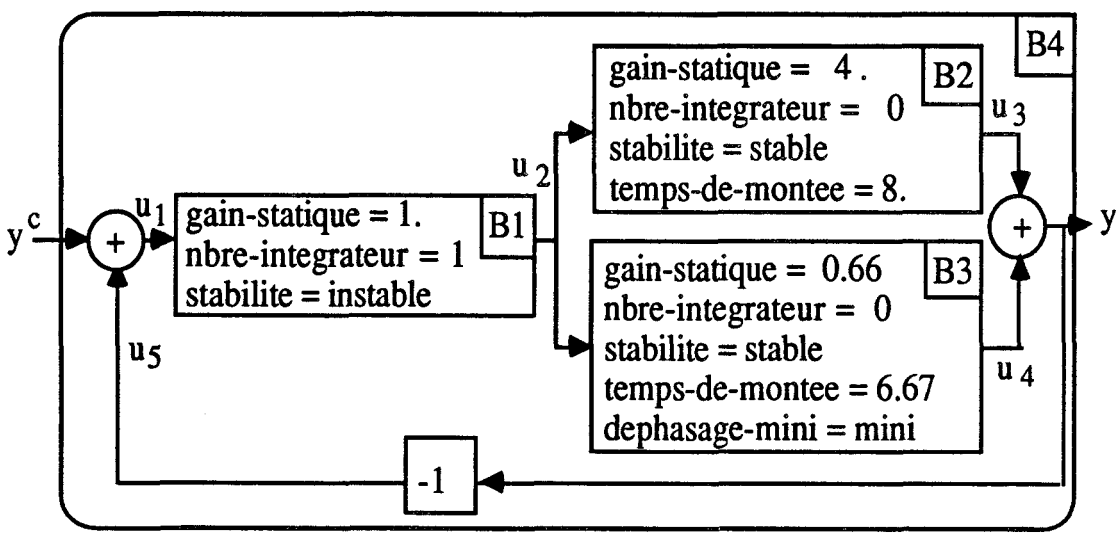


Fig. C.III.9.

- **B1** a la pseudo-fonction de transfert $\frac{1}{p}$
- **B2** a la pseudo-fonction de transfert 4

- **B3** a la pseudo-fonction de transfert $\frac{2}{3}$

avec : $\frac{Y}{Y^c} = Coef1 * Coef0 * B1$ $\quad Coef0 = \frac{14}{3}$ $\quad$ et $\quad Coef1 = \frac{1}{1 + \frac{14}{3p}}$

d'où $\frac{Y}{Y^c} = \frac{14}{3p + 14}$

- **Gain-statique = 1.** ;
- **Nbre-integrateur = 0.**

- Si on ne connaît pas la valeur de **Stabilité**, le système peut la déduire en analysant la relation **sortie/entrée-de-commande.** Ainsi la mise en série ou en parallèle de deux **process stables** est **stable**, cependant il suffit qu'un soit **instable** pour que leur combinaison le soit aussi. Dans le cas du feedback on ne peut pas conclure, sauf si on connaît toutes les fonctions de transfert des constituants.

  D'après l'exemple précédent la mise en parallèle de B2 et B3donne un processus **stable** et la mise en série de ce dernier avec B1 donne un processus **instable.**

- Pour l'attribut **Déphasage-minimum** on ne peut conclure que si on n'a que des **process** en série. La condition nécessaire et suffisante pour que le **process** résultant de cette combinaison soit à déphasage minimum est que **tous** les **process** soient à déphasage minimum (dans tous les autres cas, on doit connaître les fonctions de transfert des constituants).

- En ce qui concerne la propriété **Temps-de-montee**, sa valeur ne peut être déduite que si la relation **sortie/entrée-de-commande** n'est formée que de **process** en série ou en parallèle. Dans le cas du feedback le système ne peut pas déduire sa valeur.

  Les formules correspondantes à ce calcul sont :

  - dans le cas de n **process** en série :

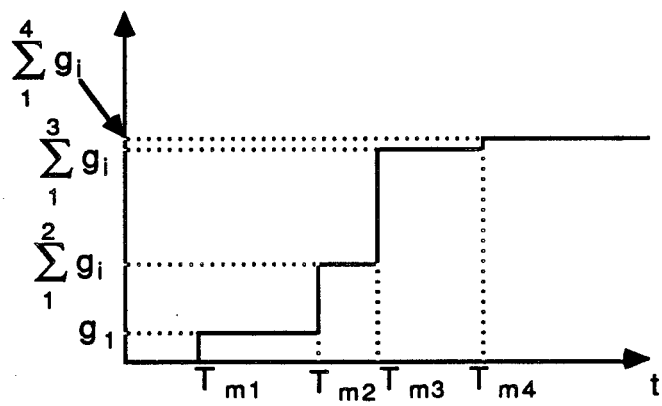
    $T_m = \sqrt{\sum_{i=1}^{n} T_{m_i}^2}$, $\quad T_{m_i}$ = temps de montée du **process** i

  - dans le cas de n **process** en parallèle, on calcule le temps de montée (à 98% de l'asymptote) comme si les sous-systèmes en

parallèle étaient des retards pur de gain g_i de retard T_{mi} (Fig. C.III.10.). Les T_{mi} étant classés dans l'ordre croissant, le **Temps-de-montée** est le T_{mi0} du plus petit i_0 tel que :

$$\sum_{i=1}^{i_0} g_i \geq 0.98 * \sum_{i=1}^{n} g_i$$

Exemple : On considère 4 process en parallèle avec :



Dans ce cas, T_m est égal au temps de montée du 3ème process car :

$$\sum_{i=1}^{3} g_i > 98\% * \sum_{i=1}^{4} g_i \Longrightarrow T_m = T_{m3}$$

Exemple :

D'après la description suivante :

```
( proc
     est-un = processus ;
     entree-commande = yc ;
     sortie = y ;
     constituants = a b c ;
     structure = (<y = u3 + u4><u3 = b * u2><u2 = a * u1><u1 = yc + u5>
                  <u5 = %-1 * y><u4 = c * u2>) ;
)
```

Le système peut inférer :

```
(proc
     est-un = processus ;
     entree-commande = yc ;
     sortie = y ;
     temps = continu ;
     num-ft = (7.00 4.90 3.00) ;
     den-ft = (7.00 6.40 6.60 1.70 1.00) ;
     temps-de-retard = 0.50 ;
     zeros-ft = (-0.82 -1.29 -0.82 1.29) ;
     poles-ft = (-0.21 -2.11 -0.21 2.11 -0.64 -1.07 -0.64 1.07) ;
     nbre-integrateur = 0 ;
     gain-statique = 1.00 ;
     stabilite = stable ;
     dephasage-minimum = mini ;
     temps-de-montee = 2.89 ;
     temps-de-reponse = 3.39 ;
     constituants = a b c ;
     structure = ((y = u3 + u4)(u3 = b * u2)(u2 = a * u1)(u1 = yc + u5)
                  (u5 = %-1 * y)(u4 = c * u2)) ;
     perturbations = nulle ;
)
```

1.2.2. La classe Blocs statiques non linéaires

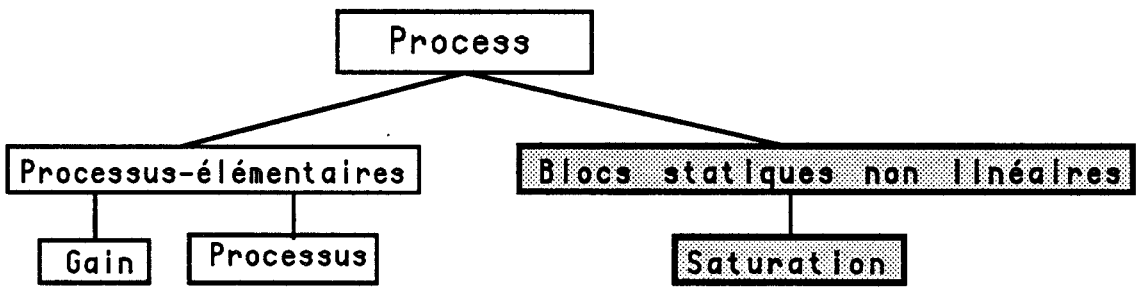


Fig. C.III.10.

Cette classe n'a pas d'attribut spécifique, elle a été créée pour donner un nom générique à cette famille. Nous lui avons défini une sous-classe **saturation** (Fig C.III.10.) qu'on représente à travers trois attributs de type réel : **bande-proportionnelle, amplitude-max** et **amplitude-min**. Ils doivent être affectés directement par l'utilisateur, toutefois la **bande-proportionnelle** prend par défaut la valeur **0**.

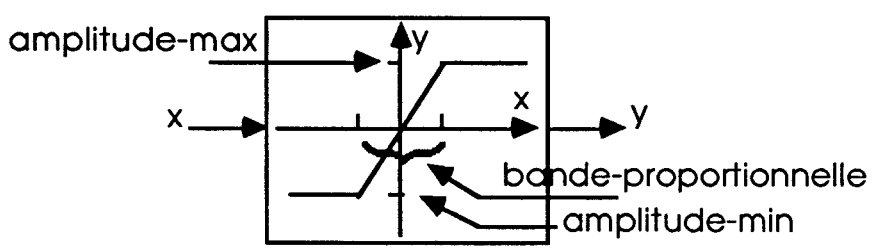


Fig C.III.11.

Nous avons décrit à travers ce module de modélisation la première moitié de notre base de connaissances. Nous consacrons le paragraphe suivant pour définir le module synthèse qui forme l'autre moitié de notre base.

2. MODULE DE SYNTHESE

En conclusion du chapitre B, nous avons présenté une synthèse d'asservissement sous forme de séquence. L'utilisation dans ce cadre, de nombreuses procédures met en valeur le choix que nous avons fait de la représentation centrée objet. Simultanément, la stabilisation qui constitue l'avant dernière étape de la synthèse fait apparaître de nombreuses heuristiques qu'on peut décrire sous forme de règles de productions en utilisant le système SPT.

Ce paragraphe est composé de quatre parties. La première décrit l'objet synthèse avec son attachement procédural pour établir la loi de commande du processus à contrôler. La seconde présente la base de règle que notre système utilise. La troisième décrit l'objet régulateur avec ses différentes sous-classes. La dernière enfin, présente un algorithme de détection de cascade.

2.1. OBJET SYNTHESE

Nous avons défini l'objet synthèse comme étant une spécialisation de la classe méthode.

L'objet synthèse est décrit, en plus de l'attribut nom-fct qu'il hérite de la classe méthode, à travers cinq attributs qui sont :

a1) **var-de-sortie**, prend pour valeur le signal qui désigne la sortie à contrôler ;

a2) **var-de-commande**, prend pour valeur le signal qui désigne l'entrée de commande du processus étudié ;

a3) **consigne**, prend pour valeur le signal qui représente la consigne à suivre (constante, rampe, etc...).

Ces trois attributs doivent être instanciés directement par l'utilisateur.

a4) **temps** de type symbole, peut prendre deux valeurs **continu** ou **discret**. S'il n'est pas instancié par l'utilisateur, il prend par

défaut la valeur **continu**. Cet attribut permet à l'utilisateur de choisir entre une commande en temps continu ou discret.

a5) **loi-commande**, prend pour valeur la liste des différents correcteurs, correspondants aux différentes boucles internes de la commande cascade, créés par le système après exécution de la procédure synthèse.

Le schéma SHIRKA de l'objet synthèse est :

```
(synthese
        sorte-de                =               methode ;
        nom-fct                 $valeur         synthese ;
        var-de-sortie           $un             signal ;
        var-de-commande         $un             signal ;
        consigne                $un             signal ;
        temps                   $un             symbole
                                $domaine        continu discret
                                $defaut         continu ;
        loi-commande            $liste-de       regulateur )
```

La fonction synthèse commence par chercher dans la liste des instances **process** qui ont été créées par l'utilisateur celle qui a pour entrée de commande la valeur de l'attribut **var-de-commande** et pour sortie la valeur de l'attribut **var-de-sortie**. S'il trouve une instance qui satisfait ces conditions alors il continue ses calculs en prenant comme processus à commander celui qu'il vient de déterminer. Sinon, il doit chercher au niveau de la liste des instances de la classe processus celle qui contient dans l'expression de sa structure les deux signaux en question. Si cette recherche aboutit à un résultat négatif, le système signale à l'utilisateur qu'il y a une erreur de formulation de son problème et lui rend la main. Par contre, dans le cas où le résultat est positif, le système créera une instance de la classe processus avec comme valeur d'attributs :

- **entree-de-commande**, la valeur de **var-de-commande** ;
- **sortie**, la valeur de **var-de-sortie** ;
- **temps**, la valeur de l'attribut **temps** du processus dont la structure a été choisie ;
- **structure**, la partie de la structure choisie où interviennent les deux signaux en question ;
- **constituants**, les différents constituants qui interviennent dans le calcul du transfert entre les deux signaux en question.

Après cette première étape le système possède donc une première description du processus dont il doit déterminer la loi de commande.

L'étape qui suit est celle de la détermination de la période d'échantillonnage s'il s'agit d'une synthèse de loi de commande en temps discret. Dans ce cas, il commence par calculer le temps de réponse du processus à contrôler et propose à l'utilisateur une période d'échantillonnage égale au 1/10 du temps de réponse. L'utilisateur peut accepter cette valeur comme il peut la refuser, si c'est le cas le système lui redonne la main pour entrer une valeur de préférence entre le 1/10 et le 1/4 du temps de réponse. Dans le cas où le système est incapable de déterminer la valeur du temps de réponse pour raison, par exemple de manque de données, l'utilisateur est obligé de donner au système une valeur de période d'échantillonnage.

La troisième phase de cette procédure consiste à déterminer la liste des signaux mesurables de mise en cascade du processus étudié. Pour cela nous utilisons un algorithme de détections de cascade que nous décrivons dans le paragraphe C.III.2.4.

Le résultat de l'application de cet algorithme sera une liste de signaux qui sont les entrées et sorties de blocs dont la mise en série nous donnera le processus étudié. Ainsi, le premier élément de cette liste est le signal d'entrée de commande du bloc le plus interne, le second est le signal de sortie du bloc le plus interne et aussi le signal d'entrée de commande du second bloc ... etc, le dernier élément est le signal de sortie du bloc le plus externe.

Exemple :

On considère le processus de la figure C.III.12.a avec comme signaux mesurables : (u e2 e6 e7 e10 e11 y). Les deux premiers tests nous donneront la liste suivante : (u e1 e2 e7 y). Comme le signal e1 ne fait pas partie de la liste des signaux mesurables, la liste définitive sera (u e2 e7 y). D'où la mise en cascade des différents blocs de la figure C.III.12.b.

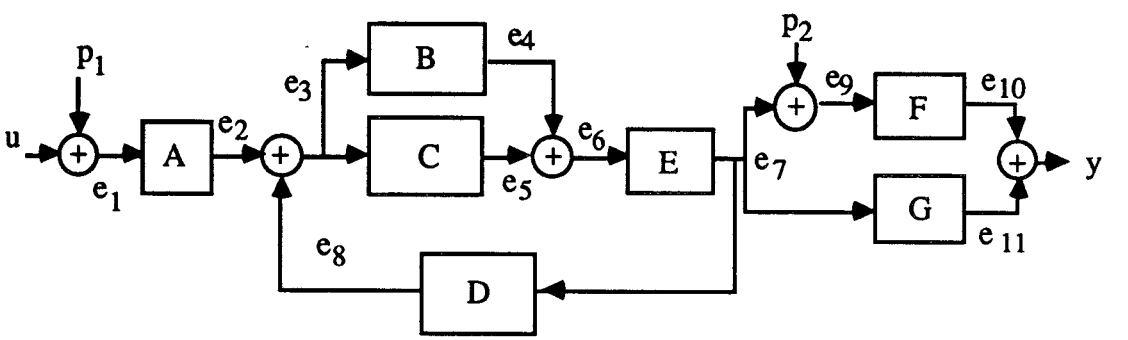


Fig. C.III.12.a

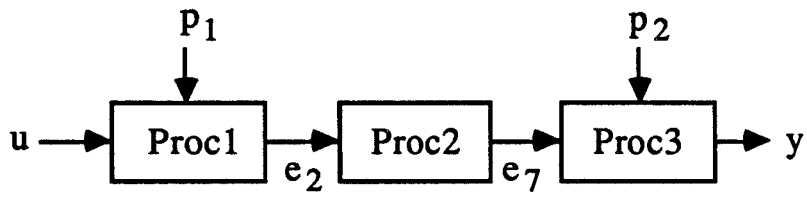


Fig. C.III.12.b

Une fois établie la liste des différents blocs de la cascade, le système doit identifier chaque bloc. Pour cela, il commence par chercher dans la liste des instances **process** qui ont été crées celle qui a pour entrée de commande l'entrée de commande du bloc et pour sortie le signal de sortie du bloc. Si cette recherche aboutit à un résultat positif, le bloc en question sera identifié au **process** trouvé. Sinon, il créera un processus dont l'entrée de commande est le signal d'entrée de commande du bloc, le signal de sortie est la sortie du bloc, la structure est la relation qui existe entre le signal de sortie et le signal d'entrée de commande du bloc et les constituants sont les différents **process** intervenant dans la structure précédente.
Pour terminer cette quatrième phase, le système doit dresser la liste des signaux de perturbations agissant sur les différents blocs.

Si l'on reprend l'exemple de la figure C.III.12. a et b on aura pour :

- Proc1 le signal de perturbation p1 ;
- Proc2 rien ;
- Proc3 le signal de perturbation p2.

Dans la cinquième étape, le système doit fixer la structure minimale du correcteur. En se référant au principe du modèle interne (cf B.III.4.1.), nous rappelons que dans un premier temps, le système doit déterminer pour chaque bloc le processus générateur des perturbations ramenées sur son entrée de commande. Pour le cas particulier du bloc le plus externe, celui dont la sortie est la même que celle du processus étudié, il faut déterminer le processus générateur des perturbations et de la consigne ramenées sur son entrée de commande. D'où, pour chaque bloc, le modèle équivalent qui est le résultat de la mise en parallèle des différents processus générateurs.
Dans un second temps, pour chaque boucle le système déduit le nombre d'intégrateurs du régulateur correspondant qui dans ce cas particulier n'est autre que le nombre d'intégrateurs du modèle équivalent.
En dernier lieu, le système doit choisir pour chaque boucle B_i une loi de commande pour la stabiliser. Ainsi, connaissant le nombre d'intégrateurs du régulateur et les propriétés même partielles du processus formant cette boucle, le système peut déduire grâce aux différentes régles que nous présentons dans le paragraphe suivant le régulateur adéquate pour la boucle

en question. Pour cela, le système lancera une session SPT en chaînage mixte avec pour but le fait **loi-commande**. SPT essayera dans un premier temps de déduire le but cherché en testant la valeur des attributs du processus en question. Dans le cas où il n'arrive pas à conclure par manque de données, il a la possibilité de dialoguer avec l'utilisateur afin d'évaluer les valeurs d'attributs ou faits manquant. Sinon, s'il arrive à atteindre son but, alors le fait **loi-commande** sera affecté du résultat trouvé. Ce résultat sera mis dans la liste-régulateur qui contiendra les différentes loi de commande correspondant aux différentes boucles de la cascade étudiée. Aussi, nous avons prévu la possibilité d'avoir plusieurs loi de commande pour un même bloc. Dans ce cas particulier, le système demande à l'utilisateur de choisir une loi parmi celles proposées. Si l'utilisateur n'arrive pas à faire son choix, alors le système prend par défaut la première possibilité.

La dernière phase de cette étude de synthèse consiste à calculer les compensations en boucle ouverte correspondant aux perturbations mesurables qui agissent sur chaque bloc de la cascade. Les différents correcteurs résultant de cette étape seront ajoutés à la liste-régulateur.
A la fin de cette étape, le système doit créer les instances des sous-classes de la famille régulateur (cf. C.III.2.3) contenues dans la liste-régulateur. Les noms de ces instances seront mis dans une nouvelle liste qui sera affectée à l'attribut **loi-commande** de l'instance de l'objet synthèse que nous venons de créer.

Nous avons décrit dans ce paragraphe l'objet synthèse et son attachement procédural. C'est une sous-classe de l'objet méthode que nous pouvons lancer en interactif sous le progiciel SHIRKA. Dans le paragraphe C.III.3. seront présentés quelques exemples de synthèse de loi de commande pour mieux illustrer les différentes étapes décrites auparavant.

Comme nous l'avons signaler dans la cinquième phase, nous présentons dans le paragraphe suivant les différentes règles que nous utilisons dans notre système.

2.2. LA BASE DE REGLES POUR LA SYNTHESE DE LOIS DE COMMANDE

La liste des règles de production définies dans notre système est la traduction en données SPT des règles (**R1** à **R14**) annoncées dans le paragraphe B.III.5.3. Il faut rappeler que cette liste peut être complétée ou modifiée suivant les besoins de l'étude.

```
(
(si nbre_integ_regulateur = 0
 et stabilite = *stable*
 alors M_A_j
 et loi-commande = *petit_gain*)
```

```
(si nbre_integ_regulateur < 2
 et nbre-integrateur = 1
 alors serie_proc_stable_integ_pur
 et test_serie_proc_stable_integ_pur)


(si nbre_integ_regulateur = 0
 et serie_proc_stable_integ_pur
 alors M_A_j
 et loi-commande = *petit_gain*)


(si nbre_integ_regulateur = 1
 et stabilite = *stable*
 alors M_A_j
 et loi-commande = *C.M.I*)


(si nbre_integ_regulateur = 1
 et stabilite = *stable*
 alors M_A_j
 et loi-commande = *P.I.Pg*)


(si nbre_integ_regulateur = 1
 et serie_proc_stable_integ_pur
 alors M_A_j
 et loi-commande = *P.I.Pg*)


(si nbre_integ_regulateur = 0
 et dephasage-minimum = *mini*
 et temps-de-retard = 0
 alors M_A_j
 et loi-commande = *grand_gain*)


(si nbre_integ_regulateur = 1
 et dephasage-minimum = *mini*
 et temps-de-retard = 0
 alors M_A_j
 et loi-commande = *P.I.Gg*)


(si nbre_integ_regulateur = 0
 et type-modele-procede = *Ziegler-Nichols*
 alors M_A_j
 et loi-commande = *P_z*)


(si nbre_integ_regulateur = 1
 et type-modele-procede = *Ziegler-Nichols*
 alors M_A_j
 et loi-commande = *P.I_z*)


(si nbre_integ_regulateur = 0
 et type-modele-procede = *integrateur_ordre-1*
 alors M_A_j
 et loi-commande = *P_critique*)


(si nbre_integ_regulateur = 1
 et type-modele-procede = *integrateur_ordre-1*
 alors M_A_j
 et loi-commande = *P.I_critique*)


(si matrices-etat = connu
 et mesurabilite-variables-etat = *mesurable*
 alors M_A_j
 et loi-commande = *retour_etat*)
```

```
(si Num-FT = connu
 et Den-FT = connu
 alors M_A_j
 et loi-commande = *placement_de_poles*)

)
```

Il faut signaler que les faits **test_serie_proc_stable_integ_pur** et **M_A_j** sont deux faits Act_ex (cf. C.I.1.2.1). Le premier permet l'appel d'une fonction Lisp qui test la possibilité de décomposer le processus étudié sous forme d'un intégrateur pur en série avec un processus stable. Ainsi il affectera le fait booléen **serie_proc_stable_integ_pur** de la valeur "**vrai**" si la décomposition est possible et de la valeur "**faux**" dans le cas contraire. Le second donne la possibilité au système de conclure sur plusieurs lois de commande si les données du problème le permettent. Exemple, dans le cas où nous avons **nbre_integ_regulateur = 1** et **stabilite = stable**, l'utilisateur pourra choisir entre C.M.I et P.I.Pg.

Les différentes conclusions des règles présentées ci-dessus sont des types de régulateurs que nous décrivons dans le paragraphe suivant sous forme de sous-classes de la classe régulateur.

2.3. L'OBJET REGULATEUR

Nous avons défini pour l'objet régulateur les deux attributs suivants : **sortie_corrigee** et **boucle_a_corriger**. Le premier correspond à la sortie de la partie du processus à corriger. Le second est la boucle à corriger de la chaîne de correction en cascade. En reprenant la figure B.III.3. nous définissons la numérotation des boucles suivantes : la B1 correspond à la boucle la plus externe, la Bi, si nous avons i blocs en cascade, correspond à la boucle la plus interne (Fig. C.III.13.).

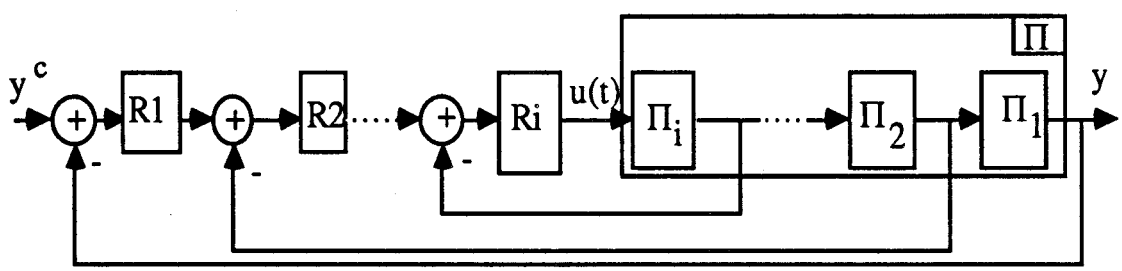


Fig. C.III.13.

La classe régulateur possède une douzaine de sous-classes (Fig. C.III.14.) qui héritent ses deux attributs. Nous n'avons défini d'autres attributs spécifiques à ces sous-classes que dans le cas particulier de **Commande_en_B_O**. Cette dernière possède en plus l'attribut **entree-de-commande** qui aura pour valeur la liste de signaux de

perturbations mesurables que le régulateur va corriger.

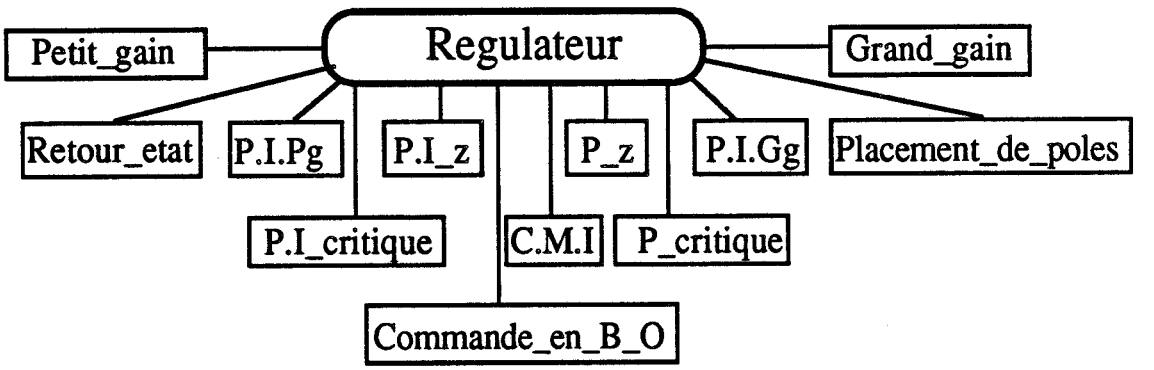


Fig. C.III.14.

Dans le paragraphe suivant nous présentons le programme qui se charge de la décomposition de processus complexe sous forme de blocs en cascade.

2.4. L'ALGORITHME DE DETECTION DE CASCADE

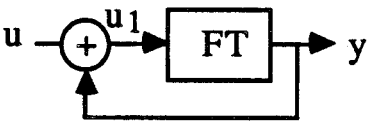
Nous avons vu, dans le paragraphe B.III.2.2., que la commande de processus monovariable peut et doit être abordée par la commande CASCADE. La première étape de réalisation de cette commande consiste à détecter l'existence de signaux mesurables en cascade entre l'entrée de commande et la sortie à contrôler. D'où, la nécessité d'un algorithme de détection de cascade que nous allons décrire dans ce paragraphe.

Avant de décrire cet algorithme, nous rappelons la définition d'un processus complexe (cf. B.II.3.2.) : il a une structure formé de liens reliant ses différents constituants. Ces liens peuvent être la mise en série, en parallèle ou en feedback de ces composants.

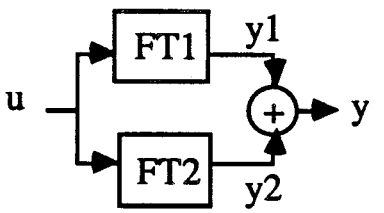
Cet algorithme est basé sur l'analyse de la structure du processus étudié. Il consiste à essayer de décomposer le processus sous forme de blocs en série. Il existe trois types de blocs à détecter :

T1) bloc élémentaire :

T2) bloc de mise en feedback :



T3) bloc de mise en parallèle :



En analysant les deux premiers blocs, nous constatons qu'en utilisant le calcul formel (cf. B.II.4.) entre le signal de sortie du bloc et son antécédent, nous pouvons savoir si il s'agit d'un bloc élémentaire ou d'un bloc de mise en feedback. Ainsi, dans le premier cas (T1) le calcul de u en fonction de y n'est pas possible. Par contre, dans le second cas (T2) le calcul de u1 en fonction de y nous donne u1 = y + u d'où l'existence d'un feedback. Dans ce cas, il faut chercher le premier antécédent de la sortie y qui ne serait pas fonction de y.

Nous détectons l'existence du troisième type de bloc (T3) en testant si la sortie y résulte d'une sommation de signaux ou pas. Si ce test est positif nous pouvons conclure sur l'existence de bloc de mise en parallèle. Dans ce cas, il faut chercher le premier antécédent commun (u) des différents signaux (y1 y2) de la somme (y).

D'après l'analyse de ces trois cas de figures nous pouvons extraire deux tests significatifs pour détecter la mise en cascade de blocs. Ces deux test sont

Test1 Est ce que l'antécédent du signal étudié est fonction du signal étudié (T2) ? Dans ce cas, il faut chercher le 1er antécédent du signal étudié qui ne satisfait pas ce test.

Test2 Est ce que le signal étudié est le résultat d'une addition de signaux (T3) ? Dans ce cas, il faut chercher le 1er antécédent commun aux différents signaux formant cette somme.

Aussi, il faut vérifier la mesurabilité de ces signaux pour avoir une commande en cascade physiquement réalisable. Ceci dit, nous ne garderons dans la liste résultant des deux premiers tests, que les signaux dont la

mesurabilité est vraie.

Nous reprenons l'exemple de la figure C.III.12. pour illustrer ces différents tests :

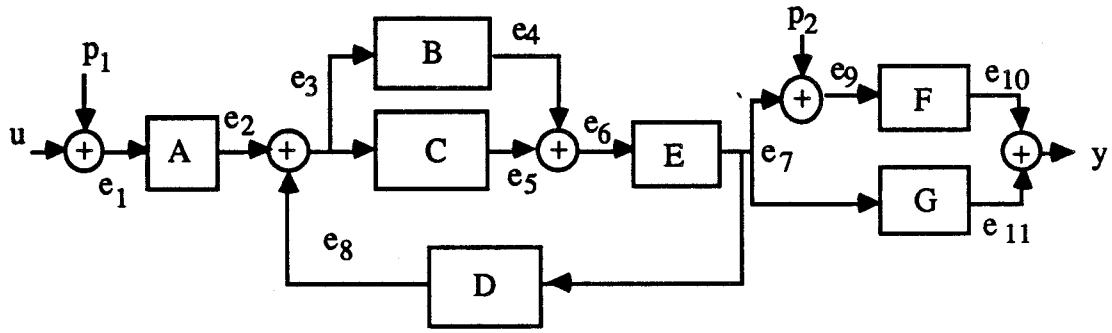


Fig. C.III.15.

L'algorithme commence par appliquer le test1 et le test2 sur le signal de sortie y :

- test1 ⟶ échec
- test2 ⟶ succès d'où la recherche du 1er antécédent commun de e_{10} et e_{11}.

Le problème est alors déplacé au niveau de e_{10} :
- test1 ⟶ échec
- test2 ⟶ échec

mais pas de relation entre e_9 et e_{11} donc e_9 n'est pas un antécédent commun à e_{10} et e_{11}.

Le problème est donc déplacé au niveau de e_9 :
- test1 ⟶ échec
- test2 ⟶ succès (pas de problème car l'un des deux signaux de la somme est un signal de perturbation)

mais il y a une relation entre e_7 et e_{11} donc e_7 est un antécédent commun de e_{10} et e_{11} d'où la possibilité de bloc entre y et e_7.

Le programme continue les tests avec comme signal de départ e_7 :
- test1 ⟶ succès, d'où la recherche du 1er antécédent de e_7 qui n'est pas fonction de e_7.
- test2 ⟶ échec

Le problème est alors déplacé au niveau de e_6 :

- test1 ⟶ succès, d'où la recherche du 1er antécédent de e_6 qui n'est pas fonction de e_7.
- test2 ⟶ succès, d'où la recherche du 1er antécédent commun de e_4 et e_5.

Le problème est alors déplacé au niveau de e_4 :

- test1 ⟶ succès, d'où la recherche du 1er antécédent de e_4 qui n'est pas fonction de e_7.
- test2 ⟶ échec

mais, il y a une relation entre e_3 et e_5, donc e_3 est un antécédent commun de e_4 et e_5.

Le problème est alors déplacé au niveau de e_3 :

- test1 ⟶ succès au niveau de e_8
  échec au niveau de e_2.
- test2 ⟶ succès

dans ce cas particulier, il faut choisir e_2 qui satisfait le test1, d'où la possibilité de bloc entre e_7 et e_2.

Le programme continue les tests avec comme signal de départ e_2 :

- test1 ⟶ échec
- test2 ⟶ échec

donc une possibilité de bloc entre e_2 et e_1.

Le problème est alors déplacé au niveau de e_1 :

- test1 ⟶ échec
- test2 ⟶ succès (pas de problème car l'un des deux signaux de la somme est un signal de perturbation)

d'où la possibilité de bloc entre e_1 et u.

A ce niveau, nous pouvons satisfaire le test d'arrêt de cette première phase qui est l'égalité entre le signal à tester et le signal d'entrée. Le résultat de cette phase est dans ce cas la liste (u e_1 e_2 e_7 y).

Nous obtenons la liste définitive en appliquant le test de mesurabilité sur cette dernière liste, d'où le résultat suivant (u e_2 e_7 y) :

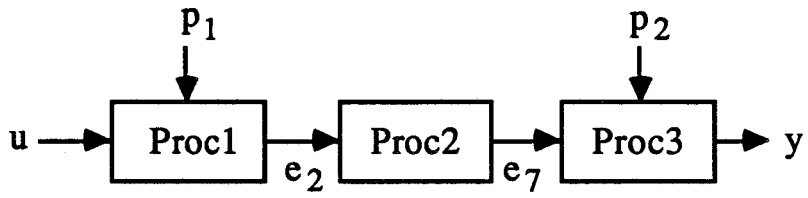


Fig. C.III.16.

Nous pouvons conclure ce paragraphe, en insistant sur l'utilité de ce programme dans le cadre de la synthèse de loi de commande pour des processus monovaribles. Grâce à lui, nous pouvons détecter les différents blocs en cascade du processus à réguler, et ainsi, appliquer l'algorithme de la commande cascade.

3. TRANSFERT DES DONNEES DANS LA BASE DE CONNAISSANCES

Notre base de connaissances est composée de plusieurs modules qui utilisent des données symboliques, numériques ou les deux à la fois (Fig. C.III.17.)

Dans ce paragraphe nous allons décrire, la façon avec laquelle nous avons procéder aux problèmes de transferts de données entre ces différents blocs.

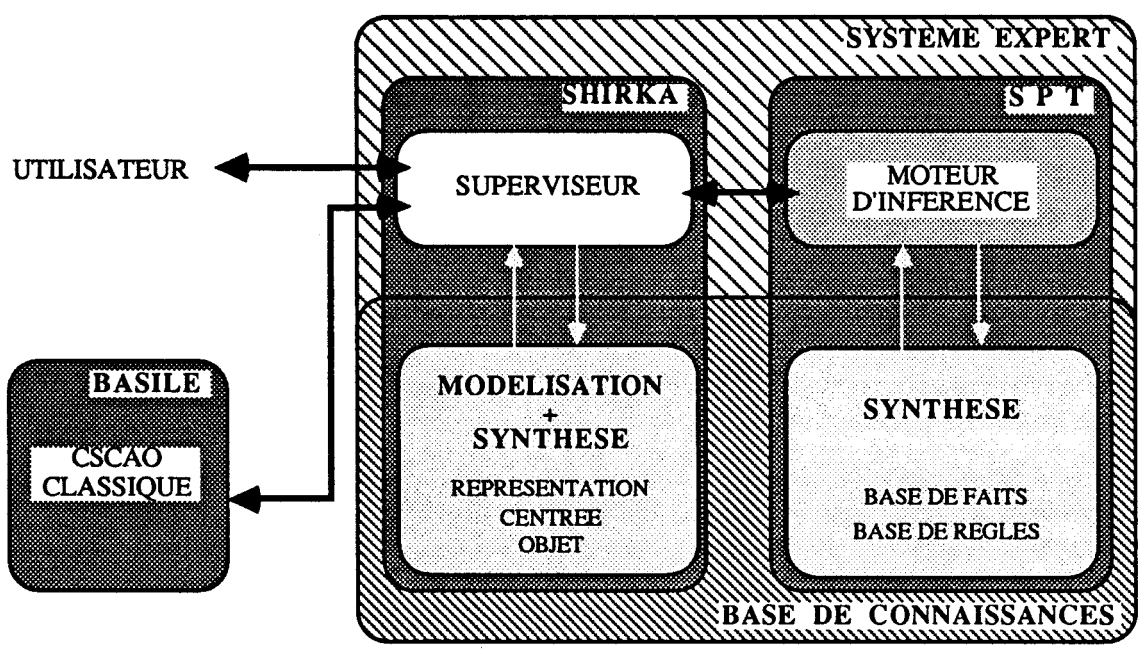


Fig. C.III.17.

3.1. TRANSFERT DES DONNEES SHIRKA ⇔ SPT

Les données que nous transférons entre SHIRKA et SPT sont les valeurs d'attributs (objets SHIRKA) que l'on doit unifier avec des faits (SPT) de même nom (attribut → fait ou fait → attribut).

SHIRKA et SPT n'utilisent pas les mêmes structures de données. Cependant, le fait qu'ils soient écrits tous les deux en Le_Lisp et que leurs concepteurs aient conçu des fonctions spéciales (pour qu'ils affectent et/ou modifient la valeur de leurs variables), nous a aidé à résoudre ce problème de transferts de données.

Un fait (SPT) a été défini comme étant un atome Lisp. Pour affecter sa valeur à un attribut, les concepteurs de SHIRKA ont créé des fonctions qui modifient ou ajoutent une valeur (du même type que l'attribut) à un attribut donné. De même, ils en ont défini d'autres pour extraire la valeur d'un attribut, qu'on peut affecter à un fait de même nom (voir Annexe 5).

3.2. TRANSFERT DES DONNEES LE_LISP ⇔ BASILE

Nous transférons des données entre deux langages différents : Le_lisp et Fortran (BASILE).

La solution adoptée pour remédier au problème de communication des données entre les deux langages, causé par la différence de leurs structure de données, est l'utilisation du système de boîte aux lettres au travers du système d'exploitation VAX/VMS [BRIAN 89].

Nous allons expliquer brièvement le principe de ce système. Dans notre cas, Le_lisp joue le rôle du chef d'orchestre. Ainsi, il commence par sauvegarder les données qu'il veut transférer dans un **fichier.don**. Ensuite, il lance une session Basile grâce à un fichier commande (système VAX/VMS). Alors, Basile grâce à un fichier commande (Basile) extrait les données du **fichier.don**, fait les opérations de calcul nécessaires et sauvegarde le résultat dans un **fichier.res**. En quittant Basile, Le_lisp reprend la main et extrait à son tour les résultats du **fichier.res** pour les affecter au(x) variable(s) correspondante(s) [CHAILLOUX 86] [DELEBECQUE 87].

Nous avons décrit ci-dessus notre base de connaissances à travers les deux modules de modélisation et de synthèse. Aussi, nous avons présenté les protocoles de transferts des données entre les différents blocs formant notre base. Nous allons illustrer ces différentes phases à travers les exemples du paragraphe suivant.

4. EXEMPLES ET MODE D'EMPLOI

4.1. EXEMPLE D'UN ASSERVISSEMENT DE POSITION

Considérons le processus décrit par le schéma suivant :

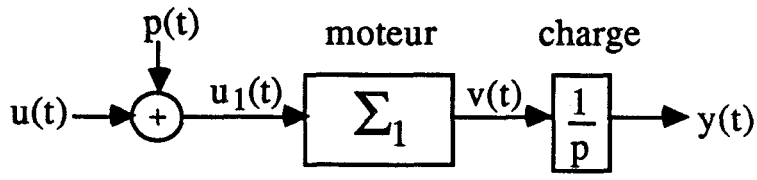


Fig. C.III.18.

p est une perturbation constante non mesurable (volt)
u est l'entrée de commande (volt)
v est la vitesse, c'est la sortie du moteur (rad/s)
y est la position, c'est la sortie de la charge (rad)

Σ_1 est un système de type premier ordre dont on connaît le gain statique k, k = 10 rad/s/v et la constante de temps τ = 100 ms.
On peut par exemple imaginer que Σ_1 est modélisée exactement par la fonction de transfert suivante :

$$\frac{k \qquad (1+\frac{\tau}{30}p)}{(1+\tau p)(1+\frac{\tau}{20}p)(1+\frac{\tau}{40}p)}$$

Le but de la session de synthèse suivante est de trouver la loi de commande en temps continu pour réguler la sortie y du processus présenté ci-dessus autour de la consigne yc.

Nous prenons comme hypothèses, yc est une constante et y est un signal mesurable. Aussi, dans un premier cas nous supposons que v est un signal mesurable et, dans un second cas que la mesurabilité de v est fausse.

1er cas : v mesurable

En premier lieu, l'utilisateur doit décrire comme suit les différents signaux et processus formant son système :

```
( y
      est-un = signal ;
      mesurabilite = vrai ;
      unite-de-mesure = rad ;
)
```

```
( yc
     est-un = constante ;
     mesurabilite = vrai ;
     unite-de-mesure = rad ;
)
( u
     est-un = signal ;
     mesurabilite = vrai ;
     unite-de-mesure = volt ;
)
( u1
     est-un = signal ;
     unite-de-mesure = volt ;
)
( v
     est-un = signal ;
     mesurabilite = vrai ;
     unite-de-mesure = rad/s ;
)
( p
     est-un = constante ;
     unite-de-mesure = volt ;
)
( a
     est-un = processus-elementaire ;
     entree-commande = u1 ;
     sortie = v ;
     type-modele-procede = ordre-1 ;
     gain-statique = 10.00 ;
     stabilite = stable ;
     temps-de-reponse = 0.40 ;
)
( b
     est-un = processus-elementaire ;
     entree-commande = v ;
     sortie = y ;
     num-ft = (1.00) ;
     den-ft = (0.00 1.00) ;
)
( proc
     est-un = processus ;
     entree-commande = u ;
     sortie = y ;
     constituants = a  b ;
     structure = ((y = b * v) (v = a * u1) (u1 = p + u)) ;
     perturbations = p ;
)
```

En second lieu, il doit lancer une session de synthèse comme suit :

```
Shirka: exec
methode? synthese
var-de-sortie? y
var-de-commande? u
consigne? yc
temps? continu
loi-commande? -
```

Au cours de ce calcul, le système détectera la présence d'une cascade selon la décomposition suivante :

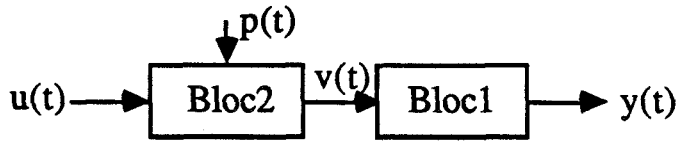


Fig. C.III.19.

Pour corriger Bloc2, il commence par déduire le nombre d'intégrateurs du correcteur R2 qui est dans ce cas égal à 1. Ensuite, et grâce aux règles de la liste C.1., il proposera une liste de lois de commande possible composée de deux éléments : une structure C.M.I (Commande par Modèle Interne) et un réseau correcteur P.I.Pg (Proportionnelle Intégrale avec Petit gain). L'utilisateur aura à choisir entre ces deux lois.

Liste C.1. :

```
(si nbre_integ_regulateur = 1
 et stabilite = *stable*
 alors M_A_j
 et loi-commande = *C.M.I*)

(si nbre_integ_regulateur = 1
 et stabilite = *stable*
 alors M_A_j
 et loi-commande = *P.I.Pg*)
```

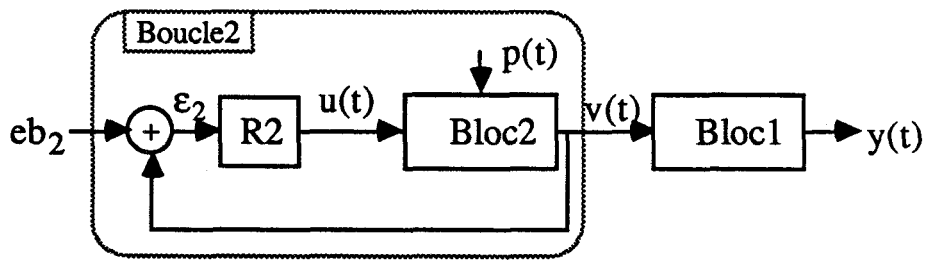


Fig. C.III.20.

L'étape suivante consiste à corriger le processus résultant de la mise en série de la boucle interne (Boucle2) avec Bloc1 (Fig. C.III.20.). Dans ce cas, nous pouvons utiliser un régulateur dont le nombre d'intégrateurs est nul. Ainsi, et grâce à aux règles de la liste C.2., le système peut conclure sur une loi de commande de type retour proportionnel à petit gain.

Liste C.2. :

```
(si nbre_integ_regulateur < 2
 et nbre-integrateur = 1
 alors serie_proc_stable_integ_pur
 et test_serie_proc_stable_integ_pur)

(si nbre_integ_regulateur = 0
 et serie_proc_stable_integ_pur
 alors M_A_j
 et loi-commande = *petit_gain*)
```

Le résultat de cette session de synthèse sera la création de l'instance synthèse suivante :

```
( $synthese-118
      est-un = synthese ;
      var-de-sortie = y ;
      var-de-commande = u ;
      consigne = yc ;
      loi-commande =  (loi_com_b2-v loi_com_b1-y) ;
)
```

Avec les loi_com_b2-v et loi_com_b1-y décrites comme suit :

```
( loi_com_b2-v
      est-un = p.i.pg ;
      boucle_a_corriger = b2 ;
      sortie_corrigee = v ;
)

( loi_com_b1-y
      est-un = petit_gain ;
      boucle_a_corriger = b1 ;
      sortie_corrigee = y ;
)
```

2ème cas : v non mesurable

Dans ce cas nous n'aurons pas de cascade car v n'est pas mesurable. Nous disposons donc d'un processus dont le type du modèle de procédé est un intégrateur_ordre-1, c'est-à-dire un processus de type premier ordre en série avec un intégrateur pur. Le régulateur doit avoir un nombre d'intégrateurs égal à 1 car p est un signal constant. Le système conclura grâce à la règle de la liste C.3. sur une loi de commande égale à P.I.critique (la Fonction de Transfert du système corrigé par la loi de commande Proportionnelle Intégrale, aura trois pôles confondus).

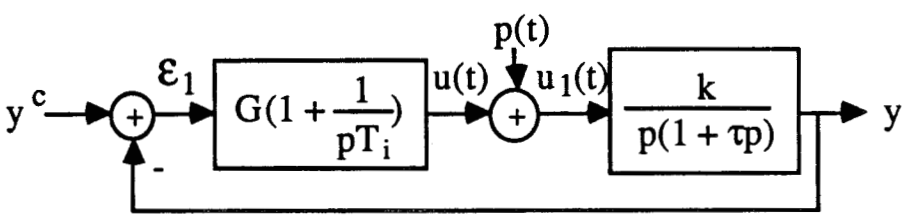


Fig. C.III.21.

Liste C.3. :

```
(si nbre_integ_regulateur = 1
 et type-modele-procede = *integrateur_ordre-1*
 alors M_A_j
 et loi-commande = *P.I_critique*)
```

L'instance de la méthode synthèse sera dans ce cas :

```
( #synthese-118
      est-un = synthese ;
      var-de-sortie = y ;
      var-de-commande = u ;
      consigne = yc ;
      loi-commande =  (loi_com_b1-y) ;
)
```

L'instance régulateur loi_com_b1-y est décrite comme suit :

```
( loi_com_b1-y
      est-un = p.i_critique ;
      boucle_a_corriger = b1 ;
      sortie_corrigee = y ;
)
```

4.2. EXEMPLE D'UN ASSERVISSEMENT DE NIVEAU ET DE TEMPERATURE

Etudions le dispositif physique suivant :

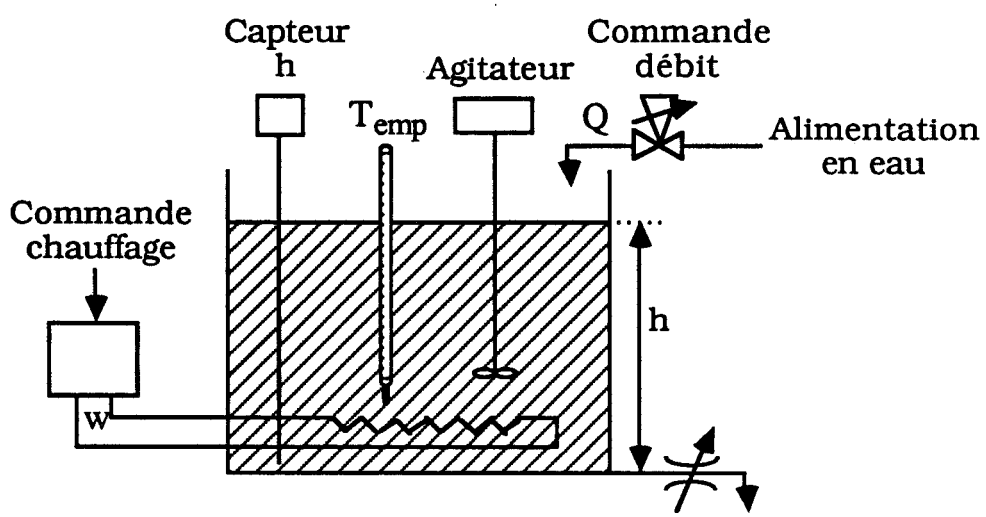


Fig. C.III.22.

Nous présentons dans la figure C.III.23. un modèle simplifié du processus ci-dessus.

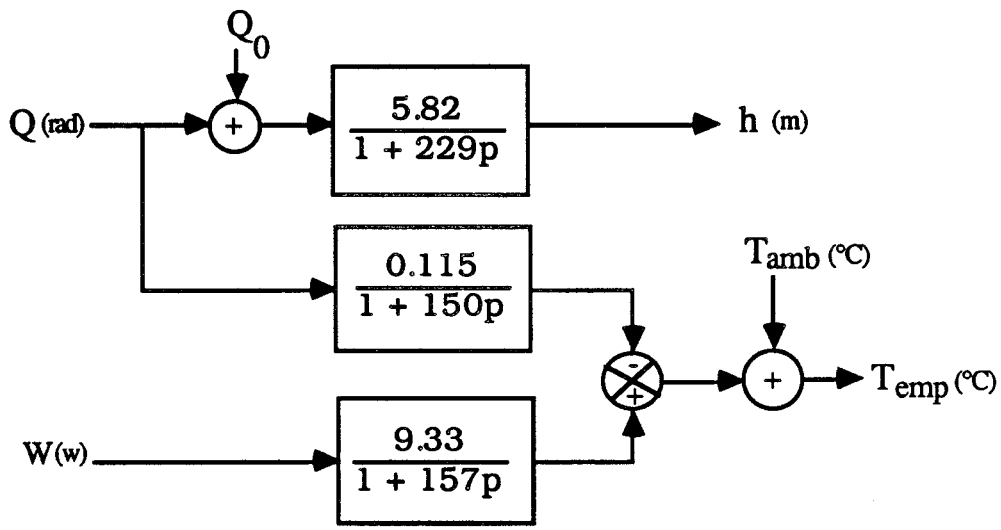


Fig. C.III.23.

Q_0 est inconnu et constant, c'est le déréglage de la vanne ;

Tamb est mesurable et constante.

Nous souhaitons réguler autour de valeur constante d'abord le niveau h, puis la température Temp. Nous allons faire cette étude en deux temps : continu et discret.

Notre système est conçu pour étudier des processus monovariables. Donc à priori, il est incapable de faire une étude de synthèse du processus décrit ci-dessus. Cependant, dans le cas où l'utilisateur (affirmé) est capable de décomposer son processus en plusieurs processus monovariables interconnectés, le système proposera des lois de commandes pour corriger les différents processus monovariables.

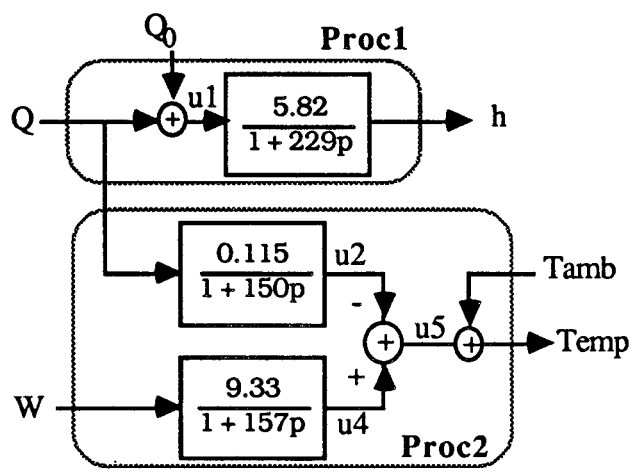


Fig. C.III.24.

Dans le cas particulier du processus décrit ci-dessus, l'utilisateur peut le décomposer en Proc1 et Proc2 (Fig. C.III.24.). Ces deux processus sont monovariables et indépendants, la seule connexion entre eux est l'action de Q en tant qu'un des signaux de perturbations de Proc2.

Durant l'étape de modélisation, l'utilisateur doit décrire comme suit les signaux et processus composant le processus étudié :

```
{ h
     est-un = signal ;
     mesurabilite = vrai ;
     unite-de-mesure = metre ;
}
{ hc
     est-un = constante ;
     mesurabilite = vrai ;
     unite-de-mesure = metre ;
}
{ q
     est-un = signal ;
     mesurabilite = vrai ;
     unite-de-mesure = rad ;
}
{ q0
     est-un = constante ;
     unite-de-mesure = rad ;
}
{ u1
     est-un = constante ;
     unite-de-mesure = rad ;
}
{ a1
     est-un = processus-elementaire ;
     entree-commande = u1 ;
     sortie = h ;
     num-ft = (5.82) ;
     den-ft = (1.00 229.00) ;
}
{ proc1
     est-un = processus ;
     entree-commande = q ;
     sortie = h ;
     constituants = a1 ;
     structure = (<h = a1 * u1> <u1 = q0 + q>) ;
     perturbations = q0 ;
}
{ temp
     est-un = signal ;
     mesurabilite = vrai ;
     unite-de-mesure = degre-C ;
}
{ tc
     est-un = constante ;
     unite-de-mesure = degre-C ;
}
{ w
     est-un = signal ;
     mesurabilite = vrai ;
     unite-de-mesure = watt ;
}
```

```
{ tamb
     est-un = constante ;
     mesurabilite = vrai ;
     unite-de-mesure = degre-C ;
}
{ u2
     est-un = signal ;
     unite-de-mesure = degre-C ;
}
{ u3
     est-un = signal ;
     unite-de-mesure = degre-C ;
}
{ u4
     est-un = signal ;
     unite-de-mesure = degre-C ;
}
{ u5
     est-un = signal ;
     unite-de-mesure = degre-C ;
}
{ b1
     est-un = processus-elementaire ;
     entree-commande = q ;
     sortie = u2 ;
     num-ft = (0.12) ;
     den-ft = (1.00 150.00) ;
}
{ c1
     est-un = processus-elementaire ;
     entree-commande = w ;
     sortie = u4 ;
     num-ft = (9.33) ;
     den-ft = (1.00 157.00) ;
}
{ proc2
     est-un = processus ;
     entree-commande = w ;
     sortie = temp ;
     constituants = b1  c1 ;
     structure = (<temp = tamb + u5> <u5 = u4 + u3> <u4 = c1 * w>
                 <u3 = %-1 * u2> <u2 = b1 * q>) ;
     perturbations = tamb  q ;
}
```

1er cas : Synthèse en temps continu

L'étape de synthèse est décomposée dans ce cas particulier en deux phases. La première consiste à corriger Proc1 pour avoir l'erreur (hc - h) nulle, la deuxième doit déduire la loi de commande qui fait tendre Temp vers Tc sans erreur.

Phase 1 :

```
Shirka: exec
methode? synthese
var-de-sortie? h
var-de-commande? q
consigne? hc
temps? continu
loi-commande? -
```

Dans ce cas nous n'avons pas de cascade. Nous corrigeons un processus stable avec un régulateur dont le nombre d'intégrateurs est égal à 1 car Q_0 est une constante. Dans ces conditions, l'utilisateur aura à choisir entre les lois de commande C.M.I et P.I.Pg sur lesquelles notre système conclura grâce aux règles de la liste C.1.

L'instance synthèse correspondant à cette première phase est :

```
( %synthese-440
     est-un = synthese ;
     var-de-sortie = h ;
     var-de-commande = q ;
     consigne = hc ;
     loi-commande = (loi_com_b1-h) ;
)
```

La loi de commande loi_com_b1-h est décrite comme suit :

```
( loi_com_b1-h
     est-un = c.m.i ;
     boucle_a_corriger = b1 ;
     sortie_corrigee = h ;
)
```

Phase 2 :

```
Shirka: exec
methode? synthese
var-de-sortie? temp
var-de-commande? w
consigne? tc
temps? continu
loi-commande? -
```

Nous nous retrouvons dans les mêmes conditions que celles de la correction du Proc1. Toutefois, Proc2 est perturbé par les signaux mesurables Q et Tamb, donc nous pouvons minimiser leurs effets sur Temp en utilisant une loi de commande en boucle ouverte.

L'instance synthèse correspondant à cette seconde phase est :

```
( %synthese-456
     est-un = synthese ;
     var-de-sortie = temp ;
     var-de-commande = w ;
     consigne = tc ;
     loi-commande = (loi_com_b1-temp loi_com_temp) ;
)
```

Les deux lois de commande loi_com_b1-temp et loi_com_temp sont :

```
( loi_com_b1-temp
     est-un = c.m.i ;
     boucle_a_corriger = b1 ;
     sortie_corrigee = temp ;
)
```

```
( loi_com_temp
     est-un = commande_en_b_o ;
     boucle_a_corriger = b1 ;
     sortie_corrigee = temp ;
     entree-de-commande = tamb q ;
)
```

2ème cas : Synthèse en temps discret

En général, une étude de synthèse en temps discret nécessite une phase de plus par rapport à celle en temps continu. Cette étape est celle du choix du pas d'échantillonnage utilisé durant la session de synthèse. Pour faire ce choix, le système commence par calculer le temps de réponse du processus étudié, s'il y arrive il proposera à l'utilisateur comme valeur de la période d'échantillonnage le 1/10 du temps de réponse. Dans ce cas, l'utilisateur a la possibilité d'accepter comme de refuser, si c'est le cas il doit proposer au système la valeur qu'il juge valable. Dans le cas où le système ne peut pas déduire la valeur du temps de réponse par manque de données, l'utilisateur est obligé de proposer une période d'échantillonnage au système.

Nous décomposons notre étude comme dans le cas de la synthèse en temps continu en deux phases :

Phase 1 :

```
Shirka: exec
methode? synthese
var-de-sortie? h
var-de-commande? q
consigne? hc
temps? discret
loi-commande? -
```

Dans cet exemple, le système proposera à l'utilisateur la valeur suivante Te = 91.60 s qui correspond au 1/10 du temps de réponse de Proc1.

La seule différence entre l'instance synthèse en temps discret et celle en temps continu est la valeur de l'attribut temps de la méthode synthèse :

```
( %synthese-450
     est-un = synthese ;
     var-de-sortie = h ;
     var-de-commande = q ;
     consigne = hc ;
     temps = discret ;
     loi-commande =  (loi_com_b1-h) ;
)
```

Nous aurons la même description que dans le cas de temps continu de la loi commande loi_com_b1-h :

```
( loi_com_b1-h
     est-un = c.m.i ;
     boucle_a_corriger = b1 ;
     sortie_corrigee = h ;
)
```

Phase 2 :

```
Shirka: exec
methode? synthese
var-de-sortie? temp
var-de-commande? w
consigne? tc
temps? discret
loi-commande? -
```

Dans cette seconde phase, le système proposera à l'utilisateur la valeur suivante Te = 60 s qui correspond au 1/10 du temps de réponse de Proc2.

L'instance synthèse correspondante à cette phase est la suivante :

```
( %synthese-466
     est-un = synthese ;
     var-de-sortie = temp ;
     var-de-commande = w ;
     consigne = tc ;
     temps = discret ;
     loi-commande =  (loi_com_b1-temp loi_com_temp) ;
)
```

Nous aurons la même description des lois de commande loi_com_b1-temp et loi_com_temp que dans le cas de temps continu :

```
( loi_com_b1-temp
     est-un = c.m.i ;
     boucle_a_corriger = b1 ;
     sortie_corrigee = temp ;
)
( loi_com_temp
     est-un = commande_en_b_o ;
     boucle_a_corriger = b1 ;
     sortie_corrigee = temp ;
     entree-de-commande = tamb q ;
)
```

Notre travail se limite dans l'état actuel du système à proposer des lois de commande sans calculer leurs différents cœfficients. Ceci dit, l'utilisateur ne remarquera pas de différence entre deux sessions de synthèse, une en temps discret et l'autre en temps continu.

D'autre part, nous avons prévu la possibilité de calculer la fonction de transfert discrète d'un processus continu. Ceci dit, une fois implantés les différents programmes de calculs des cœfficients des lois de commande, le système aura à sa disposition tous les outils nécessaires pour proposer une solution numérique dans le cas continu comme dans le cas discret.

Nous présentons à la fin de cette étude une possibilité de discrétisation du modèle du processus étudié. En prenant le plus petit temps de réponse des trois processus élémentaires qui le compose, nous échantillonnerons avec une période de 60 s qui représente le 1/10 de ce temps de réponse. Nous aurons les fonctions de transfert discrétisées suivantes :

$$A1(p) = \frac{5.82}{1 + 229p} \quad \text{----->} \quad A1(z) = \frac{1.34}{-0.77 + z}$$

$$B1(p) = \frac{0.115}{1 + 150p} \quad \text{----->} \quad B1(z) = \frac{0.04}{-0.67 + z}$$

$$C1(p) = \frac{9.33}{1 + 157p} \quad \text{----->} \quad C1(z) = \frac{2.96}{-0.68 + z}$$

IV CONCLUSION

Nous pouvons conclure après cette présentation des différents outils informatiques formant notre systèmes que l'utilisation des objets structurés pour décrire les processus étudiés et des règles de productions pour la description du raisonnement est un bon choix.

D'autre part, notre maquette a l'état actuel peut proposer à l'utilisateur une ou plusieurs lois de commande suivant la finesse de description du modèle étudié. Aussi, elle est capable de lui calculer les valeurs des différents attributs de l'objet processus si elle dispose de tous les données nécessaires pour ces calculs.

CONCLUSION GENERALE

Le travail présenté dans ce mémoire concerne l'élaboration d'un superviseur de progiciel de C S C A O. L'utilisateur est guidé dans la description de son problème de contrôle de processus et des solutions bien adaptées lui sont suggérées.

Le système adapte la technicité des solutions proposées par rapport à la finesse de représentation du processus étudié. Il est ainsi en mesure de proposer des solutions relativement satisfaisantes pour un utilisateur non-spécialiste et tenir compte des exigences particulières des utilisateurs experts.

Le prototype que nous présentons envisage la classe des systèmes analysables en termes monovariables avec une sortie contrôlée et une entrée de commande. Le fonctionnement est considéré dans le cadre linéaire et déterministe en commande continue ou échantillonnée.

L'essentiel du travail repose sur l'utilisation de la représentation centrée objet pour décrire les processus étudiés.

La souplesse de ce mode de représentation nous a permis ensuite de mener la synthèse grâce à des connaissances à la fois procédurales et déclaratives.

La pertinence de la méthode proposée repose sur le parallèle qui existe entre le degré de finesse du modèle de procédé et les performances qu'on peut espérer obtenir du système contrôlé.

L'utilisation de systèmes de déduction, fondés sur les règles de production, et de la modélisation par objets des systèmes étudiés sont bien adaptées au problème de C S C A O. Le système expert à base de règles de production permet d'utiliser des méthodes de synthèse qui n'ont pas nécessairement de lien logique entre elles.

Pour permettre des sessions de synthèse fréquentielle, il sera possible de rajouter, sans modifier la description des systèmes dynamiques, les caractéristiques fréquentielles comme les fréquences de résonance et de coupure. Nous pouvons intégrer les méthodes correspondantes sous forme de règles de production dans notre base de règles en respectant la cohérence avec les autres règles.

REFERENCES

BIBLIOGRAPHIQUES

REFERENCES BIBLIOGRAPHIQUES

* ASTRÖM K.J. (1984 a)
"CADCS : Computer Aided Design of Control Systems".
Proceeding of the sixth INRIA international conference on analysis and optimization of systems.
Nice, june 19-22, - Part 2, pp. 549-563 edited by A. Bensoussan and J.L. Lions.

* ASTRÖM K.J., HAGANDER P. and STERNBY J. (1984 b)
"Zeros of Sampled Systems".
Automatica, vol 20, n° 1, pp. 31-38.

* ASTRÖM K.J. and WITTENMARK (1984 c)
"Computer Controlled Systems : Theory and design".
Pretence Hall International Editions.

* BARRAUD A., GENTIL S. (1989)
"La C.A.O de l'Automatique".
Hermès, Paris.

* BEL G. (1990)
"Outils d'Intelligence Artificielle pour l'Intégration"
Colloque International : Productique et Intégration, Bordeaux 12-14 Juin 1990.

* BRIAN P., BUTZ and N.F. PALUMBO (1989)
"Developing an expert system to assist in control system design".
Journal of Intelligent and robotic system 2, pp. 277-295.

* CHAILLOUX J. (1986)
"Le_lisp, Manuel de référence"
I.N.R.I.A.

* COMYN G. (1987)
"Représentation des connaissances : problèmes posés par la représentation mixtes".
Résumé de l'exposé fait à l'E.P.F.L. Lausanne (29 / 10 / 1987).

* DELAHAYE J.P. (1987)
"SYSTEMES EXPERTS : organisation et programmation des bases de connaissances en calcul propositionnel".
Edition Eyrolles.

* DELARMINAT P. et THOMAS Y. (1975/1977)
"Automatique des systèmes linéaires".
1. Signaux et systèmes (1975)
3. Commande (1977)
Flammarion sciences.

* DELARMINAT (1989)
"COMMANDE MONOVARIABLE : Modèles polynomiaux et placement de pôle".
Extrait du cours d'Automatique de l'INSA.

* DELEBECQUE F., KLIMANN C., STEER S. (1987)
"BASILE : Guide d'utilisation".
Version 2 (27-5-87).

* DELEBECQUE F., STEER S. (1985)
"The interactive system BLAISE for control engineering".
3rd IFAC/IFIP INTERNATIONAL SYMPOSIUM.
Computer aided design in control engineering systems (CADCE 85).
Editors : Hansen N.E. and Martin Larsen P.
Copenhagen (31/7 au 2/8) - pp. 44-46.

* DUFOUR J., GREHANT B., LOTTIN J. (1986)
"ROBUSTNESS : Internal Model Control Approach Extension to State Affine Representation".
IMACS Villeneuve d'Ascq.

* DUBOIS A. (1987)
"SPT : Système Propositionnel Type".
Rapport DEA Labo Info. USTLFA

* ELMQVIST H. (1977)
"SIMNON - An interactive simulation program for non linear systems".
Proceeding Simulation 77, Montreux, pp. 85-89.

* FARRENY H., GHALLAB M. (1987)
"Elements d'Intelligence Artificielle".
Traité des Nouvelles Technologies, Série Intelligence Artificielle.
Edition HERMES. pp. 71-119.

* FARRENY H. (1985)
"Les systèmes experts, principes et exemples".
Cepadues-editions. - pp. 211-216, 233-237.

* FAVIER G. (1987)
"Quelques réflexions sur une B.C. pour un logiciel de CAO en Automatique".
Action CAO-Automatique Thème 3.
PARIS, (29 / 01 / 87).

* FENET E. et MEIZEL D. (1986)
"Commande adaptative : aspects théoriques et pratiques".
pp. 451-456, Ed. Masson.

* FOULARD C., GENTIL S. et SANDRAZ J.P. (1987)
"Commande et régulation par calculateur numérique".
Editions Eyrolles.

* FRANCIS B. A. and WONHAM W. M. (1976)
"The Internal Model Principle of Control Theory".
Automatica, vol 12, pp. 457-465.

* GARCIA C.E. and MORARI M. (1982)
"Internal Model Control. 1. A Unifying Review and Some New Results".
Ing. Eng. Chem. Process Des. Dev. 1982, 21, 308-323.

* GARCIA C.E. and MORARI M. (1985)
"Internal Model Control. 2. Design Procedure for multivariable systems".
Ing. Eng. Chem. Process Des. Dev. 1985, 24, 472-484.

* GENTIL S., BARRAUD A. Y. (1987 a)
"La CAO pour l'enseignement de l'automatique".
R.A.I.R.O. APII Vol. 21, n° 5, pp. 401-418

* GENTIL S., RECHENMANN F. (1987 b)
"Identification des procédés et Intelligence Artificielle".
Action CAO-Automatique Thème 3
PARIS (29 / 01 / 87)

* GENTIL S. (1988 a)
"Intelligence artificielle appliquée à l'automatique".
R. 7215 (1 - 10)
Techniques de l'ingénieur, Traité Mesures et Contrôle.

* GENTIL S. (1988 b)
"Systèmes experts pour la C A O en automatique"
AFCET/INTERFACES n° 73-74 (Novembre/Décembre 1988)

* LAPORTE P., GENTIL S., BARRAUD A.Y. (1984)
"Un logiciel d'identification assistée par ordinateur"
Colloque S.E.E. Automatique Appliquée, Nice, France.

* LAURENT P. (1987)
"Systèmes Experts et Langages Orientés Objets : un mariage fructueux"
MICRO-SYSTEMES (Juillet, Aout 1987), pp. 145-150.

* MACIEJOWSKI J.M. and MAC FARLANE A. G. J. (1982)
"CLADP : The Cambridge Linear Analysis and Design Programs".
I.E.E.E., Control Systems Magazine, Décembre 1982, pp. 3-8.

* MOLER C. (1982)
"MATLAB"
Université de New Mexico, Rapport technique C581-1.

* MOLER C. et LITTLE J. (1986)
PC-MATLAB (manuel d'utilisation) version 2.2

* PIERRET C. (1988)
"Vers un système à base de connaissances centrées-objets pour la modélisation de systèmes dynamiques en biologie".
Thèse de doctorat (22/11/1988). U T C

* POWER H. M. (1976)
"The Mad Matrician strikes again"
Electronics & Power, Avril 1976, pp. 229-233

* RECHENMANN F. (1984 a)
"Intelligence artificielle et construction de modèles dynamiques".
I.A. et productique, Paris.

* RECHENMANN F., BENSAID A., GRANIER D.(1984 b)
"SHIRKA : des systèmes experts centrés objets".
2-4 mai, Avignon 1984, pp. 1-18.

* RECHENMANN F. (1987 a)
"Manuel de SHIRKA".

* RECHENMANN F. (1987 b)
"SAFIR-SHIRKA : Un système à base de connaissances centrée-objet pour l'analyse financière".
13-15 mai, Avignon 1987, pp. 949-967.

* RECHENMANN F. (1988)
"Systèmes à base de connaissances et systèmes experts : principes et architecture"
Journées Systèmes à Base de Connaissances pour la C A O en Automatique (12-13/01/1988). INRIA, Paris.

* RIMVALL M. (1988)
"C A C S D - Software Aujourd'hui et Demain".
Journées Systèmes à Base de Connaissances pour la C A O en Automatique (12-13/01/1988). INRIA, Paris.

* RIMVALL M. (1987)
"ELCS : The Extended List of Control Software".
Number 4, December 1987, Swiss Edition.

* ROSENBROCK H.H. (1974)
"CACSD : Computer Aided Control System Design".
Control Systems Centre, University of Manchester, Institute of Science and Technology, Manchester, England, Academic Press.

* SEVELY Y. (1973)
"Systèmes et asservissements linéaires échantillonnés".
Edition Dunod Université.

* TAYLOR J.H. and FREDERICK D.K., (1984)
"An Expert System Architecture For Computer-Aided Control Engineering".
Proceeding on the IEEE, Vol. 72, No. 12,
Décembre 1984. - pp. 1795-1805.

* TAYLOR J.H., JAMES J.R. and FREDERICK D.K., (1985)
"An expert system architecture for coping with complexity in computer-aided control engineering".
3rd IFAC/IFAP International Symposium.
Computer aided design in control engeneering systems (CADCE 85).
Editors : Hansen N.E. and Martin Larsen P.
Copenhagen (31/7 au 2/8) - pp. 47-52.

* ZAFIRIOU E., MORARI M. (1985)
"Digital controllers for SISO systems: a review and a new algorithm".
Int. J. Control, vol 42, n° 4, pp. 855-876.

* RECUEIL DES LOGICIELS (1988)
Forum AFCET Automatique, Club EEA Automatique
21-22/04/1988, IUT Le Montet, NANCY

* SADT : La recherche N° 136 (SEPTEMBRE 1982).

* TUTSIM : Université de TWENTE, Pays-Bas.

ANNEXES

Annexe 1

LISTE DES SYSTEMES EXISTANTS DE C S C A O

On a retenu cinq caractéristiques des systèmes élaborés depuis la fin des années 70 dans de nombreux laboratoires d'automatique :

- le nom du programme (parfois plusieurs noms si le système est constitué de plusieurs outils) ;

- son origine (université ou entreprise, pays) ;

- sa portée (identification et / ou commande, la simulation étant toujours réalisée, au moins pour les modèles considérés) ;

- le mode d'interaction avec l'utilisateur (langage de commande ou question / réponse) ;

- le langage de programmation utilisé ;

complétées par une référence bibliographique (quand c'est possible).

EUROPE

NOM	ORIGINE	POR TEE		MODE D'INTERACTION	LANG. DE PROGRAM.	REFERENCE(S)
		Id.	Com			
KEDDC	Ruhr Allemagne	oui	oui	langage de commande + Q/R pour sous-tâches	FORTRAN	. SCHIMD et UNBEHAUEN, 79
OLID-CADCA-Siso/Mimo	Darmstadt Allemagne	oui	oui	Q/R	FORTRAN	. HABER et BAMBERGER, 75 . DYMSCHIZ, 79 . DYMSCHIZ et SCHUMAN, 79
CATPAC	Hambourg Allemagne	oui	oui	Q/R + langage de commande	FORTRAN	. BARTHELMES et Al, 82 . BUENZ et GUESTSCHOW, 85
CAIAD CSNCS	Bradford Angleterre	oui	oui	?	FORTRAN	
CLADP	Cambridge Angleterre	non	oui	Q/R + mots de commande	FORTRAN	. MACIEJOWSKI et MAC FARLANE, 82

Annexe 1

SANCAD	Salford Angleterre	non	oui	Q/R	FORTRAN	. GRAY et BROWN, 84
VASSYD	Sheffield Angleterre	non	oui	Q/R	FORTRAN	. DORLING et ZINOBER, 84
CONCEN--TRIC SYNIC	Manchester Angleterre	non	oui	Q/R	FORTRAN	. MUNRO, 79 . IEEE Control System, 82
CONTROL FREQFIT	Southampton Angleterre	non	oui	Q/R	BASIC	
PROCAL CUPID et LEVEL CUPID	Southampton Angleterre	non	oui	Q/R	BASIC + FORTRAN	
CAIAD	Wolver--hampton Angleterre	non	oui	?	BASIC + FORTRAN	
The Hull Control System Design Suite	Hull Angleterre	non	oui	Q/R	FORTRAN	. TAYLOR, 84 . TAYLOR et Al, 85
SUNS	Sussex Angleterre	non	oui	Q/R	FORTRAN	
UMIST Suites	Manchester Angleterre	non	oui	Q/R	FORTRAN	. MUNRO, 79
SUBOPT	Galles du Nord Angleterre	non	oui	Q/R	FORTRAN	. FLEMING, 79/81
	Ghent Belgique	oui	oui	?	?	. DE KEYSER et Al, 79
PAAS et PAASM	Louvain Belgique	non	oui	Q/R	FORTRAN	. PAILLET

	Madrid Espagne	non	oui	?	?	. GONZALEZ, 82
IP-DEM	Bordeaux France	oui	non	?	FORTRAN	. CARAP et Al, 84
SIRENA	Rennes - Nantes France	oui	oui en cours	Q/R + mots de commande	FORTRAN	. YEM et Al, 84 . LAPORTE et Al, 84
ACSYDE	Grenoble France	oui	oui	Q/R + mots de commande	FORTRAN	
I.S.E.R.-C.S.D.	Paris France	oui	oui	Q/R	FORTRAN et PASCAL	
POLLUX CASTOR CALLIAN	Meylan France	non	oui	Q/R	FORTRAN et GKS standard	
ARTAN-Pc	Grenoble France	non	oui	Q/R	?	
BASILE	INRIA/ SIMULOG Paris France	oui	oui	Langage de commande	FORTRAN	. DELEBECQUE et Al, 85/87
BLACK	Nantes France	non	oui	Langage de commande	Pascal	
EVANS SIMNUM	Strasbourg France	non	oui	Q/R	FORTRAN	
IDMO	Poitiers France	oui	oui	?	BASIC	
PC-REG Expert-AD	ADAPTECH France	non	oui	Q/R	?	
PIM	ADAPTECH France	oui	non	Q/R	?	

Annexe 1

MERCE--DES ICON--GRAPH	Budapest Hongrie	oui	oui	?	FORTRAN	. FECHER et Al, 79
ADPAC	Gènes Italie	oui	oui	Q/R	FORTRAN	. CASALINO et Al, 85/87
CYPROS	Trondheim Norvège	oui	oui	Q/R + langage de commande	FORTRAN	. TYSSO, 79 . TYSSO, 80
SATER	Eindhoven Pays-Bas	oui	non	Q/R	FORTRAN	. VAN DEN BOOM, 84
TRIP	Delft Pays-Bas	oui	non	Langage de commande	FORTRAN	. VAN DEN BOSCH, 85
CASAD ADCON	Bucarest Roumanie	oui	oui	Langage de commande	FORTRAN	. DAVIDOVICIU et VARGA, 84 . SIMA et POPESCU, 82 . VARGA et SIMA, 84 . VARGA et DAVIDOVICIU, 85
IDPACK	Bucarest Roumanie	oui	non	Q/R	FORTRAN	
IDPAC (identifica--tion) SYNPAC (synthèse)	Lund Suède	oui	oui	Langage de commande (INTRAC)	FORTRAN	. WIESLANDER, 76 . WIESLANDER et ELMQUIST, 78 . WIESLANDER, 79 . ASTRÖM, 83
ANSYDE	Stockholm Suède	non	oui	Langage de commande	SIMULA (ALGOL 60)	
HORPAC	Lund Suède	non	oui	Langage de commande	FORTRAN	. GUTMAN et NEUMANN, 85
IMPACT	Zurich Suisse	non	oui	Langage de commande	ADA	. RIMVALL et CELLIER, 84
INTPOS	Zurich Suisse	non	oui	?	FORTRAN	

L.A.S.	Belgrade Yougoslavie et Virginie USA	non	oui	Langage de commande	FORTRAN	. BINGULAC et Al, 85

AMERIQUE

AESOP	NASA Cleveland USA	non	oui	Q/R	FORTRAN	
CAMCSD	New-York USA	non	oui	Q/R Menu + crayon lumineux	FORTRAN	. FREDERICK et Al, 82 . PUTZ et WOZNY, 84
CTRL-C	Palo Alto USA	oui	oui	Langage de commande	FORTRAN	. LITTLE et Al, 84
DELIGHT-MIMO	Berkeley USA Londres Angleterre	non	oui	Langage de commande	RATTLE + FORTRAN	. POLAK et Al, 82 . POLAK et MAYNE, 84 . NYE, 82
DIGICON/ APL	Stanford USA	non	oui	Langage de commande	APL	. EMAMI-NAEINI et FRANKLIN, 81
EASYS	Turkwila USA	non	oui	Langage de commande	FORTRAN	
AUTOCON	Torrance USA	non	oui	Q/R	FORTRAN	. LEFKOWITZ, 83 . LEFKOWITZ
FREDOM/ TIMDOM LSSPACK	Nouveau Mexique USA	non	oui	Q/R	BASIC étendu (sur Pc)	. JAMSHIDI et MALEK-ZAVAREI, 85 . JAMSHIDI, 83
MATRIX-X	Palo-Alto USA	oui	oui	Langage de commande	FORTRAN	. WALKER et Al, 82
LSAP	Livermore USA	non	oui	Q/R	PASCAL + FORTRAN	. HERGET, 85

Annexe 1

HONEY-X DIGIKON	Honeywell USA	non	oui	Langage de commande	FORTRAN + C	. DOYLE et Al, 82
SSDP	General Electric USA	non	oui	?	?	
TOTAL	Ohio USA	non	oui	?	?	
TSPSP	Virginie USA	non	oui	Q/R	FORTRAN	
ACTS	Los-Angeles Cambridge USA	non	oui	Q/R + menus	FORTRAN PL/1	. STRUNCE et LAUB, 82
	Tolédo Ohio USA	non	oui	Q/R	FORTRAN	. LEININGER,79
	Hewlett Packard USA	non	oui	Type terminaux HP	BASIC étendu	
SSPACKtm	Livermore USA	oui	non	Q/R + langage de commande	FORTRAN	
CC	Hawthorne USA	non	oui	Langage de commande	BASIC	. IEEE ou CACSD, 85
ESTIMATE	Studio city USA	non	oui	Langage de commande	FORTRAN	
CAEBEL	Texas USA	non	oui	Q/R	FORTRAN	
LINCON STATE-VARCON MVDFCON/ MVCON	New Brunswick Canada	non	oui	Q/R + mots de commande	APL	

Annexe 1

WATERLO Control Design Suite	Waterloo Canada	non	oui	Langage de commande	FORTRAN + RATFOR	

AUSTRALIE

CAPTAIN	Cambrerra Australie	oui	non		FORTRAN	. YOUNG et JAKEMAN 79

JAPON

DPACS-F	Tokyo	oui	oui	Q/R + Langage de commande	FORTRAN	. FUTURA et AL, 79a et 79b . FUTURA et AL, 80 . FUTURA et AL, 82
DSDP	Kawasaki	non	oui	?	FORTRAN	

D'aprés [LAPORTE 85][RIMVAL 87][RECUEIL 88].

REFERENCES BIBLIOGRAPHIQUES DE L'ANNEXE 1

* ASTRÖM K.J. (1983)
"Computer Aided Modeling, Analysis and Design of Control Systems - A perspective".
Control Systems Magazine, vol. 3, n° 2, May 1983, pp. 4-16.

* BARTHELMES M. et Al. (1982)
"CAPTAC : An interactive software package for process analysis and control".
3rd IFAC/IFIP Symposium on Software for Computer Control, Madrid, Spain, pp. 149-155.

* BINGULAC S.P., WEST P.J., PERKINS W.R. (1985)
"Recent Advances in the L-A-S Software Used in CAD of Control Systems".
Proc. 1985 IFAC Symposium on CAD, Copenhagen.

* BUENZ D., GUETSCHOW K. (1985)
CATPAC : An interactive software package for control system design".
Automatica, May 1985, in press.

* CARAP L., DEPAYE G., DUROU G., JOUVE P. (1984)
"I.P. - D.E.M. A Statistical Dynamic Identification Package by Single Output Difference Equation Model to Industrial Processes on Micro-computers".
Preprints 9th IFAC World Congress, Budapest, Hungary, pp. 196-201.

* CASALINO G., DAVOLI F., MINCIARDI R. (1985)
"Finite and infinite horizon LQ adaptive controllers based on implicit model identification".
Proc. 24th I.E.E.E. Conf. on Decision and Control, Fort Lauderdale, FL, pp. 853-855.

* CASALINO G., DAVOLI F., MINCIARDI R., ZAPPA G. (1987)
"On implicit modelling theory : basic concepts and applications to adaptive control".
To appear in Automatica, March 1987.

* DAVIDOVICIU A., VARGA A. (1984).
"CASAD : An Interactive Package For Computer Aided System Analysis and Design".
Proc. 6th Int. Conf. on Analysis and Optimization of Systems, Nice, France, pp. 221-235.

* DE KEYSER R.M.C., D'HULSTER F.M., HEYSE J.G., VAN CAUWENGERGHE A.R. (1979).
"Experiments with a software package for process identification and control".
Preprints IFAC Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 473-477.

* DORLING C.M. and ZINOBER A.S.I. (1984)
"Computer-Aided Design of Variable Structure Control Systems".
In Proc. Inst. Meas. and Cont. Workshop on CACSD, Brighton, U.K., pp. 67-72.

* DOYLE J. C., KONAR A.F., MAHESH J.K., PRATT S.G., WALL J.E. (1982)
"DIGIKON and HONEY-X : Interactive Packages for Control System Analysis and Design".
Workshop on CACSD, University of California, Berkeley.

* DYMSCHIZ E. (1979)
"A process computer program package for interactive computer aided design of multivariable control systems".
Proc. 2nd IFAC/IFIP Symposium on Software for Computer Control Systems, Prague, Czechoslovakia, p. C-IV-1.

* DYMSCHIZ E., SCHUMANN R. (1979)
"A program package for the computer aided design of feedforward control algorithms with process computers".
Proc. IFAC Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 241-246.

* EMAMI-NAEINI A. and FRANKLIN G.F. (1981)
"Interactive Computer-Aided Design of Control Systems".
I.E.E.E. Control Systems Magazine, 1, pp. 31-36.

* FECHER A., KEVISKY L. et Al. (1979)
"MERCEDES : Interactive Software Package for Identification and Experimental Control of Industrial Plants by a Portable Process Computer Laboratory".
Preprints SOCOCO 79, 2nd IFAC/IFIP Symposium on Software for Computer Control, vol. II, Prague, Czechoslovakia.

* FLEMING, P.J. (1979)
"A C.A.D. program for suboptimal linear regulators".
Preprints IFAC/IFIP Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 259-266.

* FLEMING, P.J. (1981)
"A CAD Program for Suboptimal Regulators-Design Applications".
Proceedings of the IASTED Symposium on Modelling, Identification, and Control, Davos, Switzerland, pp. 136-140.

* FREDEDERIK D.K., KRAFT R.P, SADEGHI T. (1982)
"Computer Aided Control System Analysis and Design Using Interactive Graphics".
I.E.E.E. Control Systems Magazine, vol. 2, pp. 19-23, Dec. 1982.

* FUTURA K., KAJIWARA H., TSURVOKA K. (1980)
"Computer Aided Design Program for linear multivariable control system".
Preprints IFAC Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 267-272.

* FUTURA K., HATAKEYAMA S., KOMINAMI H. (1979)
"Structural Identification and Software package for linear multivariable systems".
5th IFAC Symposium on Identification and System Parameter Identification, Darmstadt, Germany, pp. 415-422.

* FUTURA K. et Al. (1982)
"Computer System Design for Furnace by CAD".
IFAC Symposium, Delhi, pp. 31-38.

* GONZALES de SANTOS P. et Al. (1982)
"A software package for C.A.D. of multivariable Control Systems".
3rd IFAC/IFIP Symposium on Software for Computer Control, Madrid, Spain, pp. 189-194.

* GRAY J.O. and BROWN L.S. (1984)
"A Computer-Aided Design Suite for the Analysis and Design of Nonlinear Systems".
Trans. Inst. Meas. and Control, Vol. 6, n° 1, Jan-Mar 1984.

* GUTMAN P.O., NEUMANN L. (1985)
"HORPAC and interactive program for robust control system design".
In Proc. 2nd I.E.E.E. Control System Society Symposium on Computer Aided Control System Design, Santa Barbara, CA, March 13-15.

* HABER R., BAMBERGER W. (1975)
"OLID-MISO-NOLI : Interactive Software Package for Identification and Extremum Control of Non Linear Dynamic Processes by Means of a Process Computer".
Preprints of 3rd IFAC/IFIP Symposium on Software for Computer Control, vol. II, Prague, Czechoslovakia.

* HERGET C.J., TILLY D.M. (1985)
"Linear Systems Analysis Program".
In Jamshidi M. and Herget C.J., (Eds.), "Computer-Aided Control System Engineering", North-Holland, Amsterdam.

* JAMSHIDI M. (1983)
"Large-Scale Systems Modelling and Control".
Elseviers, North-Holland, New-York.

* JAMSHIDI M. and MALRK-ZAVAREI M. (1985)
"Linear Control System-A System-Aided Approach".
Pergamon Press, Ltd., Oxford, England.

* LAPORTE P. (1985)
"Conception assistée par ordinateur en automatique, un logiciel d'identification".
Thèse de doctorat (10 / 07 / 1985). Grenoble.

* LEININGER G.G. (1979)
"An interactive design suite for the multivariable Nyquist array method".
IFAC Symposium on CADCS, Zurich, Switzerland, pp. 325-330.

* LEFKOWITZ C.P. (1983)
"Automated Synthesis of Control Systems : A Design Approach".
Proc. of IFAC Workshop on Applications of Nonlinear Programming to Optimization and Control.

* LITTLE J.N., EMAMINAEINI A., BANGERT S.N. (1984)
"CTRL-C and Matrix environments for the Computer-Aided Design of Control systems".
Proc. 6th Int. Conf. Analysis and Opitimization of Systems, Nice, France, pp. 191-205.

* MUNRO N. (1979)
"The UMIST Control System design and Synthesis suites".
Preprints IFAC Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 343-348.

* NYE B., TITS A. (1982)
"Delight for Beginners".
Int. Report Electronic Research Laboratory, Berkeley, U.S.A.

* PAILLET D.
"PAAS : Programme d'aide à l'analyse des systèmes".
pp. 231-233.

* POLAK E., SIEGEL P., WUU T., NYE W.T., MAYNE D.Q. (1982)
"DELIGNT-MIMO : An interactive Optimization Based Multivariable Control System Design Package".
I.E.E.E. Control Systems Magazine, vol. 2, pp. 9-14, Déc. 1982.

* POLAK E., MAYNE D.Q. (1984)
"Theoretical and Software Aspects of Optimization Based Control System Design".
Proc. 6th Int. Conf. on Analysis and Optimization of Systems, Nice, France, pp. 175-190.

* PUTZ P., WOZNY M.J. (1984)
"Parameter Space Design of Control systems using Interactive Computer Graphics".
Proc. American Control Conference, San Diego, California, U.S.A., pp. 105-114.

* RIMVALL M., CELLIER P. (1984)
"IMPACT : Interactive Mathematical Program for Automatic Control Theory".
Proc. 6th International Conference on Analysis and Optimization of Systems, Nice, France, pp. 578-597.

* SCHMID C., UNBEHAUEN H. (1979)
"KEDDC : a general purpose CAD software system for application in control engineering".
Preprints 2nd IFAC/IFIP Symposium on Software for Computer Control, Prague, Czechoslovakia, vol. 2, section C-V.

* SIMA V., POPESCU T.D. (1982)
"ADCON : Adaptative Control Package".
3rd IFAC/IFIP Symposium on Software for Computer Control, Madrid, Spain, pp. 195-201.

* STRUNCE R.R., LAUB A.J. (1982)
"Advanced Control Technology Software (ACTS)".
Workshop on C.A.C.S.D., University of California, Berkeley.

* TAYLOR P.M. (1984)
"The Hull Control System Design Suite".
In Proc. Inst. Meas. and Cont. Workshop on Computer Aided Control Systems Design, Brighton, U.K., pp. 73-78.

* TAYLOR P.M., ZHAO Y., JOBLING C.P., (1985)
"A Computer Aided Design Suite for Nonlinear Systems of a General Structure".
IEEE Conf. Control'85, Cambridge, U.K., pp. 474-479.

* TYSSO A. (1979)
"CYPROS : Cybernetic Program Packages".
Preprints IFAC Symposium on CAD of Control Systems, Zurich, Switzerland, pp. 383-389.

* VAN DEN BOOM A.J.W. and BOLLEN R. (1984)
"The idendification package SATER".
Preprints 9th IFAC World Congress, Budapest, vol. VIII, pp. 190-195.

* VAN DEN BOSCH P.P.J. (1985)
"Interactive Computer-Aided Control System Analysis and Design".
In M. Jamshidi and C.J. Herget (eds.), Advances in Computer-Aided Control Systems Engineering, North-Holland Publishing Company, pp. 229-242.

* VARGA A. and SIMA V. (1984)
"BIMAS-A Basic Mathematical Package for Computer Aided Systems Analysis and Design".
Proceedings of 9th IFAC World Congress, Budapest, Pergamon Press.

* VARGA A. and DAVIDOVICIU A. (1985)
"BISMAC-A Package of FORTRAN Subprograms for Analysis, Modelling, Design, and Simulation of Control Systems".
Preprints of 3rd IFAC Symposium on CAD in Control and Engineering Systems, Copenhagen, July 31-August 2, Pergamon Press, 1985.

* WALKER R., GREGORY C., SHAH S. (1982)
"$MATRIX_X$: A Data Analysis, System Identification, Control Design and Simulation Package".
I.E.E.E. Control Systems Magazine, vol. 2, pp. 30-37, Déc. 1982.

* WIESLANDER J. (1976)
"IDPAC User's guide".
Int. Reprot 7605, April 1976, Department of Automatic Control, Lund Institute of Technology.

* WIESLANDER J. (1979)
"Design principales for computer aided design software".
Preprints IFAC Symposium on Computer Aided Design of Control Systems, Zurich, Switzerland, pp. 493-496.

* WIESLANDER J., ELMQUIST R. (1978)
"INTRAC - A communication module for interactive program. Language Manual".
Int. Report, Department of Automatic control, Lund Institute of Technology.

* YEM Y., CHOUMLIVONG K., BARRAUD A. (1984)
"SIRENA : un outil de C.A.O. pour l'Automatique".
Proc. 6th Int. Conf. on Analysis and Opitimization of Systems", Nice, France, pp. 206-220.

* YOUNG P., JAKEMAN A., (1975)
"The development of CAPTAIN : a Computer Aided Program for Time-series Analysis and the Identification of noisy Systems".
Prep. IFAC Symposium on CAD of Control Systems, Zurich, Switzerland, pp. 391-400.

* IEEE Control Systems Magazine (December 1982).

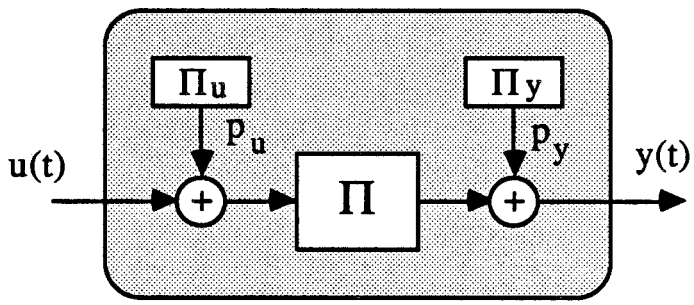
* IEEE Symposium on CACSD, Santa Barbara, March 1985.

Table des transformées en p et en z

G(p)	g(t)	G(z)
e^{-kTp}	$\delta(t - kT)$	z^{-k}
1	$\delta(t)$	1 ou z^{-0}
$\frac{1}{p}$	$u(t)$	$\frac{z}{z-1}$
$\frac{1}{p^2}$	t	$\frac{Tz}{(z-1)^2}$
$\frac{1}{p^3}$	$\frac{1}{2!}t^2$	$\frac{T^2 z(z+1)}{2(z-1)^3}$
$\frac{1}{p - \frac{1}{T}\ln a}$	$a^{\frac{t}{T}}$	$z/(z-a)$
$\frac{1}{p+a}$	e^{-at}	$\frac{z}{z - e^{-aT}}$
$\frac{1}{(p+a)^2}$	$t e^{-at}$	$\frac{Tz e^{-aT}}{(z - e^{-aT})^2}$
$\frac{1}{(p+a)^3}$	$\frac{t^2}{2} e^{-at}$	$\frac{T^2 e^{-aT} z}{2(z - e^{-aT})^2} + \frac{T^2 e^{-2aT} z}{(z - e^{-aT})^3}$
$\frac{a}{p(p+a)}$	$1 - e^{-at}$	$\frac{(1 - e^{-aT}) z}{(z-1)(z - e^{-aT})}$
$\frac{a}{p^2(p+a)}$	$t - \frac{1 - e^{-at}}{a}$	$\frac{Tz}{(z-1)^2} - \frac{(1 - e^{-aT}) z}{a(z-1)(z - e^{-aT})}$
$\frac{a}{p^3(p+a)}$	$\frac{1}{2!}\left(t^2 - \frac{2}{a}t + \frac{2}{a^2} - \frac{2}{a^2}e^{-aT}\right)$	$\frac{T^2 z}{(z-1)^3} + \frac{(aT-2)Tz}{2a(z-1)^2} + \frac{z}{a^2(z-1)} - \frac{z}{a^2(z - e^{-Ta})}$
$\frac{\omega_0}{p^2 + \omega_0^2}$	$\sin \omega_0 t$	$\frac{z \sin \omega_0 T}{z^2 - 2z \cos \omega_0 T + 1}$
$\frac{p}{p^2 + \omega_0^2}$	$\cos \omega_0 t$	$\frac{z(z - \cos \omega_0 T)}{z^2 - 2z \cos \omega_0 T + 1}$
$\frac{\omega_0}{p^2 - \omega_0^2}$	$\operatorname{sh} \omega_0 t$	$\frac{z \operatorname{sh} \omega_0 T}{z^2 - 2z \operatorname{ch} \omega_0 T + 1}$
$\frac{p}{p^2 - \omega_0^2}$	$\operatorname{ch} \omega_0 t$	$\frac{z(z - \operatorname{ch} \omega_0 T)}{z^2 - 2z \operatorname{ch} \omega_0 T + 1}$
$\frac{\omega_0^2}{p(p^2 - \omega_0^2)}$	$\operatorname{ch} \omega_0 t - 1$	$\frac{z(z - \operatorname{ch} \omega_0 T)}{z^2 - 2z \operatorname{ch} \omega_0 T + 1} - \frac{z}{z-1}$
$\frac{\omega_0^2}{p(p^2 + \omega_0^2)}$	$1 - \cos \omega_0 t$	$\frac{z}{z-1} - \frac{z(z - \cos \omega_0 T)}{z^2 - 2z \cos \omega_0 T + 1}$
$\frac{a^2}{p(p+a)^2}$	$1 - (1 + at)e^{-at}$	$\frac{z}{z-1} - \frac{z}{z - e^{-aT}} - \frac{aT e^{-aT} z}{(z - e^{-aT})^2}$
$\frac{a^2(p+b)}{p(p+a)^2}$	$b - b e^{-at} + a(a-b) t e^{-at}$	$\frac{bz}{z-1} - \frac{bz}{z - e^{-aT}} + \frac{a(a-b) T e^{-aT} z}{(z - e^{-aT})^2}$
$\frac{\omega_0}{(p+a)^2 + \omega_0^2}$	$e^{-at} \sin \omega_0 t$	$\frac{z e^{-aT} \sin \omega_0 T}{z^2 - 2z e^{-aT} \cos \omega_0 T + e^{-2aT}}$
$\frac{p+a}{(p+a)^2 + \omega_0^2}$	$e^{-at} \cos \omega_0 t$	$\frac{z^2 - z e^{-aT} \cos \omega_0 T}{z^2 - 2z e^{-aT} \cos \omega_0 T + e^{-2aT}}$

Principe du Modèle Interne : cas monovariable

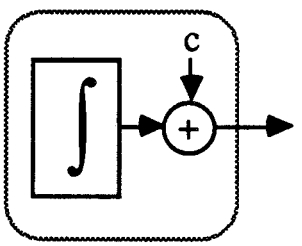
Pour illustrer ce principe, considérons le processus suivant où les perturbations sont modélisées sur l'entrée et sur la sortie .



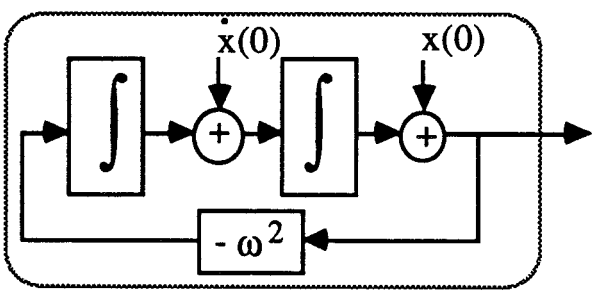
Les processus générateur Π_u et Π_y ont des sorties p_u et p_y et pas d'entrée. Ils évoluent sous l'effet de leurs conditions initiales.

Par exemple, un processus générateur :

- d'une constante c est un intégrateur



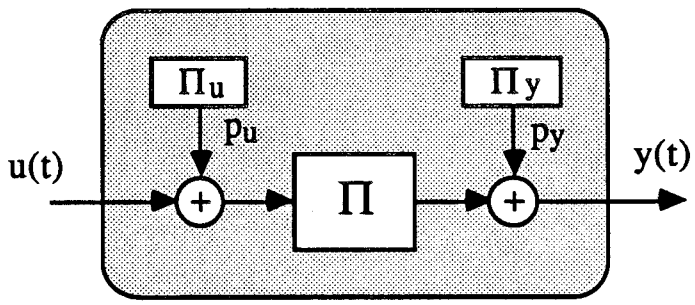
- d'une sinusoïde est un oscillateur



Le principe du modèle interne s'introduit en expliquant pourquoi un contrôleur intégral est utilisé pour rejeter une perturbation constante non mesurable p_0 avec une entrée nulle.

Principe du Modèle Interne : cas monovariable

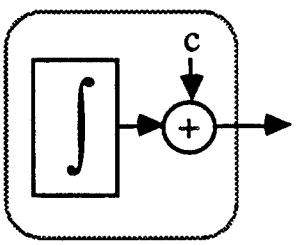
Pour illustrer ce principe, considérons le processus suivant où les perturbations sont modélisées sur l'entrée et sur la sortie .



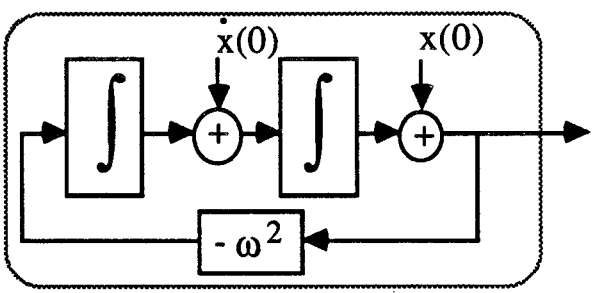
Les processus générateur Π_u et Π_y ont des sorties p_u et p_y et pas d'entrée. Ils évoluent sous l'effet de leurs conditions initiales.

Par exemple, un processus générateur :

- d'une constante c est un intégrateur

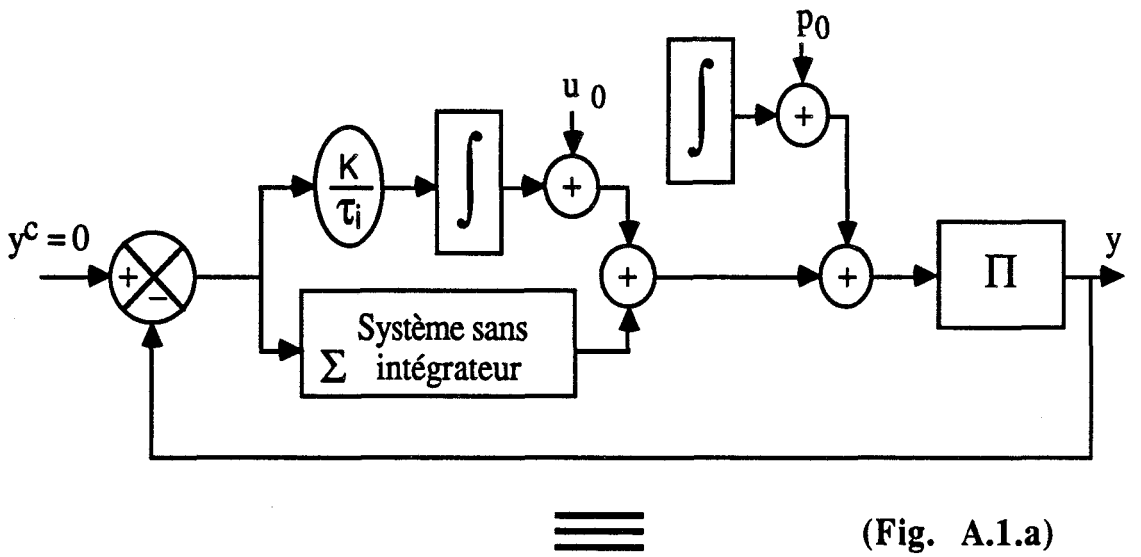


- d'une sinusoïde est un oscillateur

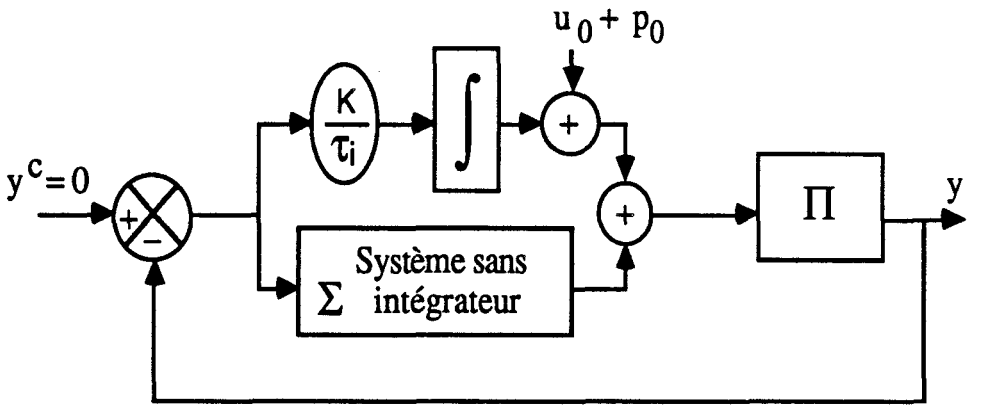


Le principe du modèle interne s'introduit en expliquant pourquoi un contrôleur intégral est utilisé pour rejeter une perturbation constante non mesurable p_0 avec une entrée nulle.

En effet les deux schémas suivant sont équivalents :



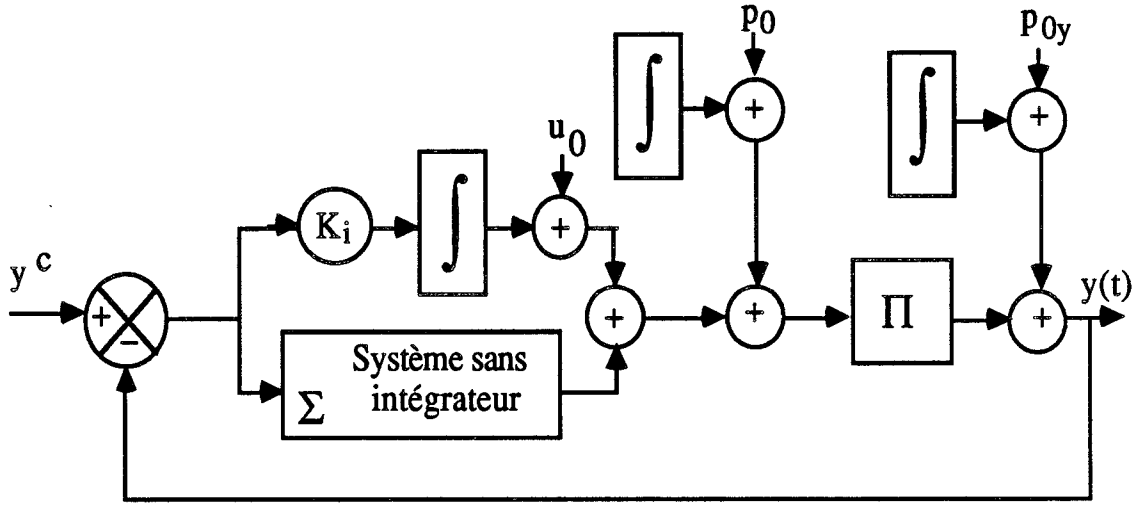
(Fig. A.1.a)



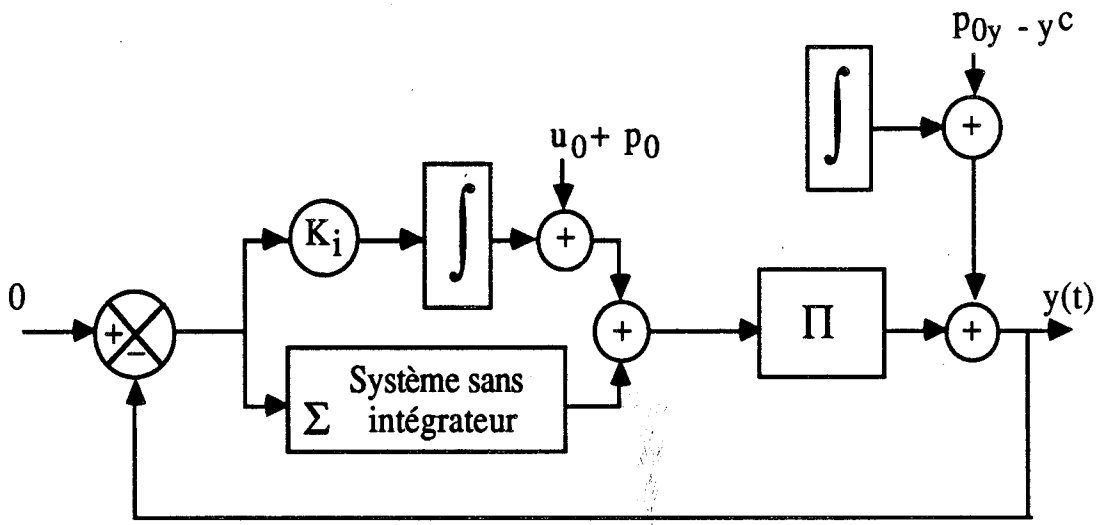
(Fig. A.1.b)

Si l'asservissement {Processus + contrôleur intégral} est stable asymptotiquement, on sait que le système éliminera indifféremment les conditions initiales (u_0 Fig. A.1.a) ou (u_0+p_0 Fig. A.1.b) de l'intégrateur et que l'erreur $\varepsilon(t)$ convergera vers 0, assurant ainsi l'objectif de régulation.

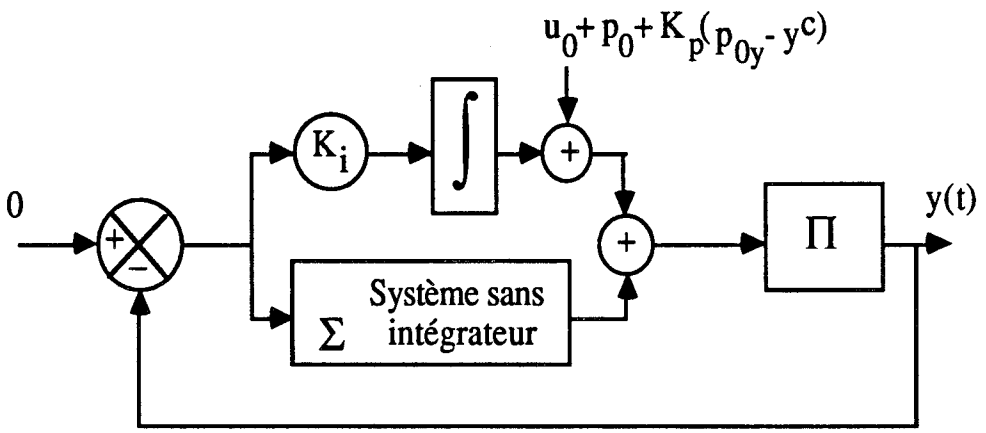
Pour un système sans intégrateur, de gain statique K_p et soumis à une perturbation p_{0y} sur la sortie et à une consigne y^c, toutes deux constantes, l'équivalence des 3 schémas suivants (Fig. A.2.a, .b et .c) permet de se ramener au cas précédent.



(Fig. A.2.a)

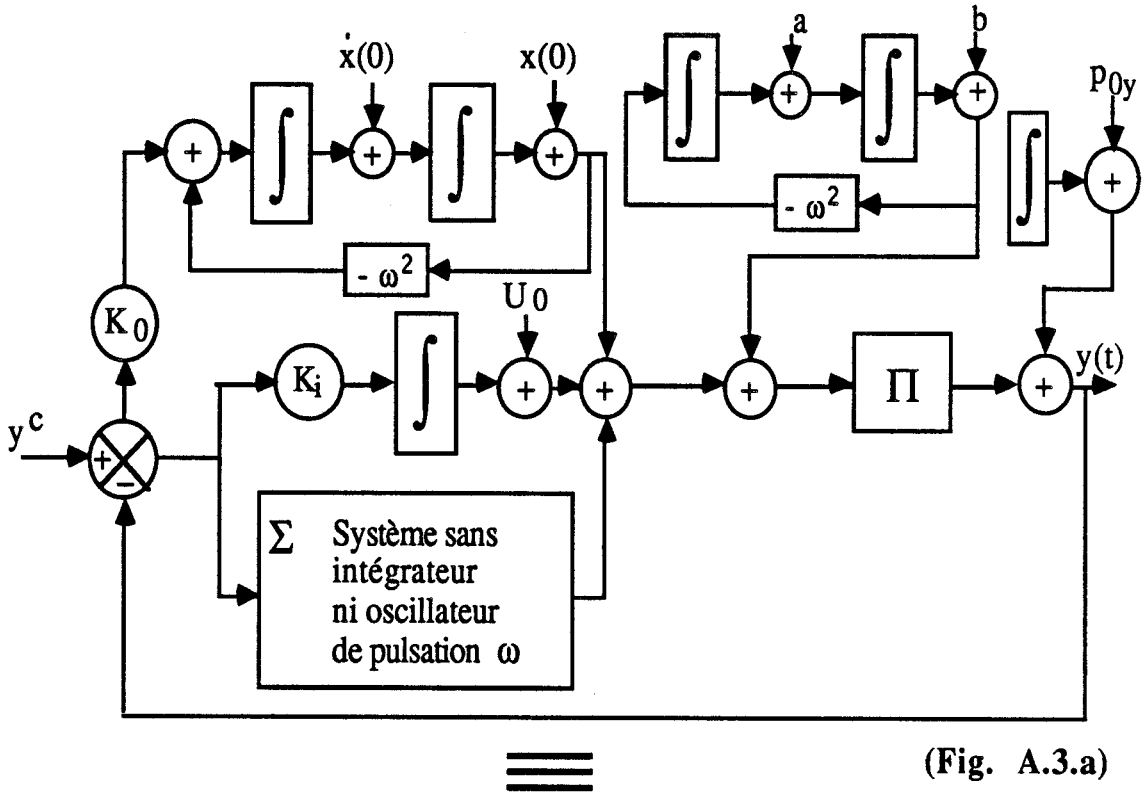


(Fig. A.2.b)

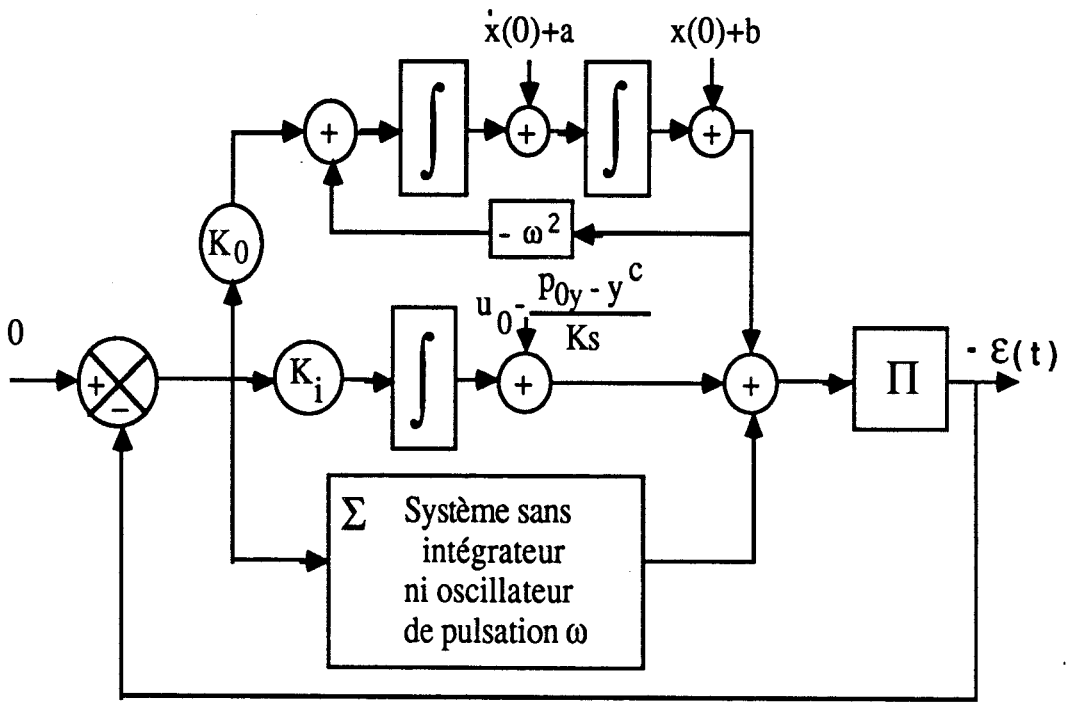


(Fig. A.2.c)

De même, si l'entrée est de plus perturbée par une sinusoïde de pulsation ω, par le même procédé, une loi de commande qui contient en parallèle un oscillateur permet de rejeter cette perturbation.



(Fig. A.3.a)



(Fig. A.3.b)

Le principe du modèle interne condense ce résultat et s'exprime très simplement en monovariable.

Pour assurer une erreur permanente nulle à un processus soumis à des signaux de perturbations et de consigne dont on connait les processus générateurs, il suffit de :

1) ramener toutes ces perturbations sur l'entrée :

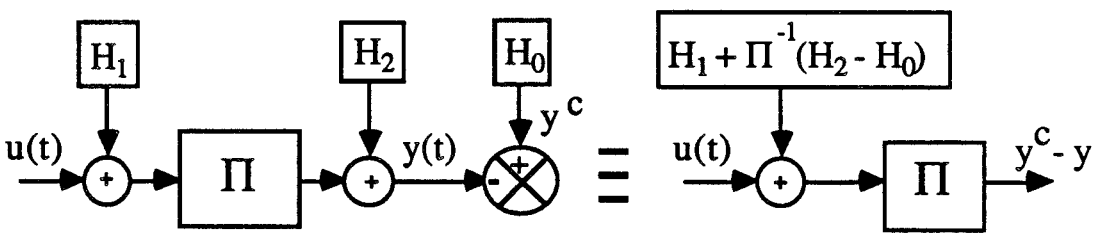


Fig. A.4.

2) placer un modèle du processus générateur de cette perturbation ramenée sur l'entrée dans une structure de loi de commande en réseau correcteur de manière à ce que cette composante soit commandable par ε et observable par u.

3) stabiliser par une technique ad-hoc l'asservissement constitué du processus et de sa loi de commande.

Remarque :

Consigne et perturbation filtrées :

Supposons qu'on produise les variations de consigne avec un modèle de référence du 1er ordre, pour adoucir les variations de consigne en échelon, le modèle générateur de la perturbation est alors celui de la figure A.5.a

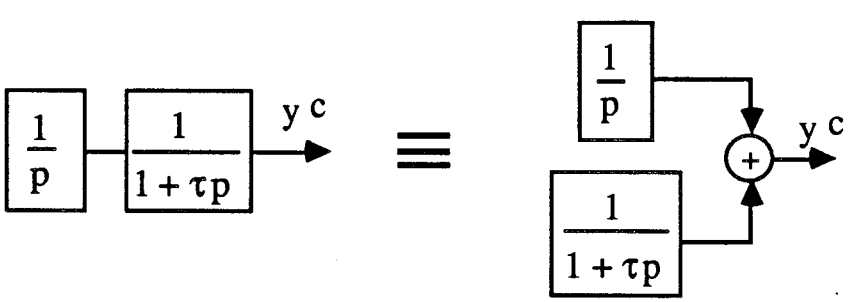


Fig. A.5.a

En notant que la sortie du filtre stable du 1er ordre converge vers 0 lorsque l'entrée est identiquement nulle, cette perturbation est rejetée si on compense seulement la composante non évanescente de la consigne (ou de la perturbation).

Le principe du modèle interne peut donc s'appliquer en limitant la modélisation des perturbations à rejeter à la composante non asymptotiquement stable des processus générateurs.

Autrement dit, si la commande ne contient pas certains pôles stables du modèle générateur de la perturbation, ce n'est asymptotiquement pas grave puisque ça correspond à l'annulation d'un transitoire bien amorti. Cette remarque permet de simplifier la structure de commande.

Liste des facettes Shirka

$a-verifier

Facette d'attachement procédural. Elle est suivie d'un ou de plusieurs schémas prédicats. Ceux-ci doivent être tous vérifiés pour qu'une valeur de l'attribut auquel est associée la facette soit admissible.

$apres-ajout

Facette d'attachement procédural. Les méthodes qui la suivent sont déclenchées en cas d'ajout de valeur à l'attribut, après que cet ajout ait été réalisé.

$apres-mod

Facette d'attachement procédural. Les méthodes qui la suivent sont déclenchées en cas de modification de la valeur de l'attribut, après que cette modification ait été réalisée.

$apres-sup

Facette d'attachement procédural. Les méthodes qui la suivent sont déclenchées en cas de suppression de la valeur de l'attribut, après que cette suppression ait été réalisée.

$apres-vi

Facette de visualisation. Les méthodes qui la suivent sont déclenchées, lors de la visualisation d'une instance demandée par la commande ou la fonction **vi**, après que l'attribut auquel elle est attachée ait été lui-même visualisé.

$avant-ajout

Facette d'attachement procédural. Les méthodes qui la suivent sont déclenchées en cas d'ajout de valeur à l'attribut, avant que cet ajout ait été réalisé.

$avant-mod

Facette d'attachement procédural. Les méthodes qui la suivent sont déclenchées en cas de modification de la valeur de l'attribut, avant que cette modification ait été réalisée.

$avant-sup

Facette d'attachement procédural. Les méthodes qui la suivent sont déclenchées en cas de suppression de la valeur de l'attribut, avant que cette suppression ait été réalisée.

$avant-vi

Facette de visualisation. Les méthodes qui la suivent sont déclenchées, lors de la visualisation d'une instance demandée par la commande ou la fonction **vi**, avant que l'attribut auquel elle est attachée ne soit lui-même visualisé.

$card-max

Facette de restriction. Elle permet de préciser le nombre maximum de valeurs admises pour un attribut multi-valué.

$card-min

Facette de restriction. Elle permet de préciser le nombre minimum de valeurs requises pour un attribut multi-valué.

$com

Facette de commentaire. Elle est suivie d'une liste de chaînes de caractères exploitées lors de l'instanciation interactive de la classe (commande **cr-inst**). Les commentaires facilitent la maintenance des bases de connaissance.

$defaut

Facette d'inférence. La valeur qui la suit est retenue comme valeur de l'attribut si aucun autre moyen d'inférence n'a fonctionné auparavant.

$domaine

Facette de restriction. Elle est suivie d'une liste des valeurs possibles de l'attribut.

$en-vi

Facette de visualisation. Les méthodes qui la suivent sont déclenchées, lors de la visualisation d'une instance demandée par la commande ou la fonction **vi**, entre les visualisations de valeurs consécutives de l'attribut multi-valué auquel elle est attachée.

$intervalle

Facette de restriction. Elle est suivie d'un ou de plusieurs intervalles donnés par une borne inférieure et une borne supérieure. La valeur de l'attribut correspondant doit être située dans l'un de ces intervalles. Cette facette ne peut être associée qu'à un attribut de type simple.

$liste-de

Facette de typage. Elle joue le même rôle que la facette **$un**, mais indique un attribut multi-valué.

$pour-vi

Facette de visualisation. Les méthodes qui la suivent sont déclenchées, lors de la visualisation d'une instance demandée par la commande ou la fonction **vi**, pour visualiser l'attribut auquel elle est attachée.

$sauf

Facette de restriction. Elle est suivie d'une liste de valeurs que l'attribut correspondant ne peut pas prendre.

$si-echec

Facette d'attachement procédural. Les méthodes qui la suivent sont appelées en cas d'échec de l'obtention de la valeur de l'attribut par inférence.

$si-succes

Facette d'attachement procédural. Les méthodes qui la suivent sont appelées en cas de succès de l'obtention de la valeur de l'attribut par inférence.

$sib-exec

Facette d'inférence et d'attachement procédural. Elle est suivie d'un ou de plusieurs schémas méthode, avec ou sans filtrage. Ceux-ci sont associés à des fonctions Le-Lisp. Lors de la recherche de la valeur de l'attribut, un schéma methode est instancié et son instance est transmise à la fonction qui la complète. L'instance est renvoyée à Shirka qui propage le résultat.

$sib-filtre

Facette d'inférence. Elle est suivie d'un ou plusieurs filtres. Un filtre décrit les instances recherchées. La valeur de l'attribut est extraite des instances qui satisfont le filtre (cf. D.II.2.4.2.).

$un

Facette de typage. Elle est suivie du nom d'un type simple (**entier**, **reel**, **chaine**, **booleen** ou **symbole**), ou du nom d'une classe. Dans ce dernier cas, les valeurs de l'attribut correspondant sont des instances de cette classe.

$valeur

Facette d'inférence. Elle fournit une valeur d'attribut valable pour toutes les instances de la classe.

$var<-

Indique que l'attribut peut recevoir la valeur de la variable dont le nom suit.

$var->

Indique que la valeur de l'attribut peut être transmise à la variable dont le nom suit.

$var-liste<-

Semblable à **$var<-** mais précise, dans le cas d'une variable associée à un attribut multi-valué, que la valeur reçue de la variable est une liste considérée dans son ensemble.

$var-liste->

Semblable à **$var->** mais précise, dans le cas d'un attribut multi-valué, que la valeur transmise à la variable est la liste dans son ensemble.

$var-nom

Permet d'associer explicitement une variable à un attribut et de lever ainsi les ambiguïtés liées au fait que par défaut une variable fait référence à l'attribut de même nom et que plusieurs attributs de même nom peuvent apparaître dans un même schéma.

[RECHENMANN 87 a]

Les fonctions SHIRKA

La connaissance de quelques fonctions Lisp de Shirka est nécessaire pour l'écriture d'attachements procéduraux, méthodes ou prédicats. Ces fonctions sont utiles, en particulier si Shirka est vu comme un sous-système d'un système informatique plus vaste.

1. def-sh

La fonction **def-sh** a pour paramètre une description complètement parenthésée d'un schéma dans laquelle les parenthèses encadrent les schémas, les attributs, les facettes et les intervalles mais pas les listes de valeurs. La fonction permet de créer des schémas directement dans une fonction Lisp.

```
(def-sh '(point-premier-quadrant
        (sorte-de (= point))
        (xcoord   ($intervalle (0.0 +inf)))
        (ycoord   ($intervalle (0.0 +inf)))))
= #[schema () point-premier-quadrant (#[attribut () () xcoord
  (#[$intervalle () () (#[intervalle () () 0.0 +inf])]) $un reel () (( 0.0
  +inf)) ()] #[attribut () () ycoord (#[$intervalle () () (#[intervalle () ()
  0.0 +inf])]) $un reel () ((0.0 +inf)) ()]) (point) () ()]
```

2. #:shirka:noyau:schema.

La fonction **:schema** n'a qu'un seul paramètre, le nom d'un schéma, de classe ou d'instance. Elle ramène le schéma lui-même, en représentation interne Shirka.

```
(#:shirka:noyau:schema 'point)
= #[schema () point (#[attribut () () xcoord (#[$com () () (abscisse du
  point)] #[$un () () reel]) $un reel () () ()] #[attribut
  () () ycoord (#[$com () () (ordonnee du point)] #[$un () () reel])
  $un reel () () ()]) (objet-du-plan) (%point-113 %point-114
  %point-119 point-premier-quadrant) (pt1 pt2)]
```

3. #:shirka:noyau:nom-sch.

C'est la fonction réciproque de la précédente. Elle a pour argument la représentation interne d'un schéma ; elle ramène son nom.

```
(#:shirka:noyau:nom-sch (#:shirka:noyau:schema 'point))
= point
```

4. #:shirka:moteur:valeur? et val?.

La fonction **:valeur?** appelle le moteur d'inférence. Elle possède deux paramètres : le nom d'un attribut dont la valeur est requise et le nom de l'instance.

La macro **val?** est destinée à faciliter l'écriture des fonctions des attachements procéduraux. En effet, ces fonctions doivent avoir pour paramètre l'instance du schéma correspondant. Elles accèdent aux valeurs des attributs de cette instance par la macro **val?** avec pour argument le nom de l'attribut "quoté". La fonction **:valeur?** peut également être employée.

```
(#:shirka:moteur:valeur? 'longueur 's1 )
= 6.081291E0
```

5. #:shirka:moteur:aj-valeur et affect.

La fonction **:aj-valeur** possède quatre paramètres: une instance (et non pas son nom), un nom d'attribut, une valeur pour cet attribut et **nil** si la valeur doit être ajoutée en début de liste dans le cas d'un attribut multi-valué, **t** si en fin de liste. La fonction ramène la valeur si celle-ci a pu être affectée à l'attribut, **nil** sinon, par exemple en cas de non admissibilité de la valeur.

La macro **affect** est destinée à faciliter l'écriture des fonctions des attachements procéduraux. En effet, ces fonctions doivent compléter l'instance qui leur est passée en paramètre en assignant une valeur à l'attribut résultat. La fonction **:aj-valeur** peut être employée à ces fins, mais la macro **affect** est plus simple: elle admet deux arguments, le nom de l'attribut "quoté" et la valeur à lui affecter.

```
(#:shirka:moteur:aj-valeur (#:shirka:noyau:schema 's1)
                'longueur
                3.45
                nil)
= 3.450000E0
```

6. #:shirka:moteur:mod-valeur.

La fonction **:mod-valeur** possède quatre paramètres: une instance, un nom d'attribut, la valeur de cet attribut à modifier et la nouvelle valeur. La

fonction ramène la nouvelle valeur si celle-ci a pu être affectée à l'attribut, nil sinon, par exemple dans le cas où la nouvelle valeur n'est pas admissible ou que la valeur à remplacer n'apparaît pas dans l'attribut.

```
(#:shirka:moteur:mod-valeur (#:shirka:noyau:schema 's1)
                'longueur
                3.45
                6.08
                nil)
= 6.08
```

7. #:shirka:moteur:sup-valeur.

La fonction **:sup-valeur** possède trois paramètres: une instance, un nom d'attribut, la valeur de cet attribut qui doit être supprimée. La fonction ramène la valeur si celle-ci a pu être supprimée et nil sinon, c'est-à-dire quand la valeur n'est pas présente dans l'attribut.

```
(#:shirka:moteur:sup-valeur (#:shirka:noyau:schema 's1)
                'longueur
                6.08
                nil)
= 6.08
```

8. #:shirka:moteur:sup-inst.

La fonction **:sup-inst** ne possède qu'un seul paramètre, l'instance à supprimer, et renvoie son nom.

```
(#:shirka:moteur:sup-inst (#:shirka:noyau:schema 'd2))
= d2
```

9. #:shirka:moteur:creer-instance

La fonction **:creer-instance** permet de créer des instances d'une classe donnée à l'intérieur d'une fonction Lisp. Elle possède trois paramètres : le nom de la classe, la liste des valeurs des attributs sous la forme d'une A-liste (**nom d'attribut . valeur**), où **valeur** est évaluée à l'intérieur de **:creer-instance**, et le nom de l'instance. Si ce dernier paramètre reçoit la valeur nil, le nom de l'instance est généré par le système. La fonction ramène nil si la création n'a pas pu s'effectuer, soit parce que la classe est inconnue, soit que l'un des attributs est inconnu dans la classe, soit que la valeur d'un attribut n'est pas admissible ou que l'instance existe déjà. Sinon, la fonction ramène le nom de l'instance créée.

```
(#:shirka:moteur:creer-instance 'point
                    '((xcoord . 0.0) (ycoord . 0.0))
                    'pt4)
= pt4
```

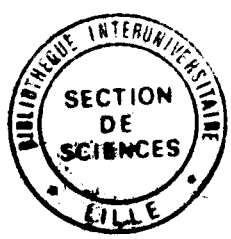
10. #:shirka:vi:visualiser

Cette fonction permet de visualiser une instance à différents niveaux, en exploitant éventuellement les facettes **$pour-vi**, **$avant-vi**, **$en-vi** et **$apres-vi**. Elle possède trois paramètres : l'instance à visualiser, le nombre de niveaux et un booléen indiquant si les valeurs des attributs doivent être inférées avant visualisation.

```
(#:shirka:vi:visualiser (#:shirka:noyau:schema 'pt1) 1 nil)

{ pt1
   est-un = point ;
   xcoord =  1.500000E0 ;
   ycoord =  2.340000E0 ;
} = pt1
```

[RECHENMANN 87 a]



RESUME

Nous présentons dans ce mémoire les différentes étapes de la réalisation d'un système à base de connaissances, d'aide à la synthèse de lois de commande, en utilisant des techniques d'Intelligence Artificielle (IA) associées à un langage de représentation centrée objet.

Dans le premier chapitre, nous présentons les différents outils informatiques utilisés pour la synthèse de commande des systèmes d'états continus. Aussi, nous décrivons les différentes parties formant un système à base de connaissances. La fin de cette partie est consacrée à la synthèse bibliographique d'un des premiers essais d'utilisation de système à base de connaissances en C S C A O.

Le second chapitre est consacré à la description des outils automatiques nécessaires à une synthèse de lois de commande. Ainsi, nous présentons la notion de signal et nous caractérisons les systèmes dynamiques. En dernier lieu, nous rappelons les différentes étapes d'une synthèse de lois de commande.

Dans le dernier chapitre, nous décrivons les outils informatiques utilisés dans notre prototype. Aussi, nous définissons les différents objets, faits et règles de production formant notre base de connaissances. Dans la dernière partie de ce chapitre, des exemples d'application permettent de présenter le mode de fonctionnement de notre système.

Mots Clés :

- *Automatique*
- *CAO en Automatique (CSCAO)*
- *Système à Base de Connaissances*
- *Représentation Centrée-Objet*
- *Calcul Formel*
- *Système Linéaire Monovariable*
- *Synthèse de Lois de Commande*