N° d'ordre : **1175**

THESE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ

en

PRODUCTIQUE, AUTOMATIQUE ET INFORMATIQUE INDUSTRIELLE

par

SAMIR EL KHATTABI
MAÎTRE INFORMATIQUE

INTÉGRATION DE LA SURVEILLANCE DE BAS NIVEAU DANS LA CONCEPTION DES SYSTÈMES À ÉVÈNEMENTS DISCRETS : APPLICATION AUX SYSTÈMES DE PRODUCTION FLEXIBLES.

Soutenue le 29 Septembre 1993 devant la commission d'Examen

M C.	ANDRÉ	**Rapporteur**
M J.P.	BOUREY	**Examinateur**
M D.	CORBEEL	**Examinateur, Directeur de travail**
M J.C.	GENTINA	**Examinateur, Directeur de Thèse**
M M.	STAROSWIECKI	**Examinateur, Président de jury**
M R.	VALETTE	**Rapporteur**

Cette thèse a été préparée au Laboratoire d'Automatique et d'Informatique Industrielle de Lille, CNRS D 1440, de l'Ecole Centrale de Lille.

AVANT-PROPOS

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Automatique et d'Informatique Industrielle de Lille (LAIL), sous la direction scientifique de Monsieur J.C. GENTINA, Professeur et Directeur de l'Ecole Centrale Lille, avec la collaboration de Monsieur D. CORBEEL, Maître de Conférences à Centrale Lille. Je tiens à leur témoigner toute ma reconnaissance pour l'aide précieuse qu'ils m'ont apporté tant sur le plan scientique que sur le plan moral.

Je suis très reconnaissant à Monsieur C. ANDRÉ, Professeur à l'Université de Nice, et à Monsieur R. VALETTE, Directeur de Recherche CNRS au LAAS de Toulouse, de l'honneur qu'ils me font en acceptant de juger ce travail et d'être les rapporteurs de cette thèse.

Je suis flatté de la présence de Monsieur M. STAROSWIECKI, Professeur à l'Université des Sciences et Technologies de Lille (USTL), qui a accepté d'être examinateur et président de jury de cette thèse.

Je tiens également à remercier Monsieur J.P. BOUREY, Maître de Conférences à Centrale Lille, Habilité à diriger des Recherches, d'avoir accepté de participer à ce jury.

Qu'il me soit permis de remercier tous les membres du LAIL, pour le soutien qu'ils ont su m'apporter, et en particulier, Messieurs E. CASTELAIN, E. CRAYE, P. YIM, A.K.A. TOGUYENI, mon collègue de l'équipe surveillance, C. AUSFELDER et Madame A. HEBRARD.

Mes remerciements vont également à Messieurs A.E.K. SAHRAOUI et D. DELFIEU du LAAS de Toulouse et Mademoiselle M.A. PERALDI du LASSY de Nice, mes collègues du projet DRED "Réseaux de Petri et langages synchrones", pour leur collaboration scientifique.

Je remercie aussi très sincèrement Madame R. DUPLOUICH, responsable du Centre de Documentation de Centrale Lille, pour m'avoir facilité la recherche de documents scientifiques ; et Monsieur M. VANGREVENINGE pour avoir pris soin de la reprographie de ce document.

Enfin, je tiens à remercier mes parents pour m'avoir guidé dans mes pas depuis mon enfance, et ma compagne pour la patience dont elle a fait preuve durant la rédaction de ce mémoire.

SOMMAIRE

INTRODUCTION GENERALE.

Les travaux présentés dans ce mémoire, ont été réalisés au sein de l'équipe Systèmes à Evénements Discrets (**SED**), du Laboratoire d'Automatique et d'Informatique Industrielle de Lille (**LAIL**) de l'Ecole Centrale de Lille, sous la direction du Professeur J.C. GENTINA. Ces travaux de recherche concernent l'intégration de la surveillance de bas niveau dans la conception des SED.

L'informatique est devenue omniprésente dans notre vie et ce, dans tous les domaines. Les SED n'échappent pas à la règle. L'automatisation des procédés industriels conduit à une amélioration des performances et évite à l'homme de répéter des tâches similaires et ennuyeuses. Cependant, cette automatisation a ses inconvénients. Ainsi, dans un Système Automatisé (**SA**), face à une situation inconnue (non programmée), le Système de Commande (**SC**) se trouve "incapable" de réagir. Une situation plus critique, telle que l'évolution anormale du procédé, risque de se produire sans que le SC ne la détecte.

La mise en place des Systèmes Flexibles de Production Manufacturière (**SFPM**) demande des investissements considérables aussi bien sur le plan matériel, que sur le plan de

la conception du système de contrôle/commande. Il est donc souhaitable d'automatiser la démarche de conception afin d'accélérer la phase d'étude, de valider celle-ci, et de minimiser les fautes de conception. Plusieurs travaux ont été menés dans ce sens, au sein de la communauté de recherche Française. Citons à ce titre quelques équipes :

- le **L**aboratoire d'**A**utomatique et d'**A**nalyse des **S**ystèmes (**LAAS**) de Toulouse, pour ses travaux sur la modélisation et la simulation des SFPM par systèmes à base de règles [BAKO, 90], et l'intégration de la surveillance au SC [COMBACAU 90 ET 91], [SAHRAOUI, 87 ET 92],
- le **L**aboratoire d'**A**utomatique et de **C**ommande **N**umérique de **N**ancy (**LACN**), pour ses travaux sur le Génie Automatique [FRACHET, 87], [MOREL, 92], et la surveillance des **E**léments de la **P**artie **O**pérative (**EPO**) basée sur la notion de filtre comportemental [ALANCHE, 86], [SFALCIN, 89],
- l'équipe I3S du **L**aboratoire de **S**ignaux et **S**ystèmes (**LASSY**) de Nice-Sophia Antipolis, pour ses travaux sur une approche mixte (synchrone et asynchrone) d'un système temps-réel. Cette approche permet d'intégrer en partie les aspects réactifs des SFPM [ANDRÉ, 91 ET 92], [FANCELLI, 91]. Ces travaux sont complétés par une approche de conception et de validation des systèmes réactifs [PERALDI, 93]. Il s'agit plus particulièrement des applications qui effuctuent du contrôle avec une prise en charge de l'orchestration du fonctionnement normal mais également de sa surveillance et les changements de mode de fonctionnement à opérer en cas d'erreurs,
- le **L**aboratoire d'**A**utomatique de **G**renoble (**LAG**), pour ses travaux sur la modélisation hybride des différents niveaux hiérarchiques du SC [LONG, 92], [DEVAPRIYA, 92], et l'évaluation des performances des SFPM par RdP non autonomes et Réseaux à files d'attente à capacité limitée [BOUCHOUCH, 92],

Actuellement au LAIL, une méthodologie est développée afin d'assister le concepteur dans sa démarche de conception d'un SFPM automatisé depuis la spécification du cahier des charges jusqu'à l'implantation sur site. Cette méthodologie constitue le projet **CASPAIM** (**C**onception **A**ssistée des **S**ystèmes de **P**roduction **A**utomatisés en **I**ndustrie **M** anufacturière). Plusieurs travaux ont été réalisés dans ce domaine, [BOUREY, 88, 89, 91 ET 93], [CASTELAIN, 87], [CORBEEL, 79], [CRAYE, 89], [CRUETTE, 91A], [KAPUSTA, 88].

Néanmoins, ces travaux n'intègrent pas ou très peu les aspects de sécurité liés au fonctionnement du système de production. L'objet de ce mémoire est de présenter un système de surveillance devant s'interfacer avec le SFPM automatisé. Ce système de

surveillance devra garantir un certain nombre de propriétés : *fiabilité, disponibilité* et *maintenabilité* [LAPRIE, 85]. Le travail présenté ici sera intégré dans le cadre du projet CASPAIM.

La fiabilité et la sécurité existent depuis que les outils font partie de notre vie. Depuis toujours, l'utilisation d'outils a conduit à leur amélioration par l'*apprentissage de la panne et de l'accident* [VILLEMEUR, 88]. Les systèmes devenant de plus en plus complexes, la maîtrise de leur fiabilité devient une nécessité absolue. Il est donc indispensable de disposer de méthodes et de techniques qui répondent à ces besoins. En effet, devant la complexité et l'automatisation de systèmes quels qu'ils soient, les méthodologies de conception des systèmes automatisés doivent intégrer la notion de fiabilité, dès le début. Afin de répondre à ce besoin, un système de surveillance doit être conçu de front avec le SC. La conception de la surveillance a posteriori engendre des surcoûts assez importants [NIEL, 92].

Dans des domaines industriels comme le spatial et le nucléaire, où la fiabilité et la sécurité sont fondamentalement nécessaires et obligatoires, les méthodes et les mesures associées à ces notions se sont multipliées. D'autres notions comme la disponibilité et la maintenabilité, adjointes à la fiabilité, forment une science à part entière : la *sûreté de fonctionnement* [VILLEMEUR, 88].

La maîtrise des risques économiques ou humains, qui peuvent être engendrés par des systèmes de complexité croissante, ne peut se contenter de l'expérience acquise. La prévention et le traitement éventuel de ces risques doivent être intégrés dans la conception des systèmes automatisés.

Les techniques actuelles de sûreté de fonctionnement font abstraction de la commande [VILLEMEUR, 88]. Elles amènent donc l'introduction des capteurs spécialisés. Une telle *redondance matérielle,* pas toujours nécessaire, doit être évitée. La solution consiste à utiliser autant que possible les capteurs destinés à la commande.

Généralement, les problèmes liés à la sûreté du système et à la commande sont abordés séparément. Dans ce contexte, la sûreté du système de commande est considérée a posteriori. La solution la plus utilisée consiste en l'introduction de redondance massive : dupliquer le système.

Il nous parait donc essentiel d'intégrer les aspects de sûreté de fonctionnement dans la conception des systèmes de commande en évitant l'incorporation de redondances massives et

matérielles. L'approche proposée permet de rendre le SC sûr de fonctionnement.

Ce mémoire se compose de quatre parties.

Dans la première partie, nous positionnerons la surveillance par rapport à la sûreté de fonctionnement, la maintenance et la supervision. Nous commencerons d'abord par définir les notions liées à la sûreté de fonctionnement.
Ensuite, nous établirons un état de l'art des approches de surveillance existantes. Enfin, nous proposerons une approche permettant de rendre le système de production sûr de fonctionnement. Cette proposition est constituée de deux modules principaux : le Module de Contrôle de Commande (**MCC**) et le Module des Filtres de Commande (**MFC**).

Dans la deuxième partie, les modèles et les outils utilisables pour la conception du Contrôle de Commande (**CC**), seront présentés. Les avantages et inconvénients de chacun, seront passés en revue.
Le deuxième volet traitera du MCC [ELKHATTABI, 91 ET 92B], qui aura pour mission de valider la réalisation des actions par le système physique. Ce module permet de rendre compte de l'évolution du procédé.

Nos besoins de modélisation, nous ont amené à développer un outil : les **O**bjets **C**ommandables (**OC**) [ELKHATTABI, 92A]. Cet outil fera l'objet d'une présentation au cinquième chapitre de la partie III. Nous énoncerons les définitions et propriétés de cet outil. Dans le second volet de la troisième partie, une mise en application des OC à des fins de modélisation comportementale du système commandé (les Filtres de Commande) sera exposée [ELKHATTABI, 92A]. La présentation d'un outil informatique conçu à cet effet sera abordée [ELKHATTABI, 93B].

La dernière partie portera sur une intégration du système de surveillance à CASPAIM. Nous présenterons succintement les différentes étapes constituant le projet CASPAIM, ainsi que les outils et les modèles qui sont mis en œuvre, pour le mener à terme. Enfin, les problèmes d'intégration de la surveillance à un système de commande seront abordés dans le dernier chapitre.

Les chapitres ont été traités de manière indépendante, volontairement, afin de laisser au lecteur la possibilité de choisir ce qui l'intéresse à partir du sommaire. Le dernier chapitre, fait une synthèse de la démarche de surveillance [ELKHATTABI, 93A], [ANDRÉ, 93], ainsi que son interfacage à un SC (CASPAIM).

PARTIE I.

SURVEILLANCE DES SYSTEMES A EVENEMENTS DISCRETS.

CHAPITRE I.

POSITIONNEMENT DE LA SURVEILLANCE.

INTRODUCTION.

Dans ce chapitre, notre attention portera avant tout sur un éclaircissement des différentes notions voisines de la surveillance. En effet, des termes comme "*supervision*", "*conduite*", "*maintenance*" ou "*sûreté de fonctionnement*" prêtent à confusions. Il est donc nécessaire de les situer les uns par rapport aux autres.

Dans un premier volet, nous aborderons les aspects de la sûreté de fonctionnement. Nous nous intéresserons ensuite à la notion de défaillance et aux concepts qui lui sont attachés. En effet, la fiabilité d'un système dépend de son aptitude, à être non défaillant [VILLEMEUR, 88]. Nous terminerons ce volet par un positionnement de la surveillance par rapport à la sûreté de fonctionnement.

Dans un deuxième volet, nous positionnerons la surveillance par rapport aux notions de supervision et de maintenance.

I. DE LA SÛRETÉ DE FONCTIONNEMENT À LA SURVEILLANCE.

I.1. INTRODUCTION.

Il existe plusieurs termes pour désigner des notions qui nous renseignent sur le fonctionnement du système. Ces notions sont de nature qualitatives et quantitatives; en l'occurrence : *la fiabilité, la disponibilité, la maintenabilité et la sécurité* sont des caractéristiques qualitatives auxquelles sont associées des mesures quantitatives. Chacun de ces concepts donne une vision particulière de la sûreté de fonctionnement. Il est nécessaire d'en donner des définitions claires, en définissant, ce que nous entendons par chacune de ces notions, afin d'éviter toute confusion.

Après avoir défini les différentes caractéristiques de la sûreté de fonctionnement, nous aborderons différents concepts liés à la notion de défaillance. En effet, plusieurs termes (faute, erreur, défaillance, défaut, panne, ...) sont utilisés dans ce domaine avec une certaine ambiguïté.

Ensuite, nous ferons un état de l'art des méthodes de la sûreté de fonctionnement.

Pour clore ce volet, nous établirons le lien entre la sûreté de fonctionnement et la surveillance, qui fait l'objet de ce mémoire.

I.2. SÛRETÉ DE FONCTIONNEMENT : TERMINOLOGIE.

I.2.1. Définition de la sûreté de fonctionnement.

La "*sûreté de fonctionnement*" est définie par LAPRIE [LAPRIE, 85] comme étant *la "qualité" du service que le système délivre, de telle sorte que les utilisateurs puissent lui accorder (au système) une confiance justifiée.* Cette définition dans sa première version a été introduite par CARTER [CARTER, 82]. LAPRIE a substitué le terme "*qualité*" à la notion de "*crédibilité et continuité*", puisqu'il estime que cette notion se prête à une ambiguïté "potentielle" pour des systèmes qui ne sont pas destinés à une utilisation continue. Dans le cadre de notre étude (les SED), la définition de LAPRIE est plus appropriée.

Dans la définition précédemment citée, deux mots-clés apparaissent : *qualité* et *confiance*. Ces deux termes montrent bien l'intérêt d'avoir un système sûr de fonctionnement. La qualité implique l'importance qui est accordée au service rendu. La mission doit être accomplie en conformité totale avec les spécifications qui en sont faites. Quant au degré de confiance accordée au système, il qualifie l'aptitude du système à répondre aux attentes des utilisateurs. Ainsi, plus le système sera *fiable*, plus la confiance qui lui sera accordée, sera accrue. Le terme "fiable" est utilisé ici pour décrire le caractère de ce qui est digne de confiance.

Nous qualifierons le *"service à rendre"*, suite à une demande de l'utilisateur, de *"mission"*. Il est à noter que ce service se caractérise par le comportement du système à un ordre émis par l'utilisateur. Il est donc nécessaire d'avoir une vision *comportementale* du système.

Nous présentons les différentes notions nécessaires à la caractérisation de la sûreté de fonctionnement et principalement la fiabilité. Il est à noter que ces notions seront vues sur le plan qualitatif et non quantitatif. L'approche quantitative est de type probabiliste, où à chaque caractéristique de la sûreté de fonctionnement est associée une mesure.

I.2.2. Fiabilité.

Le terme *"fiabilité"* a été introduit en 1962 par l'Académie des Sciences pour traduire le terme anglais *"Reliability"*. La définition a été formulée ainsi [CHAPOUILLE, 68] :

> *"Grandeur caractérisant la sécurité de fonctionnement, ou mesure de la probabilité de fonctionnement d'un appareillage selon des normes prescrites."*

Ce terme a été construit sur le vieux mot français "fiable" qui caractérise ce qui est digne de confiance (Petit Robert). La fiabilité d'un système est directement liée à sa qualité. Cette dernière caractérise la capacité d'un composant spécifique à répondre aux exigences de fabrication. La norme AFNOR [AFNOR, 77], la définit comme *"l'aptitude d'un produit ou d'un service à satisfaire les besoins des utilisateurs"*. La qualité garantit donc la conformité du produit aux spécifications en phase de fabrication mais également durant sa durée de vie. Généralement, la qualité d'un produit se limite à sa phase de conception et la fiabilité qualifie alors son aptitude à rester conforme aux spécifications durant son exploitation.

Au sens général, la fiabilité caractérise la confiance accordée par l'utilisateur dans le matériel qu'il utilise. La Commission Electrotechnique Internationale, en donne une

définition plus stricte [CEI, 85] :

"La fiabilité est l'aptitude d'un dispositif à accomplir une fonction requise, dans des conditions données, pendant une durée donnée."

Par "*fonction requise*", on qualifie une ou plusieurs fonctions nécessaires à la fourniture d'un service demandé par un utilisateur. Nous utiliserons le terme "mission" pour désigner la tâche à accomplir. Cette mission consiste à fournir un service, sans défaillance du système. Toute mission doit être réalisée selon des spécifications données portant sur la fonction à accomplir, les conditions de réalisation de la mission, et les délais nécessaires à sa réalisation.

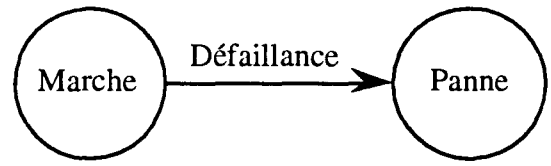


FIGURE I.1 : FIABILITÉ D'UN COMPOSANT [COSTES, 80].

Tout service délivré par le système suite à une requête d'un utilisateur, sera dit "conforme", si les spécifications associées à cette requête sont respectées. En l'occurrence, le service rendu, doit être identique à celui attendu par l'utilisateur et ce dans les délais prescrits. Dès que l'on constate un écart de ces spécifications, le système en question (celui qui doit réaliser la mission) sera dit "défaillant".

L'apparition d'une défaillance sur un composant déclenche le passage de son état de "*marche*" à un état de "*panne*" : *interruption* du service délivré par le système. Le modèle comportemental vu par les utilisateurs est un graphe d'états donné par la FIGURE I.1. L'état de marche correspond à un état de fonctionnement normal. Nous verrons au §I.3.5.2, qu'il peut y avoir un troisième état dit de "*fonctionnement dégradé*".
La modélisation du comportement du système permet d'avoir une vision de son état. Il est ainsi possible d'éviter l'envoi de requêtes à un dispositif en panne.

I.2.3. Maintenabilité.

La *maintenabilité* est définie dans [VILLEMEUR, 88] comme étant "*l'aptitude d'une entité à être maintenue ou rétablie dans un état dans lequel elle peut accomplir une fonction requise, lorsque la maintenance est accomplie dans des conditions données avec des procédures et des moyens prescrits*".

Suite à une défaillance d'un composant (FIGURE I.1), il est nécessaire de mettre en œuvre une *procédure de réparation*, remettant ainsi le système dans son état de fonctionnement normal : état de marche. On remarquera que la maintenabilité et la fiabilité sont deux notions duales. En effet, pour modéliser ce constat, nous utilisons un graphe d'états, qui est présenté par la FIGURE I.2.

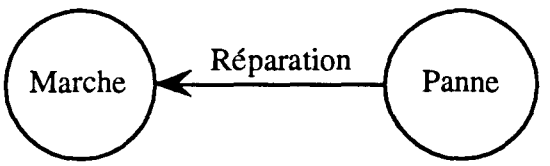


FIGURE I.2 : MAINTENABILITÉ D'UN COMPOSANT.

Ce modèle met en évidence la réparation du système défaillant. Un éclatement du modèle avec un état transitoire (en cours de réparation) entre les deux états, permet de caractériser la phase de réparation.

I.2.4. Disponibilité.

La *disponibilité* est définie dans [VILLEMEUR, 88] comme étant "*l'aptitude d'une entité à être en état d'accomplir une fonction requise dans des conditions données et à un instant donné*".

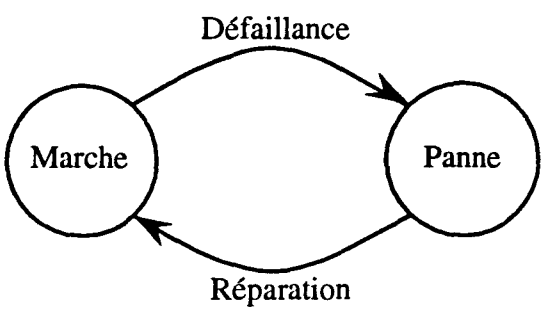


FIGURE I.3 : DISPONIBILITÉ D'UN COMPOSANT.

Un système réparable oscille entre ces deux états de marche et panne. Les événements constituant les transitions sont : défaillance (de marche à panne) et réparation (de panne à marche). Nous avons donc un système dont le comportement est cohérent. La disponibilité d'un composant peut être présentée comme étant la conjonction de la fiabilité et de la maintenabilité (FIGURE I.3).

Ce modèle regroupant les deux aspects fiabilité et maintenabilité, donne une vision globale de l'état du système. Ce modèle est nécessaire à la supervision.

Après avoir défini les différentes notions se rattachant à la sûreté de fonctionnement (nous n'avons pas traité de la sécurité par souci d'allégement, l'utilisateur en trouvera une définition dans le glossaire), nous abordons la pathologie des fautes ainsi que les méthodes qui sont mises en œuvre pour remédier à ces défaillances.

I.3. PATHOLOGIE DES FAUTES.

I.3.1. Généralités.

L'écart "*non-toléré*" qui sera constaté entre le fonctionnement réel et le fonctionnement attendu ou spécifié, permet de caractériser l'état d'un système défaillant. Le terme "non-toléré" met en évidence, le fait que pour un fonctionnement normal, il peut y avoir des écarts négligeables tolérés (perte de qualité de production). C'est seulement quand ces écarts deviennent assez importants, que l'on considère le système comme défaillant. "*La défaillance est la fin de l'aptitude d'un dispositif à accomplir sa fonction requise*" [CHAPOUILLE, 68].

Une défaillance survient quand le système est erroné. Une défaillance est donc la conséquence directe d'une erreur. Une erreur apparaît suite à une faute qui en est la cause. Nous avons donc une succession logique d'événements qui met le système dans un état défaillant : faute, erreur et défaillance. Nous allons nous intéresser à ces trois notions. D'autres notions rattachées à celles-ci seront aussi abordées comme la notion de défaut et de panne.

I.3.2. Faute.

On peut définir de manière classique deux classes de fautes [AVIZIENIS, 78], qui sont de nature *physique* ou *humaine* (FIGURE I.4).

Les fautes physiques sont dues à des perturbations internes ou externes au système. Les fautes humaines, quant à elles, sont inhérentes à l'imperfection humaine. Ces dernières peuvent avoir deux origines. On trouve d'une part les fautes de conception, liées au cycle de vie d'un produit, depuis sa spécification jusqu'à sa phase d'exploitation ; d'autre part, les

fautes d'interaction liées à l'exploitation du système : demande de service inadéquat ou inexistant, violation du fonctionnement normal du système.

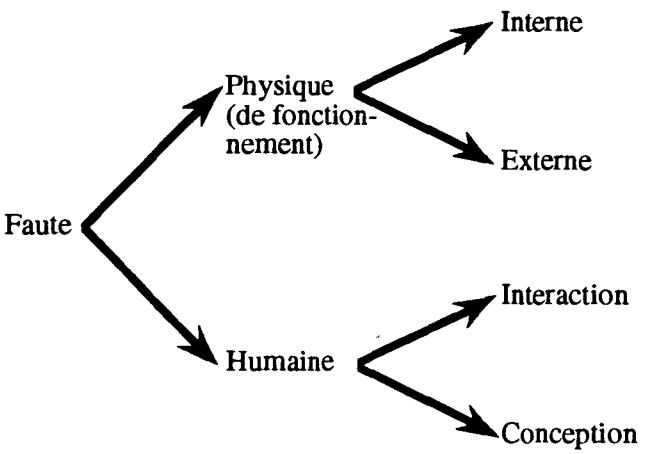


FIGURE I.4 : CLASSIFICATION DES FAUTES.

I.3.3. Erreur.

La faute est la cause directe d'une *erreur*. Une faute de conception, par exemple, amène une erreur *latente*. Cette erreur deviendra *effective* quand une procédure d'exploitation fera intervenir la faute qui en est la cause. Une faute d'interaction engendre un état erroné du système. Une erreur peut alterner entre ses états, latent et effectif (FIGURE I.5) [LAPRIE, 85].

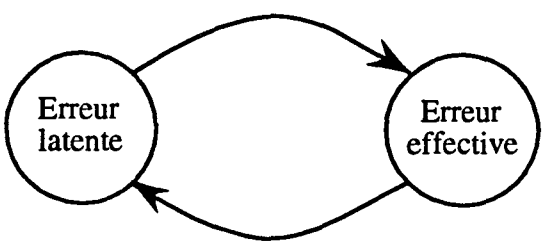


FIGURE I.5 : ETATS D'UNE ERREUR.

Hormis, les erreurs latentes, on trouve des *erreurs de propagation*. Ainsi, une erreur d'un composant peut conduire un autre composant indépendant dans un état erroné. Une erreur effective peut donc avoir deux origines : l'activation d'une erreur latente ou la propagation d'une autre erreur effective.

I.3.4. Défaillance.

Quand une erreur affecte la réalisation d'une mission, une *défaillance* survient. L'apparition d'une défaillance ne pourrait en aucun cas être vue comme la transition séparant l'état latent de l'état effectif d'une erreur.

Une défaillance peut se produire de plusieurs manières : progressive, soudaine, partielle, ou complète [CHAPOUILLE, 68]. Ces défaillances sont classifiées selon leur nature [VILLEMEUR, 88], [CHAPOUILLE, 68] (FIGURE I.6).

- **Classification en fonction de la rapidité de leur manifestation.**

On distingue pour cette classe deux types de défaillances [CHAPOUILLE, 68] :

- la *défaillance progressive*, est une défaillance prévisible par un examen antérieur des caractéristiques de l'élément défaillant,
- la *défaillance soudaine*, est une défaillance imprévisible par un examen antérieur des caractéristiques de l'élément défaillant.

- **Classification en fonction de leur amplitude.**

On distingue pour cette classe deux types de défaillances [CEI, 74] :

- la *défaillance partielle*, est une défaillance résultant de déviations d'une ou des caractéristiques au-delà des limites spécifiées mais telles qu'elles n'entraînent pas une disparition complète de la fonction requise,
- la *défaillance complète*, est une défaillance résultant de déviations d'une ou des caractéristiques, telles qu'elles entraînent une disparition de la fonction requise.

- **Classification en fonction de la rapidité de leur manifestation et de leur amplitude.**

Ces deux points de vues (rapidité de manifestation et amplitude) sont liés et peuvent être combinés dans une même défaillance. Les combinaisons de ces deux aspects, donnent naissance au regroupement suivant [VILLEMEUR, 88] :

- la *défaillance cataleptique* qui est à la fois soudaine et complète,
- la *défaillance par dégradation* qui est à la fois progressive et partielle.

On trouve aussi dans la littérature, ce qu'on appelle la *défaillance d'exploitation*. Elle est causée par le non-respect des consignes d'exploitation ou le dépassement des limites d'exploitation du composant considéré. Nous pouvons citer à titre d'exemple, la localisation dans le système de production d'éléments extérieurs perturbateurs, ou encore, le dépassement du seuil d'usure d'un composant.

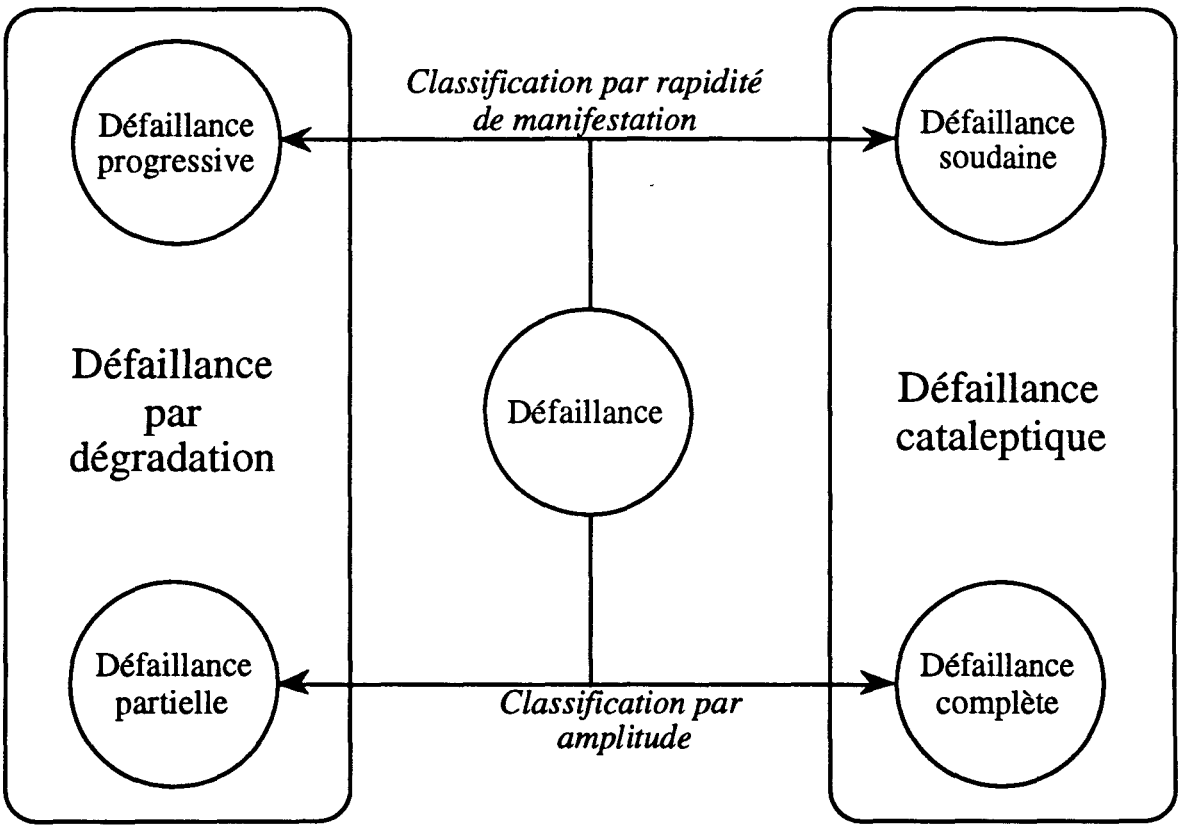


FIGURE I.6 : DIFFÉRENTS TYPES DE DÉFAILLANCES.

I.3.5. Défaut, Panne et Défaillance.

Le *défaut* est défini par VILLEMEUR [88] comme étant "*un écart entre une caractéristique d'un composant et la caractéristique voulue*". Il est considéré comme une non-conformité à des clauses de spécification. Un défaut du système n'affecte pas en général l'aptitude du système à accomplir une fonction requise. Il ne conduit donc pas nécessairement à une défaillance. Par contre, une défaillance conduit inévitablement à un défaut, puisque son inaptitude à remplir sa fonction est liée à l'écart d'une caractéristique du composant. L'état de panne d'un composant est une conséquence directe de la défaillance. En effet, la panne est considérée comme *l'inaptitude du composant à accomplir une fonction requise*. L'apparition de la panne dépend du type de la défaillance.

I.3.5.1. Le cas de la défaillance cataleptique.

Ce type de défaillance conduit irrémédiablement à un arrêt du composant (FIGURE I.7). Le composant est considéré en état de panne, dès la manifestation de la défaillance.

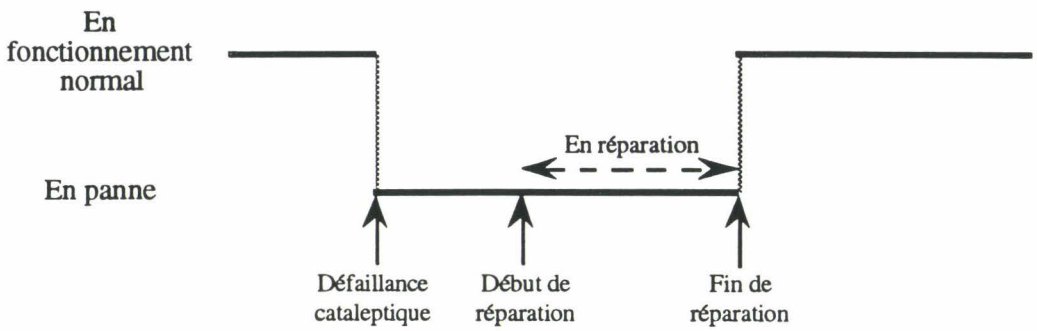


FIGURE I.7 : CONSÉQUENCES D'UNE DÉFAILLANCE CATALEPTIQUE.

I.3.5.2. Le cas de la défaillance par dégradation.

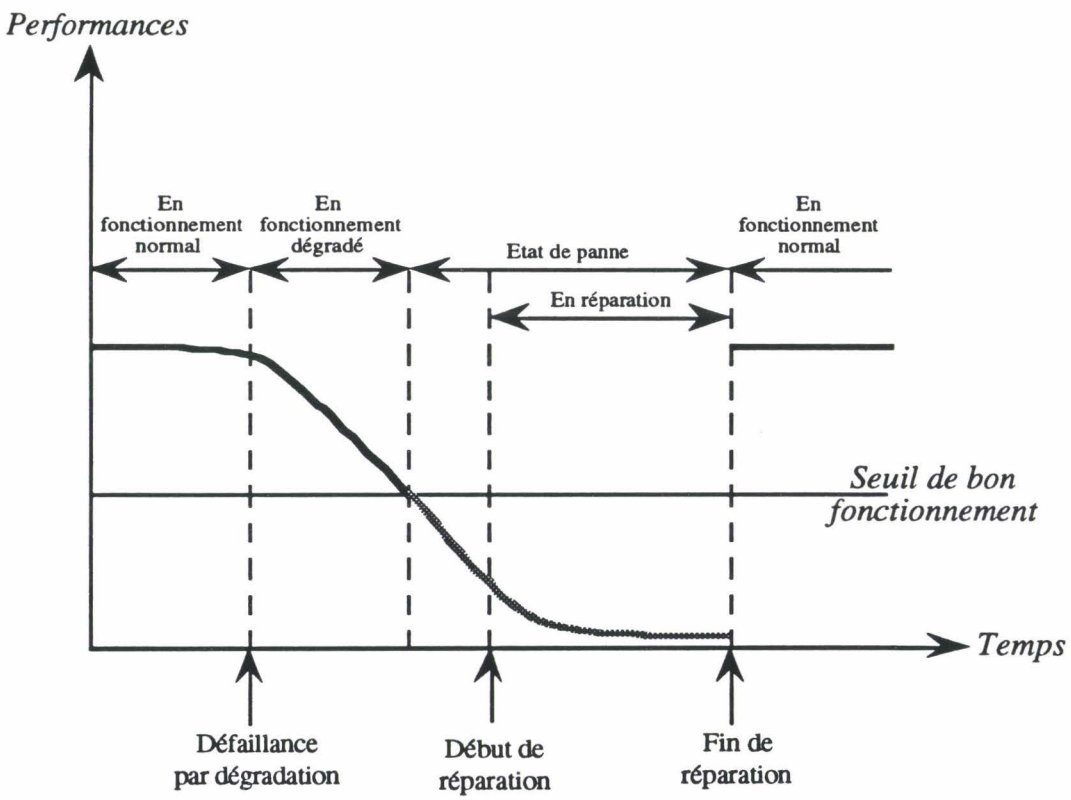


FIGURE I.8 : CONSÉQUENCES D'UNE DÉFAILLANCE PAR DÉGRADATION.

Ce type de défaillance conduit à une dégradation du fonctionnement (FIGURE I.8). Le composant défaillant sera considéré en panne dès que les performances sont en deçà du seuil.

Sur le graphique de la FIGURE I.8, on distingue trois états de fonctionnement pour tout système : "en fonctionnement normal", "en fonctionnement dégradé", et "en panne".

Le passage d'un fonctionnement normal à un fonctionnement dégradé se fait suite à une défaillance. L'état de panne est caractérisé par une perte importante des caractéristiques de fonctionnement, qui conduit inévitablement à une interruption du service (arrêt en position de repli, arrêt d'urgence, ...).

I.3.6. Conclusion.

En résumé, une défaillance survient parce que le système est erroné : l'erreur est la cause directe d'une défaillance. La cause phénoménologique d'une erreur est une faute [SAHRAOUI, 87]. Nous avons donc une suite logique d'événements :

Faute → Erreur → Défaillance.

La défaillance conduit inévitablement à un défaut. La classification de la défaillance en deux catégories (par dégradation et cataleptique) permet de mettre en évidence les conséquences sur le fonctionnement de l'élément défaillant. Ainsi, la défaillance cataleptique conduit immédiatement à une panne et donc à un arrêt instantané du fonctionnement. Ce type de défaillances imprévisibles, est très difficile à anticiper. Par contre, dans le cadre d'une défaillance par dégradation, la réparation du composant peut être anticipée afin d'éviter la panne par la mise en place d'une maintenance sur état (cf. §II.2.2.).

I.4. ANALYSE PRÉVISIONNELLE DE LA SÛRETÉ DE FONCTIONNEMENT.

Les méthodes d'analyse de la sûreté de fonctionnement sont basées sur un modèle du système. Pour élaborer ce modèle, il faut définir les points d'entrées et de sorties : interactions avec l'environnement. Les informations nécessaires sont différentes selon la méthode adoptée. Il est donc nécessaire de définir les principales caractéristiques, à prendre en compte. Ces méthodes constituent la base d'une étude prévisionnelle. Ces méthodes ont été adaptées dans plusieurs domaines comme la tolérance aux fautes des systèmes

informatiques [ARLAT, 88] et la conception de la commande numérique des systèmes électromagnétiques [AUBRY, 91].

I.4.1. Méthodes d'analyse de la sûreté de fonctionnement.

Plusieurs méthodes ont été mises au point, pour des besoins et dans des contextes différents. Nous nous contenterons ici de citer les principales méthodes, le lecteur trouvera une étude détaillée de chacune d'elles dans [VILLEMEUR, 88] :

- l'Analyse Préliminaire des Dangers (APD),
- l'Analyse des Modes de Défaillances et de leurs Effets (AMDE),
- la Méthode du Diagramme de Succès (MDS),
- la Méthode de la Table de Vérité (MTV),
- la Méthode de l'Arbre des Causes (MAC),
- la Méthode des Combinaisons de Pannes Résumées (MCPR),
- la Méthode de l'Arbre des Conséquences (MACQ),
- la Méthode du Diagramme Causes Conséquences (MDCC),
- la Méthode de l'Espace des Etats (MEE).

Ces méthodes peuvent être classées selon deux types de raisonnement :

- *inductive* : le raisonnement se fait du plus particulier au plus général. On réalise une étude détaillée des effets d'une défaillance sur le système défaillant et son environnement. Les principales méthodes sont l'AMDE, la MTV, MCPR et la MACQ,

- *déductive* : le raisonnement est inverse. Le but est de rechercher les causes des défaillances. La principale méthode est la MAC.

I.4.2. Principales étapes de l'analyse prévisionnelle.

L'étude prévisionnelle des caractéristiques de la sûreté repose sur les méthodes exposées ci-dessus. Elle est décomposée habituellement en quatre étapes.

I.4.2.1. Analyse technique et fonctionnelle.

Elle s'intéresse à la nature des informations à prendre en compte (informations sur les composants constituant le système). C'est une étape préliminaire de l'analyse qualitative.

I.4.2.2. Analyse qualitative.

Elle est subdivisée en plusieurs étapes :

- *Objectifs de l'analyse* : quelle est la caractéristique de la sûreté qui sera étudiée ?
- *Définition des limites* : quel est le niveau et le critère de décomposition du système ?
- *Décomposition du système* : le système est décomposé en composants suivant le critère et le niveau de définition choisis. On s'intéresse aussi, aux informations relatives aux modes de défaillances et à leurs causes,
- *Choix d'une méthode d'analyse et application* : quelle est la méthode la plus adaptée aux objectifs, au système à analyser et aux moyens disponibles,
- *Modélisation de la sûreté de fonctionnement* : cette étape est très importante, puisqu'elle permet d'obtenir les *défaillances pertinentes* à prendre en compte en fonction des étapes précédentes.

L'analyse qualitative permet de fixer la caractéristique étudiée (fiabilité, disponibilité, ...), d'obtenir une décomposition du système (selon les critères et niveau choisis), de choisir une méthode d'analyse et de la mettre en œuvre. Elle aboutit ainsi à un recensement des défaillances pertinentes menant à des situations indésirables.

I.4.2.3. Analyse quantitative.

Elle consiste en l'évaluation des mesures (probabilités) de la sûreté de fonctionnement. Cette évaluation est basée sur les données relatives aux événements élémentaires, les durées de fonctionnement (durée d'usinage, de transport, ...), les durées de maintenance et les données statistiques (fréquence) sur les événements indésirables. Ces mesures sont calculées avec des marges d'incertitudes. Cette étape permet d'identifier et d'évaluer les points faibles du système, les composants critiques, ...

I.4.2.4. Synthèse.

La synthèse des deux analyses (qualitative et quantitative) permet d'identifier les défaillances qui compromettent le fonctionnement du système et les composants critiques. Il est ainsi possible d'apporter des améliorations pour pallier ces insuffisances. Dans le cas où, ces constats ne répondent pas aux exigences de la sûreté de fonctionnement, des

propositions peuvent être faites. Ainsi, un composant critique sera dupliqué pour éviter des arrêts inopportuns du système et accroître la disponibilité matérielle : *redondance complémentaire* ou *modification de la redondance.* Ou bien, une machine qui compromet le fonctionnement, sera remplacée par une autre machine de même type mais plus fiable : *amélioration de la fiabilité du système.* Un autre avantage de cette étude permet la mise en place d'un calendrier de la *maintenance préventive.*

I.5. SÛRETÉ DE FONCTIONNEMENT ET SURVEILLANCE.

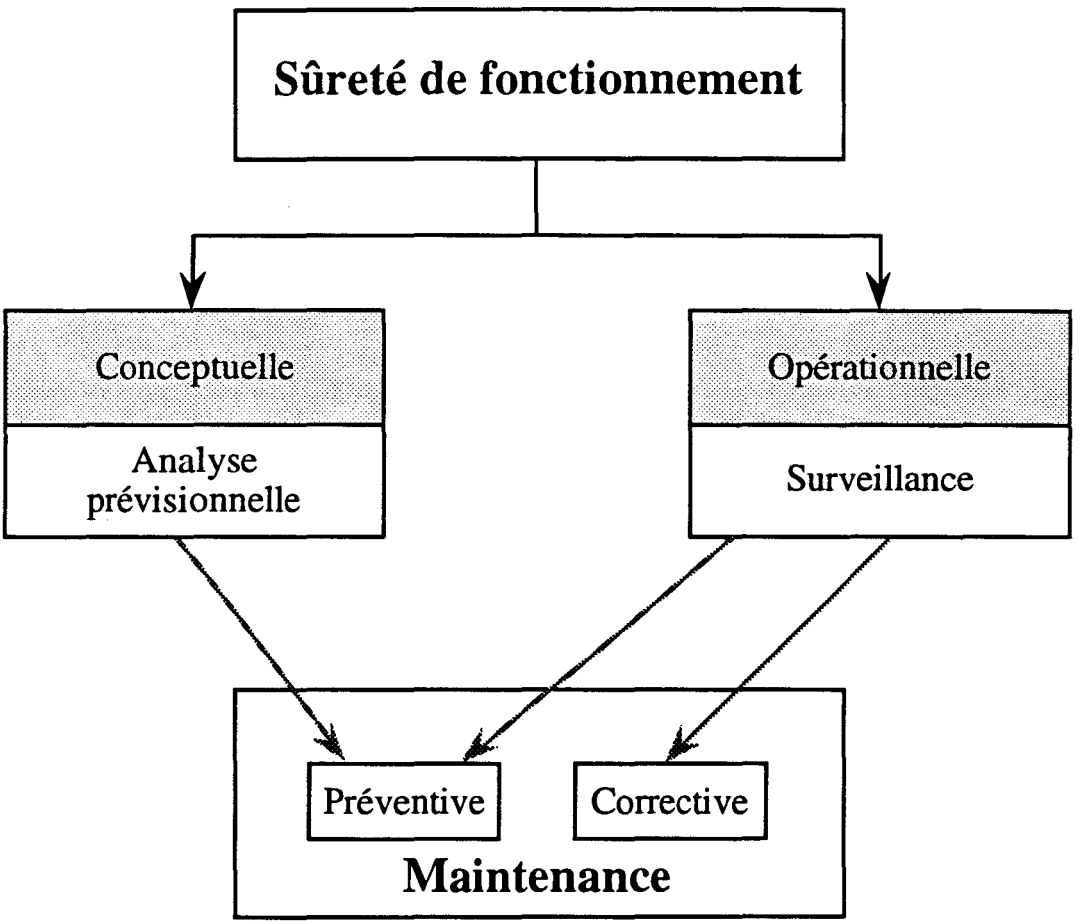


FIGURE I.9 : SÛRETÉ DE FONCTIONNEMENT ET SURVEILLANCE.

Au niveau de la conception d'un système de production, les méthodes d'analyse sont très appréciées et permettent une amélioration des caractéristiques du système. Par contre, elles sont inadaptées en phase d'exploitation et leur utilisation possible serait très pénalisante. En effet, il n'est pas concevable de réitérer l'analyse prévisionnelle durant la phase d'exploitation, pour introduire de la redondance ou pour remplacer un composant critique.

Ce qui nous amène à séparer les deux aspects : conception et exploitation (FIGURE I.9). Cette séparation de la sûreté, nous permet d'introduire la notion de la *sûreté de fonctionnement "opérationnelle"* appelée *surveillance* [TOGUYENI, 92]. En effet, la surveillance permet d'améliorer et de garantir un certain nombre de caractéristiques du système en phase d'exploitation, par une surveillance du procédé physique. Il s'agit en l'occurrence de :

- la fiabilité : détecter des erreurs, les traiter, ...
- la maintenabilité : activer la maintenance corrective et améliorer la maintenance préventive,
- la disponibilité : améliorer de la fiabilité et de la maintenabilité,
- la sécurité du système : éviter les risques humains et de dégradation des équipements par des arrêts d'urgence.

Nous verrons au §II.2 de ce chapitre, l'incidence de la sûreté de fonctionnement sur la maintenance.

I.6. CONCLUSION.

Les caractéristiques de la sûreté de fonctionnement peuvent être étudiées sur deux plans : qualitatif et quantitatif. L'aspect quantitatif a été délibérément écarté. Le lecteur trouvera plus de détails dans [VILLEMEUR, 88] et [PAGÈS, 80].

Le concepteur des systèmes de production peut avoir à automatiser un atelier déjà existant. Dans ce cas, seule la surveillance peut être proposée, néanmoins, une étude prévisionnelle de la sûreté permettra de déterminer les situations critiques, afin de les prendre en compte dans l'élaboration de la surveillance. Dans le cas inverse, la conception d'une architecture matérielle du système de production doit intégrer la sûreté de fonctionnement "*conceptuelle*" et la sûreté de fonctionnement *"opérationnelle"*.

II. SUPERVISION, MAINTENANCE ET SURVEILLANCE.

La surveillance n'est pas un concept nouveau. Il est apparu très tôt avec le développement des premières centrales nucléaires. Dans ce contexte, l'objectif de la surveillance était d'assurer la sécurité. Ce concept a été largement repris par le monde industriel sous le terme de supervision. Les systèmes de supervision sont généralement

caractérisés par les fonctions suivantes : acquisition des données, traitement des données, gestion d'alarmes, et présentation de synoptiques de conduite ou de surveillance.

Dans ce contexte, l'opérateur joue un rôle essentiel. Il assure le bouclage entre les fonctions de surveillance et de conduite. C'est notamment, lui, qui prend toutes les décisions faisant appel à l'intelligence.

II.1. SUPERVISION ET SURVEILLANCE.

L'automatisation des systèmes de production nécessite l'introduction d'un niveau supérieur afin d'optimiser la production, le fonctionnement et la sécurité : la *supervision.* Deux fonctions sont assurées par ce niveau : la *conduite* et la *surveillance.* La coordination a pour fonction de coordonner l'ensemble des sous-systèmes composant le procédé industriel. Actuellement, elle est automatisée et évite l'intervention de l'opérateur.

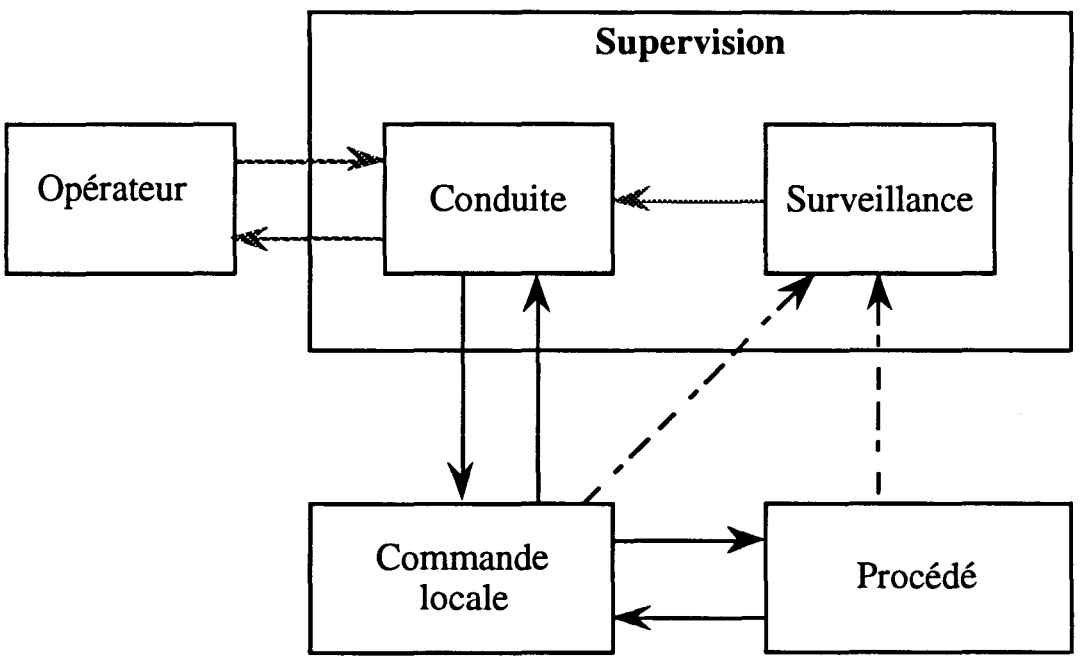


FIGURE I.10 : ORGANISATION D'UN SYSTÈME DE SUPERVISION.

Les systèmes de supervision qui existent actuellement souffrent d'un manque de flexibilité, qui résulte du fait que seul le comportement normal est décrit. Dès l'apparition d'un écart du fonctionnement décrit, le système de commande se trouve, bloqué et ne peut plus évoluer. La solution à ce problème passe par l'*intégration d'un système de surveillance.*

Les fonctions d'un système de *supervision* seront donc la conduite et la surveillance [MILLOT, 88]. Ces deux fonctions doivent être séparées mais elles doivent dialoguer pour des prises de décision (FIGURE I.10). En effet, le système de supervision doit fournir à tout instant l'état du système pour faciliter les interventions de l'opérateur, en cas d'écart du fonctionnement normal. Ce qui associe systèmes de conduite et de surveillance.

Actuellement, la supervision se concrétise par des postes de conduite qui centralisent les informations issues des capteurs, n'assurant ainsi que la fonction conduite. L'intervention humaine n'est généralement pas requise pour un fonctionnement normal du système de production. Cependant, en présence de situations de dysfonctionnement imprévues ou inconnues par le système automatisé, la décision doit être assurée par l'opérateur. Pour éviter les erreurs humaines dans les tâches de décision, les recherches actuelles visent à intégrer les systèmes de surveillance, et des outils d'assistance aux opérateurs [MILLOT, 88]. Afin de décentraliser les interventions, les systèmes de commande doivent intégrer les systèmes de surveillance (FIGURE I.11).

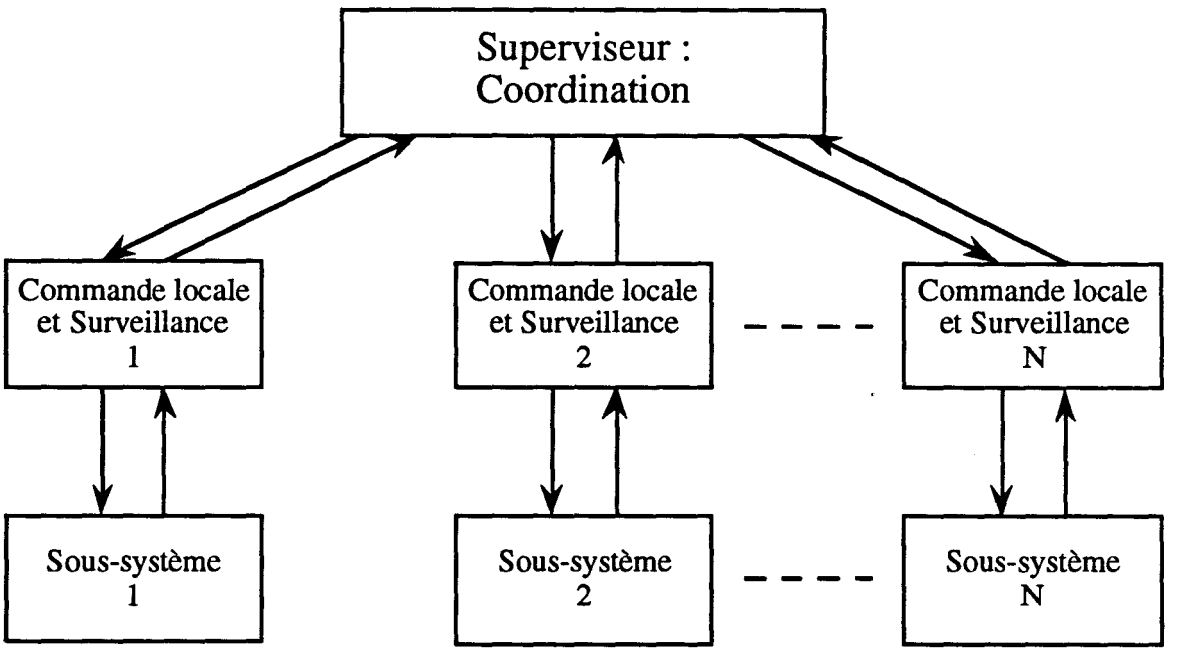


FIGURE I.11 : SYSTÈME DE SUPERVISION DE SHERIDAN.

Un dysfonctionnement donne naissance à une défaillance. En réalité, les défaillances sont provoquées par une combinaison d'événements ou de dysfonctionnements, ce qui complique le diagnostic et la reprise puisqu'une défaillance peut avoir plusieurs causes. A l'inverse, une défaillance peut générer d'autres défaillances sur des composants inter-connectés : propagation des erreurs. De plus, l'ordre d'apparition est peu significatif pour la

localisation de la cause initiale du défaut, si on tient compte des délais et des temps de réponses des équipements.

Un autre problème concerne l'apparition *simultanée* des défaillances, sur des sous-systèmes disjoints. La prise en compte de l'état de fonctionnement, rend généralement le traitement d'erreurs séquentiel. Il est donc nécessaire de définir un ordre d'intervention en fonction de l'urgence et de la gravité de chaque défaillance.

ESCORT [SACKS, 86] répond en partie à ces besoins. Il propose des solutions vraisemblables et ordonnées suivant un critère de crédibilité et de gravité. Il est construit autour de règles de production établies à partir de connaissances fonctionnelles. Ces connaissances définissent les liens directs entre les défaillances et les causes. Néanmoins, ce genre d'outils d'aide à la surveillance de procédés, se heurte au *besoin de la rapidité du temps de réponse* (inférieure à une seconde) [MILLOT, 88]. Leur utilisation dans les systèmes complexes est ainsi affaiblie.

La mise en place des systèmes de contrôle "intelligents" est confrontée à deux problèmes importants d'ordre méthodologiques. En effet, ils nécessitent l'utilisation des modèles temporels, pour répondre au besoin de la représentation dynamique du procédé à surveiller [MCDERMOTT, 82] et d'un modèle fonctionnel du procédé pour détecter les déviations par rapport au fonctionnement nominal [MILLOT, 88]. Ce dernier modèle a une double vocation, premièrement, détecter les causes d'anomalies et deuxièmement, prédire l'évolution comportementale du procédé (cf CHAPITRE VI).

Pour les SED, la nécessité de la séparation de la conduite et de la surveillance a été établie depuis longtemps. La conduite étant généralement automatisée, il devrait en être de même pour la surveillance. De plus, le concept d'atelier flexible a fixé les limites d'une telle approche. Notamment, on peut citer, l'interaction entre le SC et le procédé; la flexibilité de production; et la flexibilité des ressources. A cet effet, un nouveau concept a été proposé : la surveillance en ligne [SAHRAOUI, 87 ET 92]. Dans ce mémoire, nous présenterons une approche qui permet de répondre à ces besoins. Nous pensons que la hiérarchisation et la répartition des modules de surveillance s'imposent.

Après avoir positionné la supervision et la conduite par rapport à la surveillance, nous allons présenter la maintenance et ses liens avec la surveillance.

II.2. MAINTENANCE ET SURVEILLANCE.

La conception de la maintenance doit être menée en parallèle avec celle des autres fonctions du système (commande, procédé physique, fiabilité, ...). En phase d'exploitation, la sûreté de fonctionnement d'un système est accrue par la maintenance. La maintenance consiste à mettre en œuvre un certain nombre d'actions techniques sur site, pour maintenir ou remettre une entité dans un état lui permettant d'accomplir sa fonction requise. En général, la surveillance continue et le contrôle (examen systématique) d'un système sont considérés comme des opérations de maintenance.

La maintenance peut prendre plusieurs formes et intervenir à des moments différents. En effet, on dénombre généralement deux types de maintenance [VILLEMEUR, 88].

II.2.1. Maintenance corrective.

La *maintenance corrective* (Norme AFNOR 60.010) *consiste à réparer les conséquences d'un dysfonctionnement, devenues irréversibles.*

Les défaillances cataleptiques sont considérées comme irréversibles puisqu'elles sont imprévisibles et totales. La maintenance corrective est donc effectuée suite à la détection d'une défaillance cataleptique (voir la connexion entre surveillance et maintenance corrective sur la FIGURE I.9) et consiste à remettre l'entité défaillante dans un état de fonctionnement apte à accomplir la fonction requise. Son objectif principal est de minimiser le temps moyen d'indisponibilité (**MDT** ou **M**ean **D**own **T**ime) de l'entité.

Dans ce contexte, dès l'apparition d'une défaillance, le système est arrêté. Il faut réparer le système le plus vite possible (rapidité d'intervention, rapidité de réparation). La prévision de ces défaillances est quasi impossible vu leur rapidité d'apparition (soudaine) et leur amplitude (complète).

II.2.2. Maintenance préventive.

La *maintenance préventive* (Norme AFNOR 60.010) ***répond au souci de conserver un potentiel d'heures de fonctionnement sans panne, grâce au remplacement périodique d'organes ou de fluides sujets à usure régulière (l'intervalle de temps entre deux opérations d'entretien étant le fruit de l'expérience).***

Ce concept de maintenance correspond à une méconnaissance des phénomènes de dégradation et de rupture de matériaux [SCHOENAUER, 90]. Cette maintenance est effectuée à des intervalles prédéterminés afin de réduire la fréquence des défaillances et la dégradation du fonctionnement d'une entité. D'un point de vue pratique, la probabilité de défaillance est variable, étant donné les incertitudes sur les lois utilisées en maintenance (FIGURE I.12).

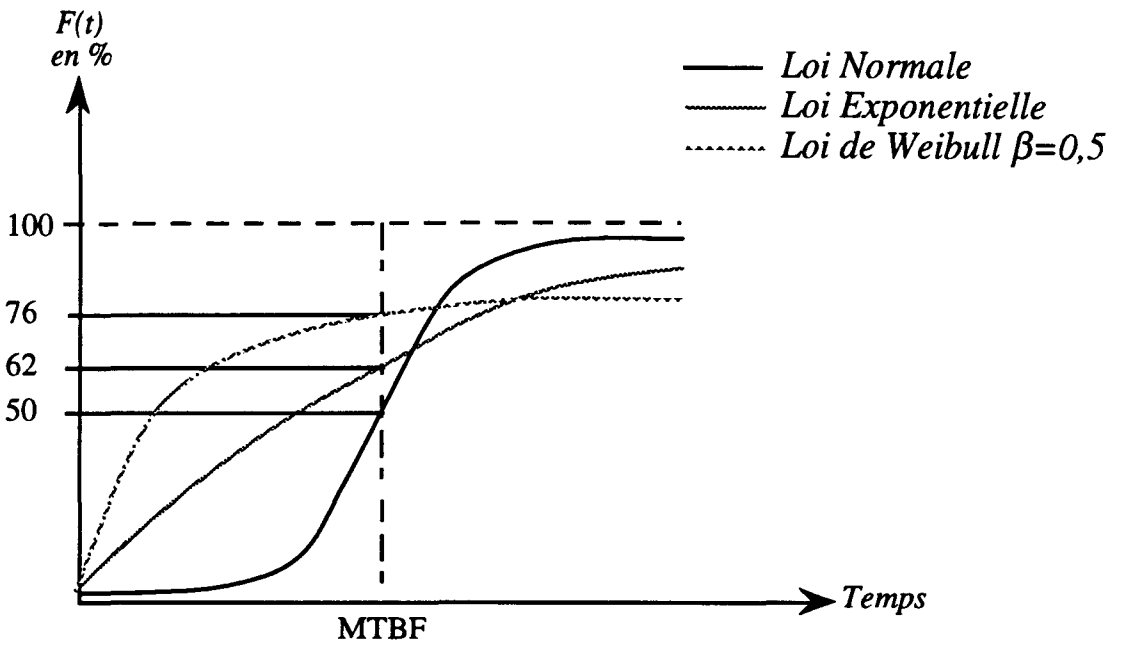


FIGURE I.12 : VARIATION DE LA PROBABILITÉ DE DÉFAILLANCE POUR TROIS MODÈLES DE DURÉE DE VIE DE MÊME VALEUR MOYENNE [OGUS, 90].

La maintenance préventive peut être programmée (respect de calendrier) ou non (non-respect du calendrier, suite à une indication sur l'état d'une entité). Le calendrier est établi à partir des paramètres comme la durée moyenne de fonctionnement d'une entité avant la première défaillance (**MTTF** ou **M**ean **T**ime **T**o **F**ailure) et la durée moyenne entre deux défaillances consécutives d'une entité réparée (**MTBF** ou **M**ean **T**ime **B**etween **F**ailure). Les entrées d'une telle maintenance sont données sur la FIGURE I.9.

Cette technique s'adapte parfaitement aux défaillances par dégradation (FIGURE I.8). En effet, ces défaillances concernent essentiellement l'usure et le vieillissement de l'équipement. Cependant, la maintenance préventive présente deux inconvénients majeurs :

- remplacement d'un composant avant usure "réelle",
- non-remplacement d'un composant à usure prématurée.

Ce type de maintenance est assez coûteux. Suite à ces deux inconvénients, Brunet préconise la "*maintenance selon état*". Ce type de maintenance permet d'éviter les deux inconvénients cités, s'appuyant sur : "*la prise en compte de l'état actuel du système et des équipements associés [...], minimisant ainsi la perte de disponibilité de l'outil de production*" [BRUNET, 90]. Le remplacement d'un composant sera donc effectué en fonction de son état réel (utilisation d'indicateurs sérieux d'usure ou de fatigue). Le graphique de la FIGURE I.8, nous permet d'affirmer qu'il est souhaitable d'intervenir en phase de fonctionnement dégradé. On évite ainsi le passage de la courbe des performances en dessous du seuil et la mise en panne du composant considéré. Brunet préconise donc la *maintenance sur état* qui regroupe la maintenance prédictive et la maintenance corrective [BRUNET, 90].

CONCLUSION.

Nous venons de montrer l'intérêt de surveiller un SA, et de traiter les défaillances mettant la production, les équipements ou les hommes en situation dangereuse. Les défaillances cataleptiques, de nature imprévisibles, conduisent à un arrêt instantané du fonctionnement et seule la maintenance préventive peut être envisagée dans ce cas. Par contre, dans le cas des défaillances par dégradation, la mise en état peut être anticipée par une maintenance sur état. Il est donc nécessaire de mettre en œuvre des procédures de maintenance afin de prévenir les défaillances, d'en diminuer les fréquences et les risques, et de prolonger la durée de vie du système. Pour augmenter l'efficacité de la maintenance, tout en réduisant les coûts de production, il est nécessaire d'avoir une bonne connaissance des phénomènes d'usure et de dégradation des matériaux.

La surveillance doit être développée dans le cadre d'une politique générale de la maintenance. Cette surveillance ne doit pas être conçue a posteriori, mais de front avec le SC. Il sera très utile de réaliser une étude prévisionnelle de la sûreté de fonctionnement dans le cadre de la définition d'une architecture matérielle ou existante. En effet, une telle étude permet de déterminer les défaillances et composants critiques qui compromettent le fonctionnement du système. Il sera ainsi possible d'apporter des solutions afin d'améliorer les caractéristiques du système. Les solutions envisageables sont plus restreintes dans le cas d'un procédé existant. Un calendrier de la maintenance préventive est dressé à partir de cette étude.

CHAPITRE II.

VERS UNE APPROCHE DE LA SURVEILLANCE.

INTRODUCTION.

Ce chapitre a pour objectif de proposer une démarche de conception de la surveillance des Systèmes à Evénements Discrets (SED) appliquée aux Systèmes Flexibles de Production Manufacturière (SFPM).

Les différentes composantes et fonctionnalités d'un système de surveillance seront présentées, ainsi que la partie du système à surveiller.

Ensuite, nous nous intéressons aux principales approches développées, actuellement. Il s'agit des travaux réalisés au LAAS (Laboratoire d'Automatique et d'Analyse des Systèmes), au CRAN (Centre de Recherche d'Automatique de Nancy), et au LAIL (Laboratoire d'Automatique et d'Informatique Industrielle de Lille).

Nous terminerons ce chapitre par la définiron du contexte et des hypothèses de travail, la description des systèmes que nous désirons surveiller, ainsi que la proposition d'une approche de la surveillance pour les SFPM automatisés et son positionnement par rapport aux approches proposées par d'autres laboratoires.

I. LA SURVEILLANCE EN LIGNE.

La surveillance en ligne a pour but de surveiller en permanence un **SA**, pendant son fonctionnement. Nous allons nous intéresser en particulier à la surveillance en ligne intégrée à la commande des ateliers flexibles.

La surveillance doit être conçue à des fins d'amélioration des caractéristiques de la sûreté de fonctionnement mais aussi dans le cadre d'une politique générale de maintenance. Il s'agit en l'occurrence de la fiabilité, la disponibilité et la sécurité des moyens de production. La démarche de la surveillance proposée s'inscrit dans le cadre d'une maintenance corrective et préventive. L'objectif sous-jacent est l'identification réactive des composants défaillants (maintenance corrective), mais aussi la prévision des pannes afin de réduire le MDT et d'assurer la disponibilité (maintenance préventive).

Notre approche étant de nature discrète, l'observation du procédé est assurée par des signaux discrets. Ce contexte discret engendre la disparition de quelques fonctions de surveillance [BRUNET, 90]. La contrainte "temps réel", induite par l'aspect en ligne, implique le regroupement de quelques fonctions. Les principales fonctions de la surveillance en ligne, sont [SAHRAOUI, 87] :

- l'acquisition des données de la commande de synchronisation et du procédé,
- la détection de défaillances,
- le diagnostic,
- le recouvrement de défaillances.

Nous reprenons dans le paragraphe suivant en détail les différentes fonctions énumérées ci-dessus.

II. FONCTIONNALITÉS D'UN SYSTÈME DE SURVEILLANCE.

La surveillance d'un système automatisé a pour objectif de rendre le système de production sûr de fonctionnement. Pour ce faire, un système de surveillance doit avoir les fonctions suivantes [BRUNET, 90] :

- *percevoir* et *détecter* les défaillances,
- *localiser* et *diagnostiquer* leurs causes,
- mettre en œuvre des procédures de *traitement*.

Le système de surveillance doit être décomposé afin de faciliter sa conception. Cette approche modulaire permet de mieux appréhender le fonctionnement général du système. Un module aura pour charge d'assurer une ou plusieurs fonctions.

La *détection* a pour objectif de relever tout écart du fonctionnement normal. Cette fonction peut nécessiter l'utilisation de sources d'informations supplémentaires (capteurs de surveillance, redondance logicielle, ...). Suite à une défaillance, on procède à une *localisation* du composant défaillant du système. Cette fonction peut être assurée par le module de détection.

L'étape suivante consiste à *identifier* l'origine de la défaillance en fonction des informations disponibles. Il s'agit d'un système de *diagnostic.*

A ce stade (après détection, identification et diagnostic), il faut *traiter* la défaillance. Ce traitement consiste généralement à élaborer un traitement correctif et la reprise du fonctionnement normal.

Hormis ces fonctions, un système de surveillance doit être capable de mettre en œuvre des *procédures d'urgence* répondant à des critères de sécurité. Nous détaillons par la suite le fonctionnement de chacun de ces modules et les liens entre eux.

II.1. DÉTECTION DES ERREURS.

Puisque la construction d'un système parfait est irréaliste, il faut prévoir et s'attendre à l'apparition de défaillances pendant son fonctionnement. Il est donc nécessaire de détecter les erreurs responsables de ces défaillances. Plusieurs études et méthodes ont été proposées dans ce sens. Elles reposent sur deux approches différentes : détection *directe* ou *indirecte.* La détection directe est basée principalement sur les informations récoltées auprès des capteurs utilisés à cet effet (capteurs de commande et de surveillance). La détection indirecte consiste à interpréter des signaux prétraités. Cette approche est basée sur le concept de déduction.

Cette phase a donc pour objectif de décider si le système *est* ou *n'est pas* dans un état structurel normal [BRUNET, 90]. C'est une opération logique dont la réponse est *binaire.* Il faut donc définir à partir de l'observabilité du système l'ensemble des situations normales et anormales. Il serait illusoire de vouloir décrire toutes les situations anormales puisqu'il est

impossible de prévoir toutes les défaillances possibles à l'avance : "*tout ce qui ne sera pas reconnu comme normal sera considéré comme anormal*". M. COMBACAU définit deux ensembles de comportements normaux : états utilisables (vu par le procédé) et états autorisés par la commande [COMBACAU, 91]. Le deuxième est un sous-ensemble du premier.

La fonction détection doit être capable de reconnaître les requêtes non compatibles avec l'état du système physique (filtrage) et les écarts de fonctionnement (contrôle). Le filtrage des requêtes consiste à éviter la réalisation des consignes incohérentes et de mettre le procédé dans un état erroné. Quant au contrôle du fonctionnement, il permet d'éviter une propagation de l'erreur à condition que la détection soit réactive. La détection réactive permet de détecter la défaillance dès son apparition et de localiser aussitôt le composant défaillant.

Nous devons de plus considérer les défaillances qui peuvent affecter les organes de détection : capteur, détecteur d'erreur, ... Il est donc souhaitable, voire même nécessaire, de prémunir le système de redondance matérielle de ces organes.

II.2. IDENTIFICATION ET DIAGNOSTIC.

La localisation *attribue* le défaut détecté à l'organe défaillant (capteur, ressource, commande, ...). Le diagnostic est une opération de classification qui caractérise le défaut par type et degré de sévérité. Elle est adaptée aux défauts et pannes d'usure. Elle implique une estimation de la cause et une quantification du degré de confiance sur le diagnostic émis [BRUNET, 90].

Le diagnostic permet de rechercher les causes d'une défaillance. Dans le cadre des SFPM, il s'agit de déterminer le composant défaillant ayant déclenché la détection d'un symptôme d'anomalie. A partir d'une défaillance, il est aisé de décrire les symptômes qui en résultent. Cependant, il est plus difficile de déterminer une défaillance à partir d'un symptôme (un même symptôme peut résulter de plusieurs défaillances). Le diagnostic sera vu comme le bloc servant à l'*identification* d'anomalies détectées [SAHRAOUI, 87].

II.3. TRAITEMENT D'ERREURS.

L'intégration d'un traitement d'erreur permet de rendre le système tolérant aux fautes [AVIZIENIS, 79]. Ce traitement peut être de deux natures : automatique ou interactif (assisté

par un opérateur). Il a pour objectif d'éviter l'apparition de défaillances critiques et la propagation des erreurs à d'autres composants du système.

D'après [LAPRIE, 85], la mise en œuvre du traitement d'erreur effective peut revêtir deux formes : recouvrement d'erreurs ou compensation d'erreurs. La compensation d'erreurs a comme effet de masquer les erreurs. De plus, elle ne porte pas sur les erreurs physiques ou de conception. C'est pour ces deux raisons que le recouvrement apparaît comme la solution la plus efficace pour le traitement d'erreurs.

Le recouvrement d'erreurs repose sur une détection rapide des erreurs, afin d'accroître l'efficacité du traitement [LAPRIE, 85]. Il peut prendre deux formes :

- la reprise : c'est la remise en service du système par transformation de l'état erroné en un état exempt d'erreur, soit en remplaçant l'état erroné par l'état exempt d'erreur, soit en "défaisant" les actions réalisées depuis le dernier état correct,
- la poursuite : elle consiste à mettre le système erroné dans un état correct qui n'est jamais survenu auparavant.

Le recouvrement d'erreurs active la transition de l'état effectif à l'état latent de l'erreur. Il est souhaitable de mettre en œuvre une procédure de reconfiguration du système, si l'on suppose que l'erreur a de fortes chances de redevenir, effective.

Le traitement d'erreurs peut être complété par des procédures de maintenance corrective (élimination des erreurs latentes précédemment effectives) et préventive (élimination des erreurs latentes non activées).

III. SURVEILLANCE DU PROCÉDÉ OU DU SYSTÈME DE COMMANDE.

Le procédé physique peut être surveillé de plusieurs manières. Nous allons nous intéresser à ces techniques. Il s'agit en l'occurrence de la surveillance des produits, de la surveillance des moyens et de la surveillance des opérations.

III.1. SURVEILLANCE DES PRODUITS.

Généralement, cette approche repose sur une comparaison de flux d'entrées et de sorties, tout en intégrant les caractéristiques nominales de production. Un écart important résultant de la comparaison, permet de conclure sur une défaillance dite d'exploitation. Il peut y avoir deux raisons : usure du matériel ou une mauvaise conception de la planification qui n'aurait pas intégrée toutes les contraintes de production. Cette approche ne peut être rapide et par conséquent nous l'écartons pour des raisons de sécurité. En effet, il est impossible de détecter, par exemple avec une telle approche, les collisions d'éléments du procédé.

L'approche exposée, est de nature quantitative. Il existe d'autres approches qualitatives comme la métrologie qui permettent d'être plus précis. Néanmoins, elles sont généralement lentes et sont donc à écarter.

La surveillance des produits peut avoir des avantages à long terme et peut donc être utilisée à des fins de maintenance préventive.

III.2. SURVEILLANCE DES MOYENS DE PRODUCTION.

Elle consiste à une surveillance directe des moyens par des capteurs dédiés à la surveillance. Il est donc nécessaire d'ajouter au procédé des capteurs et réaliser une étude des besoins en capteurs et du positionnement de ceux-ci [TOGUYENI, 92]. Malheureusement, il n'existe pas de méthode systématique à cet effet et en général, on procède par expérience. En plus de cette difficulté de positionnement, cette approche est coûteuse en capteurs. Par contre, les avantages qu'elle apporte sont incontestables. Elle permet de détecter rapidement les défaillances et de localiser directement le composant défaillant. Par conséquent, le recouvrement est déclenché (le diagnostic étant inutile), les erreurs ne sont pas propagées, et les arrêts d'urgence sont gérés directement. De plus, les risques d'accidents et de collisions sont écartés. Cette approche peut être améliorée par l'introduction d'actionneurs et capteurs intelligents [BAYART, 91], [STAROSWIECKI, 91].

Malgré les avantages offerts par une telle approche, pour nous elle est à écarter, en raison de son coût et de la non maîtrise du positionnement des capteurs.

III.3. SURVEILLANCE DES OPÉRATIONS.

Cette approche repose sur la vérification systématique du bon déroulement d'une commande. Il s'agit ici d'une surveillance indirecte du procédé par l'opération. Dans le cadre des SFPM automatisés, cette vérification est généralement réalisée par le SC par le biais des capteurs de position et de présence. Elle est basée sur le concept de sûreté de séquence [SAHRAOUI, 92]. En effet, la commande est supposée sûre et par conséquent si, il y a défaillance, il ne peut s'agir que du procédé.

Cette méthode permet de détecter les défaillances a posteriori en raison de l'utilisation des capteurs de position et de présence. C'est pour cette raison que nous estimons nécessaire d'ajouter des capteurs afin d'accélérer la détection. Il s'agit de capteurs permettant de rendre compte du déroulement de la commande et non de capteurs de surveillance. Cette approche, nous paraît-être la plus judicieuse et constitue une partie du système de surveillance proposé. Nous montrerons au chapitre IV, comment avec une telle approche, nous aboutirons à une détection instantanée des défaillances [ELKHATTABI, 92B].

III.4. SURVEILLANCE DE LA COMMANDE.

L'automatisation de la conception des SC, basée sur des modèles formels, permet une validation des modèles destinés à l'implantation. Cette validation rend le SC sain du point de vue des ordres destinés au procédé. Néanmoins, en phase d'exploitation, des incompatibilités entre le SC et le système physique peuvent exister. Les raisons de ce genre de situations peuvent être diverses : des erreurs de conception non détectées par la validation, des défaillance du réseau de transmission, ...

En général, la solution préconisée afin d'éviter ces incompatibilités, consiste en la mise en place d'un système de filtrage des commandes. Ce système modélise le comportement du procédé, et sert de référence pour le SC. Cette solution constitue la deuxième partie de notre approche [ELKHATTABI, 92A]. Il s'agit d'un modèle représentant tous les comportements élémentaires du système avec une intégration des contraintes de sécurité et de fonctionnement. Nous verrons au chapitre VI, comment un tel modèle sera construit et les avantages qu'il présente.

IV. APPROCHE DU LAAS.

Le LAAS propose une approche pour l'intégration de la surveillance dans les systèmes de commande automatique temps-réel. Une première proposition [SAHRAOUI, 87 ET 92] développait une approche fonctionnelle de la surveillance. J. CARDOSO propose une surveillance tolérante qui intègre au niveau de la commande, le comportement anormal du procédé [CARDOSO, 90]. Dans une approche plus récente [COMBACAU, 90], on privilégie l'aspect hiérarchisation. En effet, le système est organisé en niveaux de commande-surveillance. Ainsi, à chaque niveau d'abstraction de la commande correspond un module surveillance. La surveillance préconisée par COMBACAU intègre la fonction détection à la commande; tandis que les fonctions diagnostic et décision sont assurées de manière séparée.

IV.1. ORGANISATION D'UN MODULE COMMANDE-SURVEILLANCE

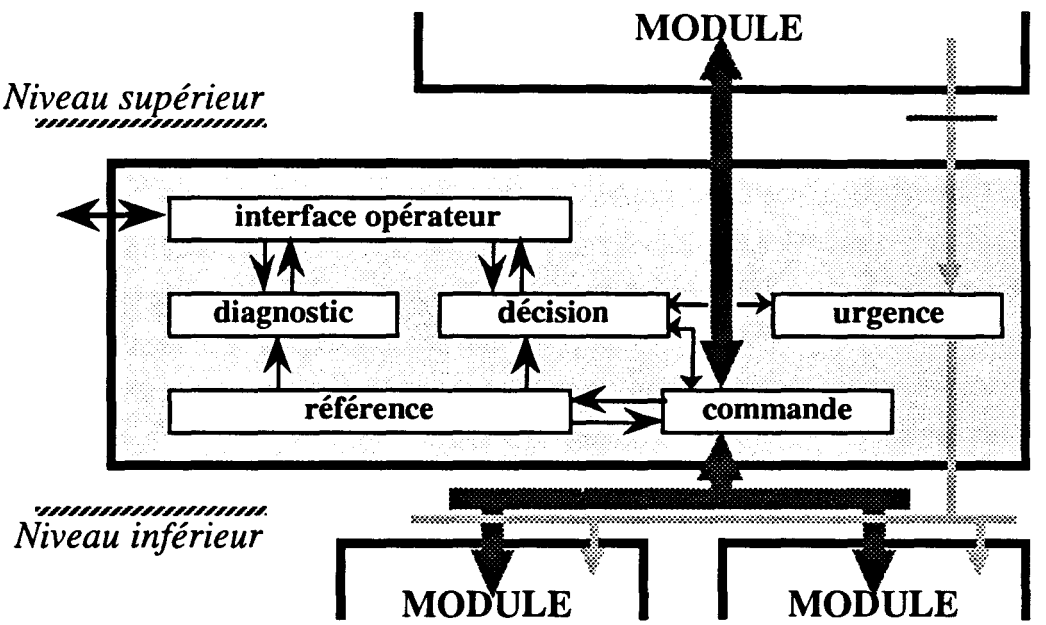


FIGURE II.1 : STRUCTURE D'UN MODULE COMMANDE-SURVEILLANCE [COMBACAU, 91].

Un module commande-surveillance est organisé en blocs (FIGURE II.1). Chaque bloc assure une fonction du système. Les blocs commande et référence forment le système de commande et les autres blocs constituent le système de surveillance.

IV.2. SYSTÈME DE COMMANDE.

Les deux blocs sont modélisés par des **R**éseaux **d**e **P**etri à **O**bjets (**RdPO**) [SIBERTIN, 90], [PALUDETTO, 91]. Ces modèles s'appuient sur le concept d'*Activité* [COMBACAU, 91]. Une activité peut aussi bien représenter une commande de bas niveau, qu'une commande plus complexe, permettant ainsi une généricité de la commande aux différents niveaux d'abstraction.

Une place représente une activité en cours d'exécution, et un jeton modélise l'entité physique du procédé commandée. Sur la FIGURE II.2, la place P_m modélise l'exécution de l'activité A_m, les places P_l et P_k sont les préconditions (états potentiels des entités du procédé) de A_m et les places P_n et P_q représentent les postconditions. D_i et F_i portent sur les attributs des jetons.

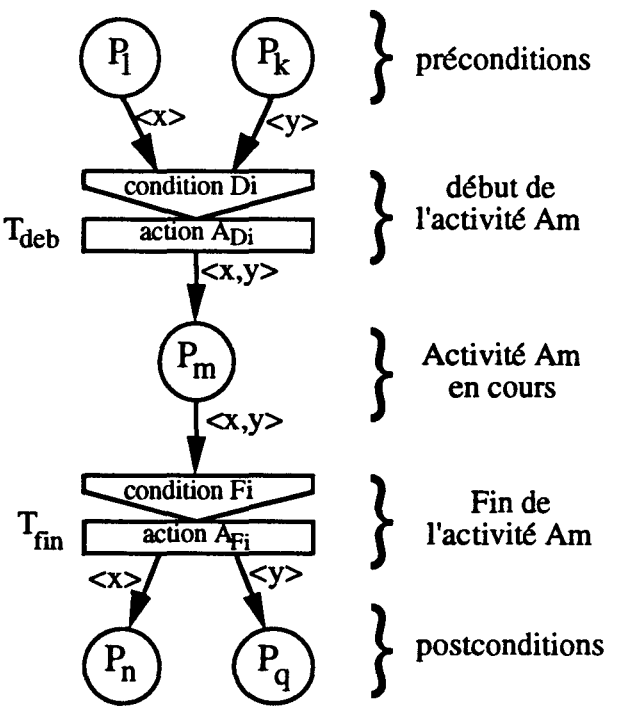


FIGURE II.2 : MODÉLISATION D'UNE ACTIVITÉ PAR RDPO [COMBACAU, 91].

Le modèle de référence définit l'ensemble des comportements normaux sous forme d'activité. Il décrit tous les états utilisables, du sous-système en question. Les transitions représentent les changements d'états du procédé correspondants à des débuts et des fins d'activités. Un changement d'état modélise soit un envoi de requête vers le niveau inférieur

soit une réception de compte rendu depuis le niveau inférieur. Il réagit donc à chaque action émise par la commande et aux comptes-rendus formulés par le procédé.

Ce bloc a pour vocation de fournir une image aussi fidèle que possible de l'état réel du système commandé. Il interagit avec trois blocs du module de même niveau :

- avec la commande, afin de prendre en compte des contraintes de fonctionnement à l'émission de commande,
- avec le diagnostic, afin de lui fournir l'état du procédé,
- avec la décision, afin de déterminer les états envisageables (séquence de reprise).

Le bloc commande modélise les séquences opératoires d'un mode de fonctionnement particulier du procédé. Cette séquence est construite depuis les activités du bloc référence. La construction d'une telle séquence permet d'imposer un ordre d'exécution des activités et de choisir les ressources qui vont les exécuter. Il s'agit d'une commande séquentielle, qui ne prend pas en compte les contraintes liées aux moyens de production.

Le fonctionnement du SC se traduit par l'évolution synchronisée des deux blocs. En effet, l'envoi d'une requête (après vérification des préconditions de l'activité) se traduit par un appel vers le bloc référence et la réception d'un compte rendu émis par le niveau inférieur signifie la fin de l'activité. Garantir un tel fonctionnement, oblige le concepteur à maîtriser de manière continue les communications interniveaux.

IV.3. SYSTÈME DE SURVEILLANCE.

Dans cette approche, la surveillance est assurée par les fonctions classiques de la surveillance :

- **la détection** : elle est basée sur la comparaison du comportement réel du procédé et du comportement spécifié par le modèle de commande. Cette fonction est intégrée à la commande. Un écart significatif (compte rendu anormal ou dépassement d'une date limite) est interprété comme étant un symptôme de défaillance. Cette fonction est basée sur le principe de chien garde. Deux types de défaillances peuvent être détectés :
  - l'activité n'a pas débuté à la date de début au plus tard,
  - l'activité n'est pas terminée à la date de fin au plus tard (dysfonctionnement).

- **Le diagnostic** : il permet d'identifier l'incident dont les symptômes ont été reconnus. Cette fonction est séparée de la commande. Le diagnostic est mis en œuvre sur la base d'un système de règles de production. Les défaillances non prévues sont ignorées par la surveillance et conduisent généralement à un arrêt du SC.

- **La décision** : elle assure la reconfiguration des modèles de la commande, déclenche des procédures d'urgence, détermine une séquence et un point de reprise et propage la défaillance vers le niveau supérieur, le cas échéant.

- **La reprise** : il s'agit de l'exécution d'une séquence d'activités afin de remettre le système dans un état cohérent. Cette fonction est intégrée au bloc commande.

- **Les procédures d'urgence** : ce sont des traitements prioritaires qui garantissent la sécurité des équipements et des hommes.

L'**interface opérateur,** définie pour chaque module, permet à l'utilisateur d'intervenir dans la boucle de traitement de défaillance à tous les niveaux. D'après COMBACAU, cette interface est utile surtout pour pallier les insuffisances de l'interface procédé/commande. Un deuxième avantage de cette interface réside dans le fait que le système n'est pas autorisé à décider de manière autonome du traitement correctif. Cette interface peut être utilisée en fonctionnement normal, afin d'éviter l'apparition d'une défaillance prévue ou constatée à l'avance par l'opérateur.

IV.4. FONCTIONNEMENT.

La séquence de base d'une boucle de surveillance consiste en la détection, le diagnostic, l'élaboration d'une solution et l'application de cette solution. Le traitement d'une défaillance a été rendu purement séquentiel pour découpler les fonctions entre-elles. Le fonctionnement général est donné par le Réseau de Petri de la FIGURE II.3.

La détection assurée par la commande (T2), génère un symptôme (P2). Ce symptôme déclenche (T6) le diagnostic (P4). Le diagnostic s'achève (T7) après récolte éventuelle d'informations (T3 et/ou T10), par le passage au bloc (T7) décision. La décision (P6) peut être activée (T8) pour trois raisons : analyse d'un symptôme (T7), déclenchement d'une procédure d'urgence par l'opérateur (T11) ou par le niveau supérieur (T14). Un manque

d'information peut amener à une consultation (T4) ou à une demande (T12). La transition T13 sert à activer éventuellement une procédure d'urgence (P8) demandée par l'opérateur. La décision se termine par l'élaboration d'une séquence (P3) qui sera transmise (T9) au système de commande.

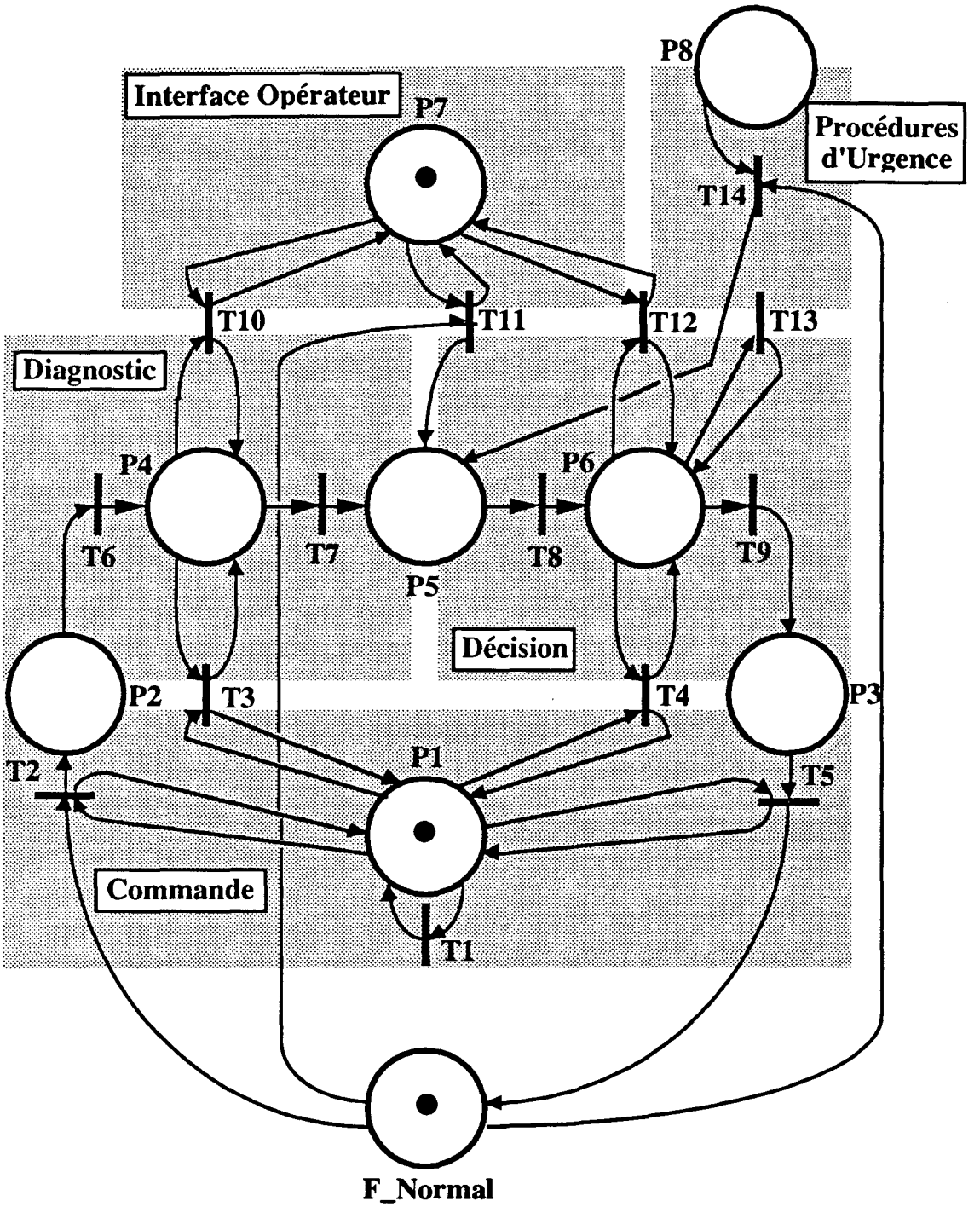


FIGURE II.3 : FONCTIONNEMENT D'UN SYSTÈME DE SURVEILLANCE [COMBACAU, 91].

V. APPROCHE DE A.K.A. TOGUYENI DU LAIL.

V.1. INTRODUCTION.

A.K.A. TOGUYENI propose une approche de la surveillance en ligne des SED. Les fonctions assurées par cette approche sont de nature classique : acquisition, détection, diagnostic et recouvrement. La structuration (FIGURE II.4) est inspirée de la notion de filtre introduite par le CRAN [LHOSTE, 91].

V.2. LA DÉTECTION.

L'idée de base de la détection correspond au principe suivant :

"Toute panne d'une entité matérielle se manifeste par la violation du temps imparti à la réalisation d'une action qu'elle réalise." [TOGUYENI, 92]

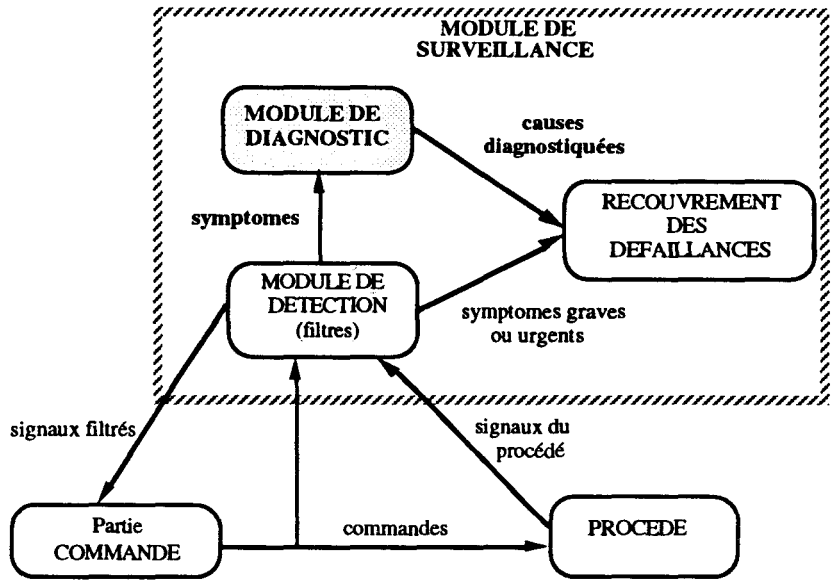
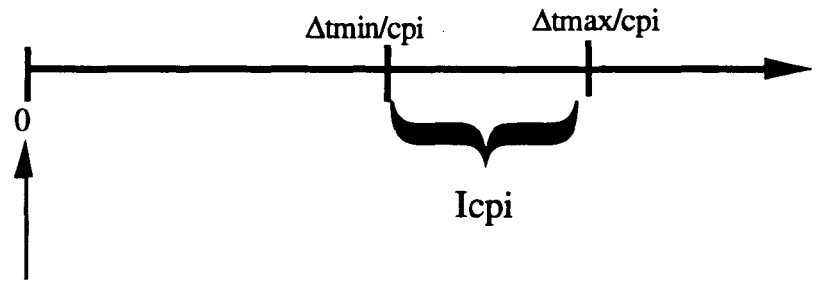


FIGURE II.4 : STRUCTURATION DE LA FONCTION SURVEILLANCE ET CONNEXION AVEC LA PARTIE COMMANDE ET LE PROCÉDÉ [TOGUYENI, 90].

En partant de ce principe, quelque soit le niveau de commande, il construit la détection sur l'exploitation d'une modélisation temporelle de chaque opération demandée au procédé (FIGURE II.5).



Date d'arrivée de l'événement de départ déclenchant l'analyse du signal de validation émis par cpi (start-event/cpi)

Intervalle de temps durant lequel le signal de validation est attendu

FIGURE II.5 : COMPORTEMENT TEMPOREL NORMAL DE CHAQUE ENTITÉ [TOGUYENI, 90 ET 92].

Ce modèle met en relation un événement concernant une opération (*Start-Event*) et son compte-rendu (*CR*) de validation. Le CR est attendu dans une fenêtre temporelle déterminée à partir des caractéristiques physiques du procédé. Dans l'hypothèse d'une commande saine, la surveillance d'opération est équivalente à une surveillance indirecte du procédé. La réactivité du modèle est assurée par la simplicité d'interprétation.

Sur la base du modèle générique de la FIGURE II.5, deux types de symptômes correspondants à des comportements défaillants différents, sont définis. Pour la mise en œuvre de la détection, un mécanisme de *perception-filtrage* des CR est proposé. Ce mécanisme est spécifié à l'aide des Réseaux de Petri Temporels Objets [CARDOSO, 90]. Le filtrage des CR est complété par une *classification* des symptômes, ayant pour objectif d'accroître la réactivité du processus de recouvrement.

V.3. LE DIAGNOSTIC EN LIGNE.

Les travaux de A.K.A. Toguyeni, portent essentiellement sur le diagnostic [TOGUYENI, 90 ET 92]. Il est basé sur une modélisation qualitative du Procédé dans le contexte du *diagnostic en ligne*. Ce contexte peut être résumé par l'énoncé de trois contraintes : le temps réel, l'absence de données basées sur l'expérience (la surveillance de systèmes neufs) et la nécessité d'un diagnostic sûr pour le recouvrement.

Face aux limites des systèmes de diagnostic actuels (Automatic Test Equipment et systèmes experts hors ligne), une méthodologie de conception orientée sur l'exploitation

d'outils d'intelligence artificielle a été proposée. La modélisation qualitative retenue est une modélisation fonctionnelle. Elle comprend trois composantes qui sont : la composante *structurelle*, la composante *fonctionnelle*, et la composante *comportementale* [SEMBUGAMOORTHY, 86], [CHANDRASEKARAN, 89].

Le concept de base de cette approche est la fonction qui est définie comme étant :

"la réponse d'un système ou d'un composant à un stimulus donné, indépendamment du contexte dans lequel il est sollicité."

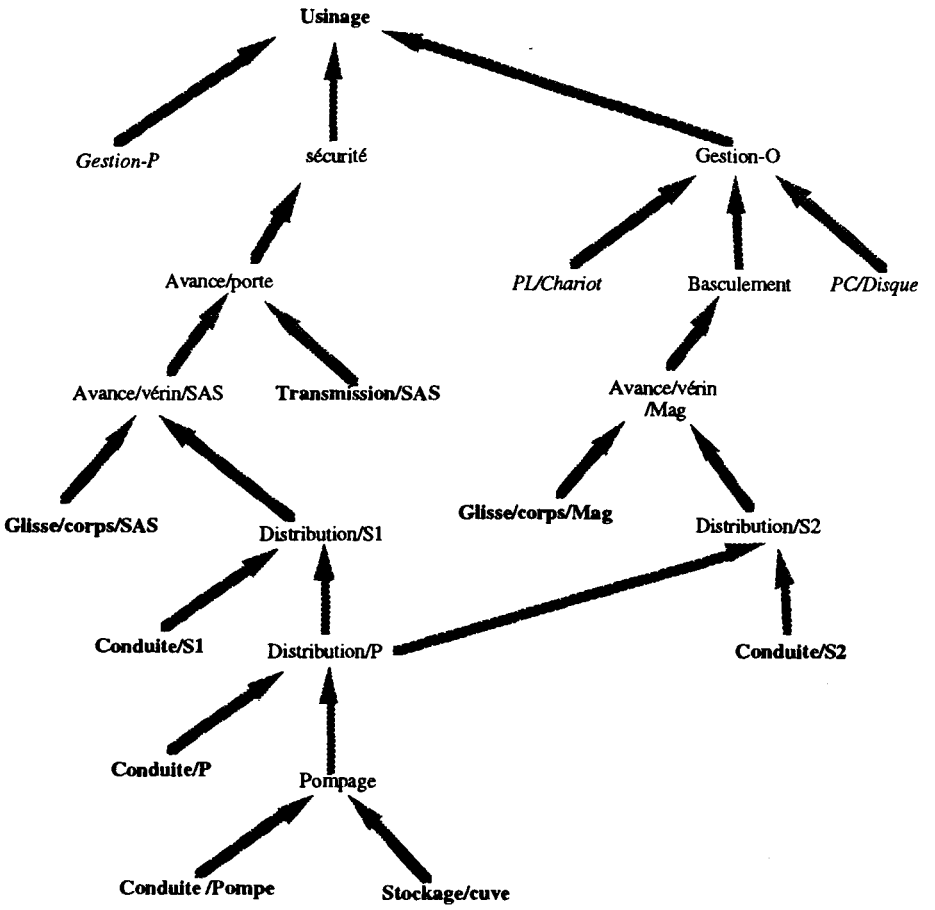


FIGURE II.6 : LE GF D'UN CENTRE D'USINAGE [TOGUYENI, 92].

A partir de ce concept de base, une méthodologie est développée. Elle aboutit à la conception de deux modèles distincts : le modèle *structurel* et le modèle *fonctionnel*. Le modèle structurel représente la topologie et la hiérarchie du système à travers les liens structurels entre composants. Il intègre la flexibilité du système. Ce modèle est utilisé dans le cadre du *pronostic* qui est un processus de *recouvrement immédiat* intégré au diagnostic. Le diagnostic est mis en œuvre à partir du modèle fonctionnel appelé Graphe Fonctionnel (GF)

(FIGURE II.6). Le GF modélise les liens de dépendances fonctionnelles entre "toutes" les fonctions d'un système et celles de ses composants proches ou lointains. Un algorithme de positionnement des capteurs, à partir des propriétés du GF, a été développé. Ce positionnement doit permettre d'assurer l'observabilité du système (du point de vue de la détection des défaillances) avec un nombre optimal de capteurs. Ce positionnement optimal des capteurs est une originalité.

Le diagnostic est structuré en deux étapes : la *localisation* et *l'identification* [TOGUYENI, 91]. Le but de la localisation (ou diagnostic local) est de circonscrire l'origine de la défaillance à une zone du système. Elle est réalisée par les noeuds correspondants aux fonctions du système ayant une observabilité directe, appelés *noeuds initiateurs*. Le mécanisme de localisation est mis en œuvre par l'interprétation de Signature Temporelle Causale (STC) [TOGUYENI, 91], [FARAH, 91]. Un STC est une extension des règles de production qui modélise le comportement défaillant d'une portion du système.

L'identification (ou diagnostic global) a pour rôle de déterminer précisément quelles *fonctions initiatrices* ou *noeuds terminaux* sont à l'origine d'une défaillance. Le GF est interprété comme un graphe d'hypothèses dans lequel quatre types de noeuds sont définis : les noeuds *initiateurs*, les noeuds de *validation-directe*, les noeuds de *validation-indirecte*, les noeuds *simples*. Chaque type de noeud correspond à des fonctions qui du fait de leurs propriétés statiques (position dans le graphe) ou dynamiques (type et nature de l'observabilité), ont un apport différent dans le cadre de l'identification. Ainsi, les hypothèses sont générées par les noeuds initiateurs, et sont validées par les noeuds de validation-directe ou de validation-indirecte.

Le GF et le modèle structurel, sont conjointement utilisés pour réaliser l'analyse des conséquences (ou pronostic) afin d'anticiper l'appel par le système de commande de fonctions dont la défaillance est latente.

V.4. LE RECOUVREMENT.

Cette fonction a pour objectif d'assurer la sécurité du système et de permettre la poursuite de la production sur les ressources de production saines. A partir de ces deux objectifs qui peuvent paraître opposés, du point de vue des actions à mener sur le procédé et la commande, une classification des procédures de recouvrement est proposée. A.K.A. TOGUYENI a formulé un certain nombre de propositions qui visent à adapter la

commande et le procédé à une gestion implicite des modes de marches dégradés par la fonction de pilotage d'un atelier flexible [TOGUYENI, 91 ET 92]. Il propose notamment une structuration de la commande en couches modélisant chacune une forme de flexibilité du procédé. L'idée sous-jacente est que le recouvrement et donc la gestion des modes de marches dégradés, sont fonction de la flexibilité résiduelle du système de production.

V.5. CONCLUSION.

L'approche proposée pour la surveillance en ligne s'inscrit dans le cadre d'une mise en œuvre répartie. Le but est de déduire le maximum d'information en local à partir des données brutes. Ensuite, le diagnostic est affiné par coopération des différentes entités délocalisées, concernées par une défaillance. Ainsi, la quantité d'information transitant par les réseaux est minimisée, ce qui correspond plus à une application temps réel. Elle s'inscrit également, dans le contexte des travaux sur le développement de capteurs et actionneurs intelligents.

L'aspect réparti de la surveillance, pose le problème d'asynchronisme propre à toute application répartie. Pour la conception, ce problème a été occulté, étant donné que l'ensemble des processeurs locaux (les noeuds du modèle) sont synchronisés par la même horloge : celle du calculateur.

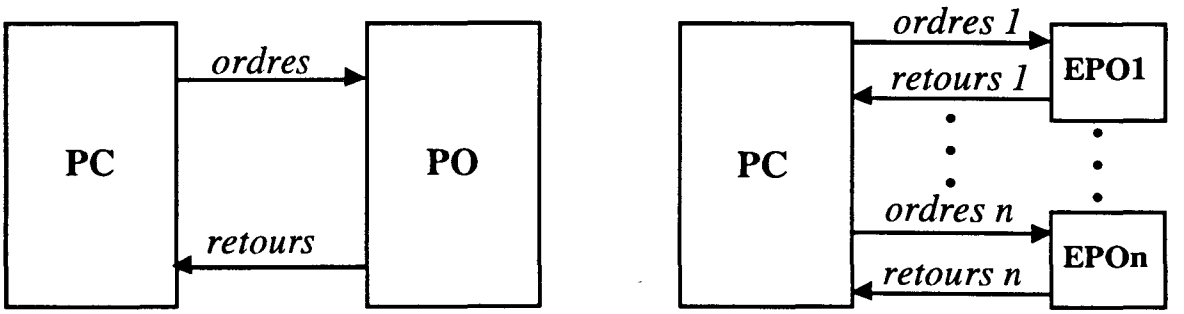
VI. APPROCHE DU CRAN.

VI.1. DE LA VISION ATOMIQUE À LA VISION MOLÉCULAIRE.

L'approche du CRAN consiste à modéliser la **P**artie **O**pérative (**PO**) afin de structurer la **P**artie **C**ommande (**PC**). Elle rejette le schéma bloc fonctionnel classique (FIGURE II.7A), caractérisé par une vision atomique de l'interface PO/PC; et propose d'appréhender cette interface avec une vision moléculaire (FIGURE II.7B) [ALANCHE, 86]. Elle est construite autour de la notion d'**E**lément de la **P**artie **O**pérative (**EPO**). La PO est vue comme un ensemble d'EPO.

Chaque EPO est défini par un ensemble d'ordres auxquels il est sensible et de comptes rendus qu'il formule. Un ensemble associé à un EPO est disjoint de tout autre ensemble caractérisant un autre EPO. Chaque ensemble est régi par une sémantique qui

définit les relations de cause à effet entre les ordres et les retours. Ces relations constituent le *comportement* de la PO. Cette vision moléculaire conduit à une modélisation du comportement des différents éléments.



(a) : vision atomique **(b) : vision moléculaire**

FIGURE II.7 : D'UNE VISION ATOMIQUE À UNE VISION MOLÉCULAIRE DE LA PO [ALANCHE, 86].

VI.2. MODÉLISATION DE COMPORTEMENT.

Le comportement d'un EPO est caractérisé par des états en nombre fini : des états physiques stables et des états transitoires. La modélisation consiste à identifier ces états, et à les organiser afin de caractériser l'évolution des caractéristiques observables par rapport aux ordres de la commande. Le modèle comportemental est de type états/transitions. Cette modélisation de comportement normal s'intéresse au fonctionnement normal. Ainsi, tout ce qui sort de cet état de choses, sera considéré comme un dysfonctionnement. Ce modèle est développé à des fins d'aide à la conduite et de détection des défauts de fonctionnement de la PO. Le modèle comportemental peut être utilisé de deux manières différentes : en émulation et en filtre.

VI.2.1. Utilisation du modèle en émulation.

Le modèle comportemental de l'EPO émule le fonctionnement de l'EPO. Il génère ainsi des comptes rendus aux ordres qu'il reçoit. Des temporisations sont introduites afin de contrôler l'évolution de l'EPO.

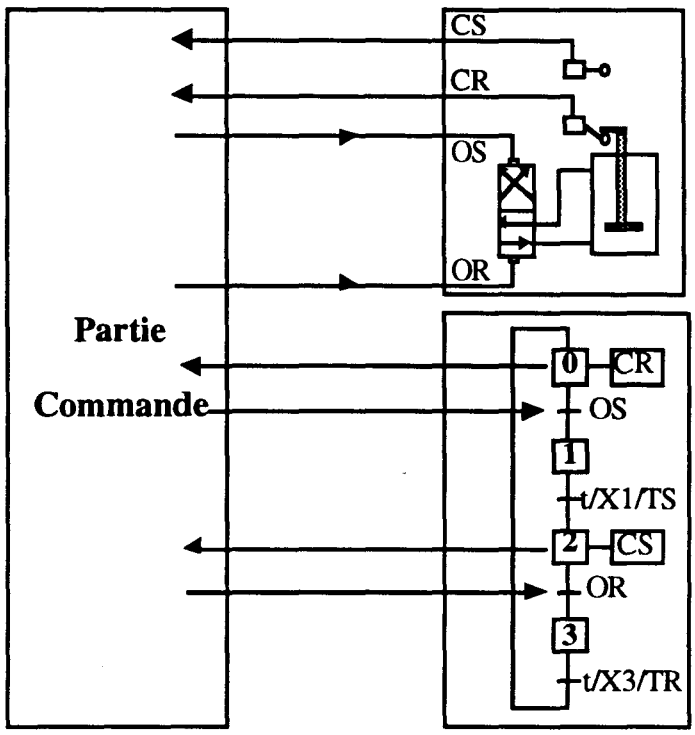


FIGURE II.8 : UTILISATION EN ÉMULATION [LHOSTE, 91].

VI.2.2. Utilisation du modèle en filtre.

Le modèle de l'EPO est placé, entre la PC fonctionnelle et la PO, afin de réaliser une surveillance par prévision du comportement. Un modèle d'EPO permet [TIXADOR, 89] :

- de détecter que les ordres reçus sont cohérents avec l'état du modèle des EPO et par conséquent avec la PO (prévision des ordres),
- de détecter que les retours issus de l'EPO sont bien attendus par la PC implémentant le modèle comportemental (prévision des retours),
- de mettre à jour en temps-réel des indicateurs de gestion technique, nécessaires à la mise en œuvre d'une politique de maintenance préventive efficace.

Les filtres de comportement permettent d'un point de vue opérationnel d'analyser en temps réel les stimuli qu'un équipement reçoit et les ordres qu'il transmet, afin d'assurer une fonction de contrôle/commande en cohérence avec l'état d'actionnement et le procédé [SFALCIN, 89].

L'implémentation des filtres est réalisée sous la forme de composants logiciels. L'atelier **SPEX** (**SP**écification **EX**écutable), est utilisé comme plate-forme

d'implémentation [TIXADOR, 89], [PANETTO, 91]. Cet outil offre un environnement de développement dédié à la spécification exécutable des Machines et Systèmes Automatisés de Production (MSAP). La méthode utilisée est une combinaison d'une analyse descendante avec IDEF-0 [VORGRIG, 86], et d'une synthèse ascendante avec les Boites Fonctionnelles (BF). La BF est définie comme étant un composant logiciel réutilisable. Ainsi, à chaque EPO sera associé une BF générique.

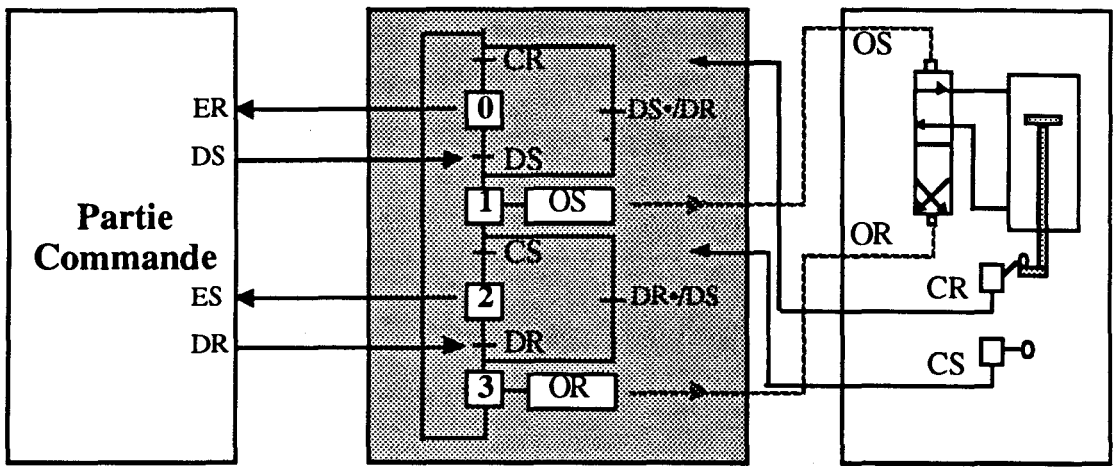


FIGURE II.9 : UTILISATION EN FILTRE [LHOSTE, 91].

VI.3. SURVEILLANCE PAR PRÉVISION DE COMPORTEMENT.

La surveillance par prévision de comportement (basée sur la notion de filtre) permet la détection de retours tardifs, intempestifs de la PO par rapport à son activité, de même que les ordres de la PC inattendus par rapport à l'état de la PO. Cette surveillance peut être de nature logique (attente d'un retour) ou temporelle (attente d'un retour pendant un intervalle de temps) [SFALCIN, 89].

Un ordre émis par la PC est filtré par la détection afin d'évaluer sa compatibilité avec l'état du procédé. Une erreur est générée en cas d'incompatibilité. Dans le cas où, l'ordre est compatible, une prévision des retours "normaux" est élaborée. Un test de comparaison du retour aux retours prévisibles génère soit un retour à la PC (en cas d'égalité), soit une erreur (en cas d'inégalité) au module de traitement d'erreurs intégré à la PC.

L'objectif d'une telle approche est d'opérer à une détection des défauts rapide et précise [ALANCHE, 86]. Néanmoins, cette vision moléculaire n'intègre pas les contraintes fonctionnelles qui régissent le fonctionnement de plusieurs EPO coopérants pour réaliser une

tâche. ALANCHE préconise l'utilisation du filtre comme modèle comportemental à toute sollicitation de l'EPO. Cette utilisation systématique implique l'enrichissement du modèle qui risque d'aboutir à une explosion combinatoire du nombre d'états du modèle et par conséquent de réduire la rapidité de détection des défauts. En effet, SFALCIN [89] montre que l'utilisation des filtres réduit de 75% les temps de détection.

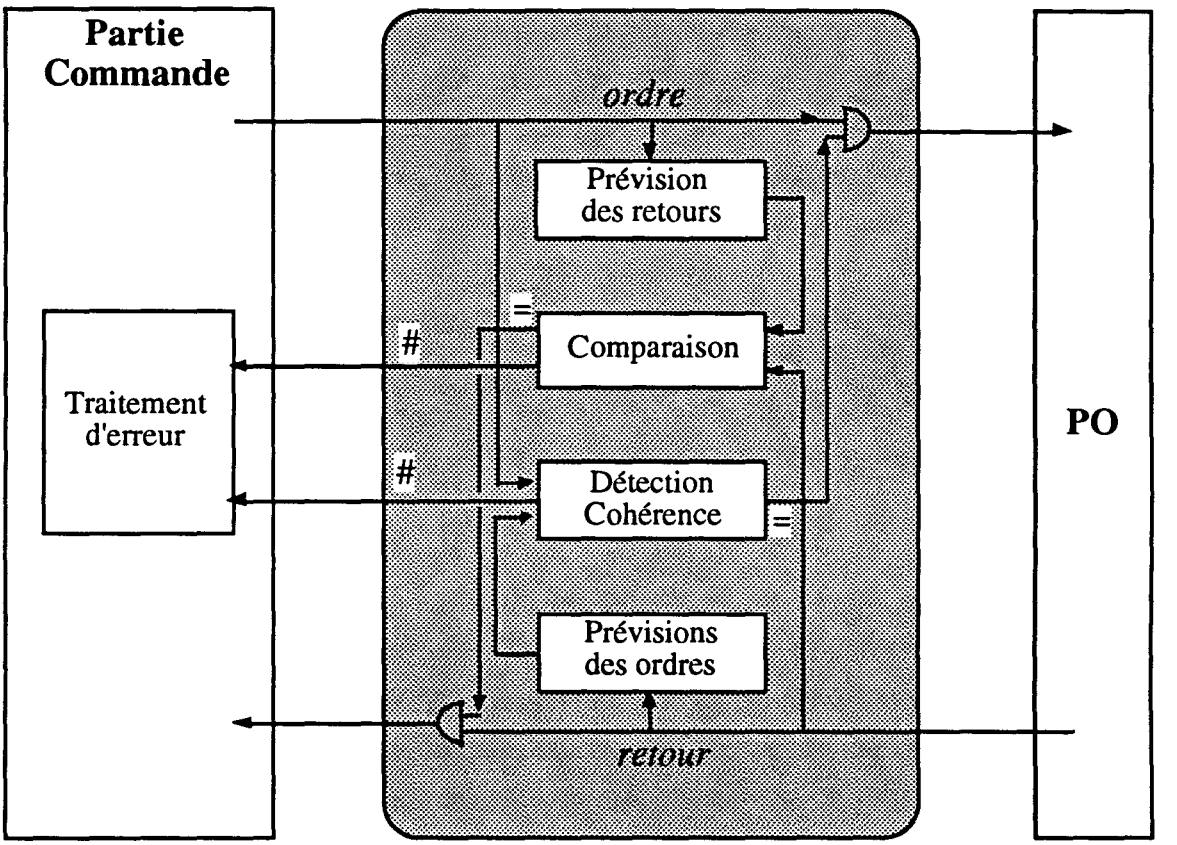


FIGURE II.10 : SURVEILLANCE PAR PRÉVISION DE COMPORTEMENT [LHOSTE, 91].

VII. PROPOSITION D'UNE STRUCTURATION DE LA SURVEILLANCE.

VII.1. HYPOTHÈSES DE TRAVAIL.

L'approche que nous allons développer dans ce mémoire concerne les SED et plus particulièrement les SFPM automatisés. Etant dans le contexte des SED, le comportement est de nature discrète. En effet, les états sont discrets, et l'évolution se fait à base d'événements.

Ce qui nous amène à considérer l'information fournie par les capteurs sous forme binaire même si ces capteurs à l'origine, sont analogiques ou numériques . A cet effet, la définition d'un seuil par rapport au domaine d'appartenance du signal, permet de transformer le signal en valeur booléenne, donc discrète (dans le cadre de la surveillance : normal ou anormal).

Nous supposons que la commande est saine et respecte les contraintes fonctionnelles régissant la coopération des ressources du procédé. Les contraintes fonctionnelles permettent de déterminer un séquencement strict des commandes. Ces contraintes sont nécessaires afin d'éviter les risques d'accidents humains ou d'équipements et des collisions (sécurité), et d'assurer par là même une disponibilité des ressources. Cette supposition, nous conduit à une surveillance basée sur la réalisation des commandes : surveillance indirecte du procédé.

Notre objectif principal est d'aboutir à une détection réactive (cf CHAPITRE IV) des défaillances. La rapidité de détection et le séquencement strict des commandes permettent d'éviter la propagation des erreurs. En effet, la détection d'une défaillance gèle le système de commande du composant défaillant et le séquencement évite la "contagion" des autres composants.

Nous verrons plus loin que l'organisation de la surveillance en modules (cf CHAPITRE IV), dédiés à la surveillance des différents composants du procédé, rend les étapes de localisation et d'identification des composants défaillants, implicites et immédiates. L'identification des composants défaillants déclenche un recouvrement d'erreurs qui peut engager éventuellement un processus de maintenance. La surveillance doit donc être construite dans le cadre d'une politique de maintenance. Cette organisation nous amène à réaliser, une surveillance de bas niveau, intégrée au SC. Le système de surveillance sera chargé de fournir les comptes rendu et de détecter les défaillances : ces informations doivent être donc des faits certains.

VII.2. NATURE DU SYSTÈME À SURVEILLER.

Tout système de production d'un atelier flexible peut se décomposer en trois parties [BOUREY, 88] (FIGURE II.11) : le Niveau Hiérarchique, la Partie Commande et le Procédé.

Le procédé englobe les **O**bjets **C**ommandables **E**lémentaires (**OCE**) [ELKHATTABI, 92A], le système de transport, les actionneurs des OCE, les capteurs, ainsi que les commandes numériques associées à certains éléments de l'atelier (Robot, Tour, ...).

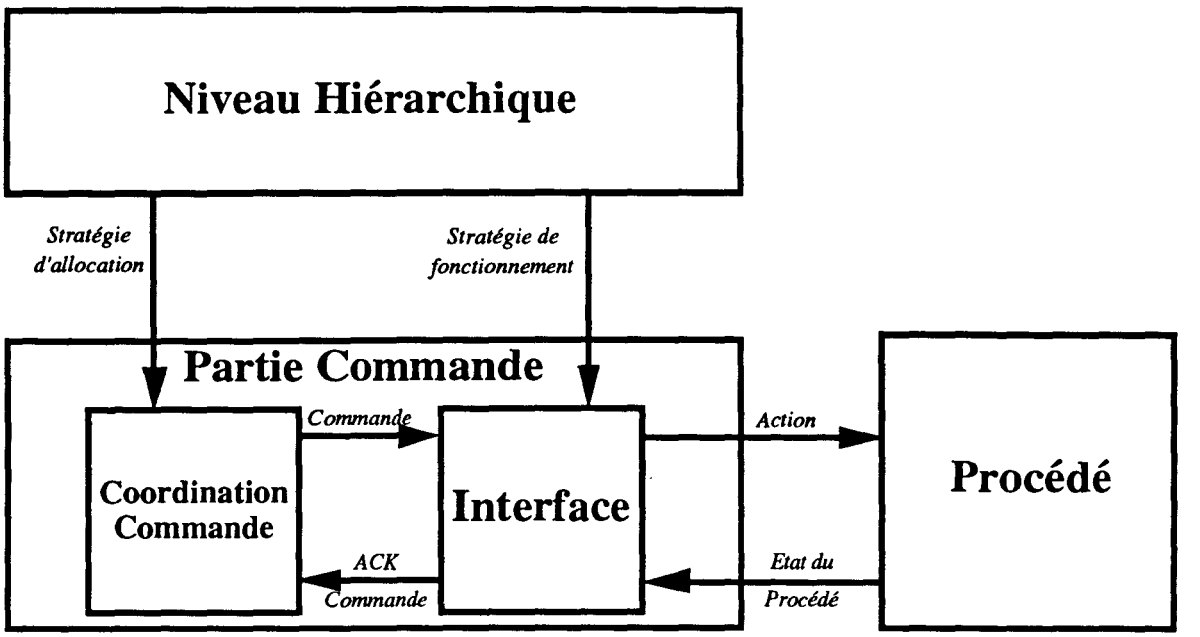


FIGURE II.11 : SYSTÈME DE PRODUCTION.

La Partie Commande (**PC**) est utilisée pour la synchronisation des différentes ressources d'un processus flexible et pour assurer la coopération entre les éléments du système. La flexibilité du système amène un niveau de parallélisme assez élevé. Ce parallélisme introduit des conflits et des indéterminismes directionnels qui ne peuvent être résolus par la PC [BOUREY, 88; KAPUSTA, 88].

Le Niveau Hiérarchique (**NH**) intègre des fonctions de décision, qui permettent de superviser et de paramétrer la commande : résolution de conflits et d'indéterminismes de la PC. Il sert aussi à mettre en œuvre les stratégies d'allocation et de fonctionnement du système de production [CRAYE, 89].

VII.3. STRUCTURATION PROPOSÉE.

Pour répondre à nos objectifs, nous proposons d'interfacer la PC et le procédé par un système de surveillance (FIGURE II.12). Ce système se décompose en quatre modules :

- Filtres de commande,
- Contrôle de commande,
- Recouvrement des erreurs,
- Traitement des anomalies.

Les trois premiers modules sont destinés à la surveillance de la réalisation des commandes. Ils se chargent ainsi de la détection des erreurs de conception, de la compatibilité des commandes avec l'état réel du procédé et des erreurs physiques et de leur recouvrement. Quant au module de traitement des anomalies, il se rattache à la surveillance du procédé.

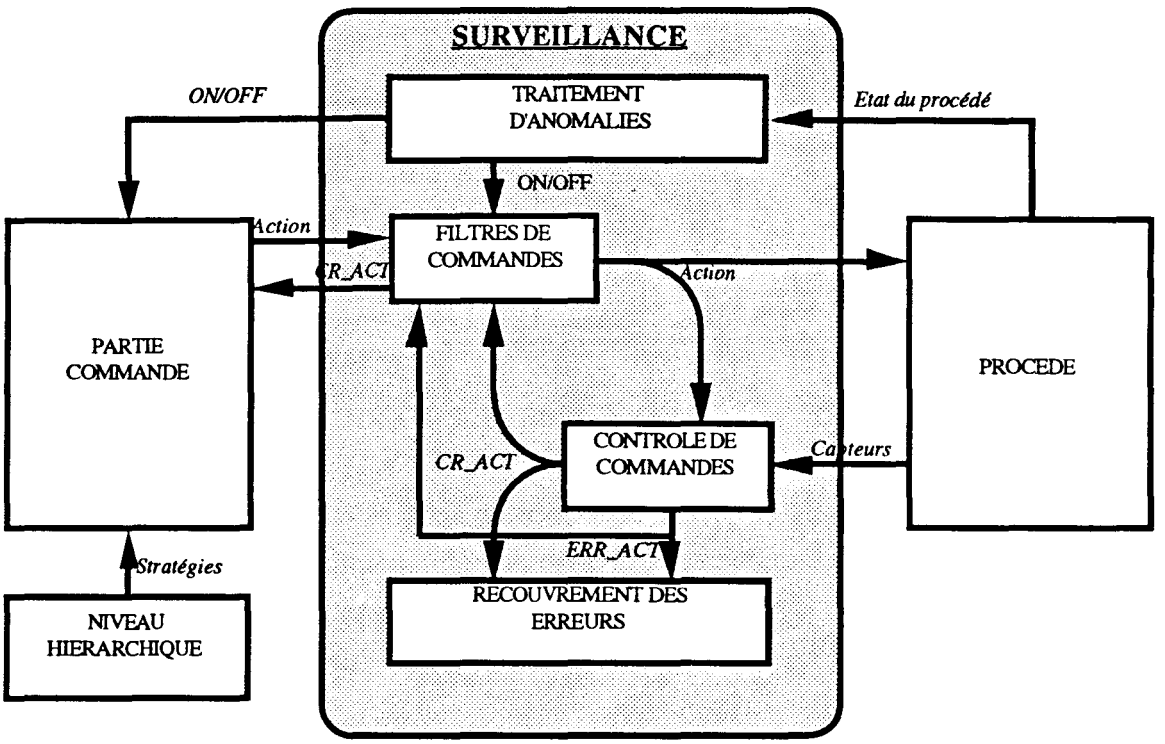


FIGURE II.12 : SYSTÈME DE PRODUCTION ET SYSTÈME DE SURVEILLANCE.

VII.4. FILTRES DE COMMANDES.

Les actions transmises au procédé passent par le Module des Filtres de Commande (MFC). Ce dernier s'assure de la compatibilité des actions avec l'état du procédé [ALANCHE, 86]. Une action n'est transmise au procédé, que si, elle est compatible avec son état, c'est à dire réalisable. Elle est dite valide [ELKHATTABI, 92A ET 92B]. Cette fonction est mise en oeuvre grâce à une image du procédé constamment tenue à jour. Le modèle représente à cet effet tous les comportements élémentaitres des différents composants du système physique : description fine et complète.

Des contraintes de sécurité et de fonctionnement sont adjointes à ce modèle, afin d'éviter les risques d'accidents (humains ou matériels) et d'assurer le bon fonctionnement du système.

Ce modèle peut être de plus utilisé à plusieurs niveaux du système de contrôle/commande. En effet, il peut servir à déterminer des séquences de commande à des fins de recouvrement ou de pilotage temps-réel. Une application informatique a été développée afin d'assister et de systématiser cette démarche [ELKHATTABI, 93B]. La conception de ce module fait l'objet du chapitre VI.

VII.5. CONTRÔLE DE COMMANDES.

Le Module de Contrôle de Commande (**MCC**) sert à contrôler la réalisation des actions reçues par le procédé : surveillance indirecte du procédé [ELKHATTABI, 91 ET 92B]. Afin de juger du bon déroulement de l'action, il a besoin de connaître tous les changements effectifs des états du procédé, et cela de manière instantanée. La détection instantanée des dysfonctionnements, permet d'éviter la propagation de l'erreur à d'autres composants. On assure ainsi une fiabilité au système de commande.

L'évolution du système permet de confirmer ou d'infirmer la réalisation de l'action. Ce constat se fait grâce à l'information récoltée auprès des capteurs. Etant dans un environnement discret, cette information sera considérée sous une forme binaire. Pour des données non binaires, nous utiliserons un seuil de discretisation. Le chapitre IV présente la démarche de conception du MCC de manière détaillée.

VII.6. RECOUVREMENT D'ERREURS.

En cas d'erreur (non réalisation de l'action), le Module de **R**ecouvrement des Erreurs (**MRE**) est activé. Il prend la relève de la PC, afin de mettre le procédé dans l'état initialement souhaité par le système de commande. Le traitement peut être de nature automatique ou interactif ou les deux à la fois. Ayant une détection rapide, le recouvrement peut revêtir deux formes : reprise et poursuite [LAPRIE, 85].

Le recouvrement consiste à déterminer la séquence des commandes à réaliser afin de mettre le système dans un état exempt d'erreurs. Une reconfiguration du SC est souhaitable si l'erreur a de fortes chances de redevenir effective. Dans ce cas, des procédures de maintenance sont adjointes au recouvrement pour le compléter. Le recouvrement d'erreurs n'est pas développé dans ce mémoire et fera l'objet d'une réalisation future.

VII.7. TRAITEMENT D'ANOMALIES.

Outre les erreurs citées, il peut y avoir des défaillances matérielles inattendues : panne d'un OCE, présence d'un élément extérieur, ... Le Module de Traitement des Anomalies (MTA), a pour but de détecter ces erreurs et de gérer la mise hors service (OFF) d'un OCE défaillant. Dans une telle situation, une reconfiguration de la commande est nécessaire pour assurer un fonctionnement normal. Après remise en état de l'OCE défaillant, ce module gère la remise en service de celui-ci (ON). Ce module ne sera pas traité dans ce mémoire et constitue une des voies futures de recherche.

CONCLUSION.

Dans ce chapitre, nous nous sommes intéressés aux approches de surveillance existantes et à notre propre vision de la surveillance en ligne. La structuration de notre approche est composée de deux modules principaux : le MCC et le MFC. Nous verrons dans les parties suivantes, comment avec une telle structuration nous allons atteindre nos objectifs. Il s'agit de garantir tout à la fois la sécurité humaine et la sécurité des équipements, mais aussi d'accroître la disponibilité matérielle et d'assurer la fiabilité du SC.

L'approche proposée est de nature discrète. Elle est basée d'une part sur une surveillance indirecte du procédé et un filtrage des commandes d'autre part. La réactivité du système de détection et l'organisation interne du MCC (CHAPITRE IV) rendent implicites les étapes de localisation et d'identification des composants défaillants. Le recouvrement d'erreurs rend le SC tolérant aux fautes.

La détection instantanée des défaillances, permet un recouvrement rapide de celles-ci et évite la contagion d'autres composants du système. Ces deux caractéristiques ont un effet direct sur la disponibilité des composants du sytème physique.

Le filtrage des commandes, intégrant les contraintes de sécurité et de fonctionnement, évite le gel de la commande en assurant la fiabilité du SC et garantit la sécurité humaine et matérielle (CHAPITRE VI).

PARTIE II.

LE CONTROLE DE COMMANDE.

CHAPITRE III.

UN OUTIL POUR LE CONTRÔLE DE COMMANDE.

INTRODUCTION.

Dans ce chapitre, nous ferons un tour d'horizon des outils de modélisation qui existent dans le domaine des SED, afin de mettre en évidence les avantages et les inconvénients de chacun d'eux. Nous pourrons ainsi dégager les outils qui répondent à nos besoins de modélisation. Nous nous contenterons de décrire les aspects importants de ces outils. Le lecteur pourra trouver les informations complémentaires dans les références bibliographiques.

Le but de cette description est de rechercher un outil de modélisation du module de contrôle de commandes. Notre recherche est axée sur un outil temporel, impératif, déterministe et réactif.

I. PROGRAMMATION DES SYSTÈMES TEMPS-RÉEL.

Les systèmes temps-réel se caractérisent par le temps de réponse. Le point central de ces applications, concerne leur capacité de réponse rapide, parfois urgente, à des sollicitations externes. Le terme temps-réel évoque implicitement l'existence de contraintes

temporelles. Ainsi, un résultat correct du système mais tardif, est considéré comme fautif. Ils doivent en plus, être capables d'assurer la coopération et l'échange d'informations, entre plusieurs sous-systèmes indépendants. Enfin, le système doit être fiable et maintenable pour assurer la productivité qui est la contrainte de base de l'utilisateur. Pour la mise en œuvre des applications parallèles et temps réels, nous distinguons trois familles de langages.

La première concerne les langages de programmation asynchrones de haut niveau, conçus pour le temps réel, et axés sur la structuration et le traitement des données. Parmi ces langages, on trouve MODULA, ADA, OCCAM,... Les sorties sont asynchrones par rapport aux entrées les ayant provoquées. MODULA et ADA sont basés sur la notion de tâche. Ils offrent des primitives de gestion des tâches (activation, arrêt, suspension) et des formalismes de synchronisation et de communication, basés sur le rendez-vous (ADA) et les sémaphores (MODULA). La manipulation du temps est très difficile. Ces langages sont mal adaptés à la programmation des systèmes réactifs (cf §II).

La deuxième famille est formée des exécutifs temps-réel. L'exécutif assure la coopération entre les tâches. Cette coopération se matérialise par l'échange d'informations entre tâches. L'exécutif doit alors disposer de mécanismes de communication, de gestion des tâches et de synchronisation. Parmi ces exécutifs, on trouve SCEPTRE [BONI, 84] et ELECTRE [CREUSOT, 89] :

- SCEPTRE est un noyau normalisé des exécutifs temps réel, qui exprime la volonté de normaliser la conception des exécutifs,
- **ELECTRE** est un **E**xécutif et **L**angag**E** de **C**ontrôle **T**emps-**RE**el. Il est destiné à exprimer les comportements temporels admissibles des tâches dans une application temps-réel. Le procédé physique est vu par ELECTRE comme un ensemble d'activités physiques.

La dernière famille se tourne vers de nouveaux langages, dits synchrones : ESTEREL [BERRY, 87], [BOUSSINOT, 91], [GONTHIER, 88], LUSTRE [CASPI, 87], SIGNAL [LEGUERNIC, 86] et STATECHARTS [HAREL, 87]. Ils sont dédiés à la programmation des systèmes de contrôle/commande temps-réel, et plus généralement à des systèmes réactifs. Ils visent à rendre la programmation indépendante d'une architecture d'implantation particulière. Ces langages délaissent l'hypothèse d'asynchronisme et la substituent par une hypothèse de synchronisme plus ou moins forte. Le modèle est rendu déterministe (l'asynchronisme est source de non déterminisme). Les sorties sont synchrones aux entrées, et le temps de calcul est infiniment petit. Ils offrent des primitives temporelles permettant une spécification temporelle du comportement. Néanmoins, ces langages se heurtent au problème

d'implantation lié à l'hypothèse de synchronisme. La solution proposée consiste à transformer un programme synchrone parallèle, par compilation, en un automate à états finis purement séquentiel équivalent.

Les langages synchrones sont le fruit de la recherche française, du CMA-INRIA pour ESTEREL, du LGI pour LUSTRE et de l'INRIA-IRISA pour SIGNAL. Ces communautés sont regroupées au sein du groupe C2A (Collaboration CAO Automatique) du pôle Automatique du CNRS. L'objectif de ce groupe est principalement la mise en place d'un socle commun afin de permettre une normalisation de ces langages. D'autres points sont abordés par ce groupe de travail comme la répartition des programmes synchrones, la spécification des exécutifs et l'expérimentation sur site.

Le domaine d'application de ces langages est assez large. Dans ce mémoire, nous nous intéressons particulièrement aux SFPM. Les aspects de réalisation et d'implantation ne seront pas traités (problèmes de nature trop technique).

II. SYSTÈMES RÉACTIFS.

D. HAREL et A. PNUELI, ont été les premiers à séparer en deux classes les systèmes programmés : transformationnels et réactifs [HAREL, 85]. La traduction de la définition qui a été donnée d'un système réactif est la suivante :

> *"Un système réactif désigne un système réagissant à des entrées provenant de son environnement en produisant lui-même des sorties vers cet environnement."*

Ces systèmes se caractérisent par l'importance des échanges d'informations par rapport aux calculs à faire. Ce sont des systèmes parallèles formés de composés concurrents soumis à des contraintes temporelles strictes [PHAN, 92]. Les contrôleurs des systèmes réactifs doivent réagir rapidement pour ne pas perdre d'information et présenter les réactions en temps utile [ANDRÉ, 92]. Ces contraintes sont celles des systèmes temps-réel. Nous représenterons ce genre de système (réactif) par des boîtes, appelées réactives. Nous faisons apparaître les interactions entre le système et son environnement (FIGURE III.1).

L'évolution des systèmes réactifs consiste en la séquence de trois étapes : stimuli, réaction et repos.

Les systèmes réactifs sont à opposer aux systèmes transformationnels. Les premiers (réactifs) ne fournissent pas ou très peu de traitement, alors que pour les seconds (transformationnels), les résultats qu'ils produisent sont essentiellement le fruit de calculs. Les systèmes réactifs se distinguent par leur dépendance de l'environnement et leur faible traitement des données. Les systèmes réactifs sont donc destinés aux systèmes de type réflexe. Le système doit réagir de manière instantanée à toute excitation extérieure.

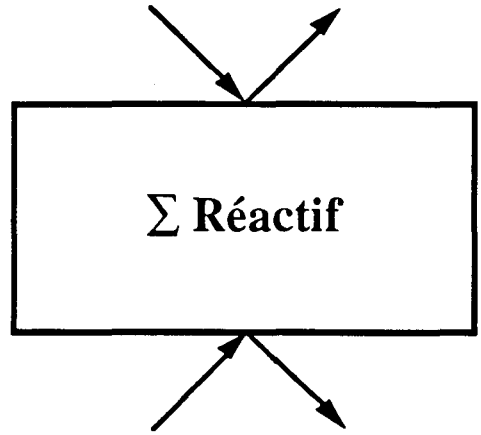


FIGURE III.1 : SYSTÈME RÉACTIF.

Malheureusement, les systèmes de production ne sont ni réactifs ni transformationnels, mais tiennent des deux à la fois. A cet effet, C. ANDRÉ [ANDRÉ, 91], propose une approche mixte. Cette approche n'est pas toujours envisageable, surtout pour les SED de type manufacturiers, étant donné, la quantité d'événements à gérer et la combinatoire qui en résulte.

Nous présentons les différents langages synchrones et nous discuterons notre choix pour la modélisation du contrôle de commande.

III. ESTEREL.

ESTEREL est un langage *synchrone orienté activité*. L'aspect synchrone du langage suppose que toutes les actions (communication, calculs, ...) sont réalisées de manière *instantanée*. L'écriture d'une application, sous ESTEREL, se fait par une description *temporelle* du comportement. Ce langage est de type *impératif*. Le développeur doit donc décrire, de manière explicite et précise, les relations entre les entrées et les sorties.

Ce langage intègre une nouvelle notion du temps : le *temps multiforme*. Cette notion fait abstraction totale de l'unité de temps physique. Chaque signal reconnu, définit une *unité* de temps. Un *intervalle* de temps est délimité par deux occurrences d'un même signal. La conséquence directe, est qu'il n'existe pas d'horloge *maître*. La synchronisation peut être faite par rapport à toute unité.

Une caractéristique fondamentale et importante de ce langage est sa capacité de *préemption*. Il permet en effet d'arrêter le traitement en cours. Cet aspect est possible grâce à l'hypothèse faite sur l'*instantanéité* du traitement.

Dans cette partie du mémoire, nous décrirons l'environnement de développement existant autour d'ESTEREL, l'organisation d'un programme, ainsi que les différentes primitives du langage.

III.1. HYPOTHÈSE DE SYNCHRONICITÉ.

Les langages synchrones rejettent l'idée de l'asynchronisme qui est source d'indéterminisme. Ils la remplacent par une hypothèse de synchronicité. En ESTEREL, cette hypothèse porte sur la correspondance séquence d'entrée/séquence de sortie. Elle permet d'avoir une description impérative du comportement du système.

Cette hypothèse est rendue plus forte, en supposant que les réactions sont synchrones aux entrées. Afin de répondre aux exigences de cette approche instantanée, il est nécessaire de disposer de machines infiniment rapides. Ce qui pose a posteriori, un problème de validité de cette hypothèse. Il est donc nécessaire et souhaitable de la vérifier.

Sur le plan de l'implantation, on se heurte à un autre problème. Il s'agit de l'asynchronisme de l'environnement extérieur. En effet, les événements reçus par un programme synchrone arrivent de manière asynchrone.

III.2. ENVIRONNEMENT DE DÉVELOPPEMENT.

Le développement d'une application synchrone en ESTEREL, se concrétise par la réalisation de trois étapes : l'analyse, le codage et la validation.

La première consiste à décrire les entrées, les sorties et le comportement temporel du cahier des charges. Ce comportement détermine la correspondance entre les séquences d'entrées et les séquences de sorties.

La deuxième étape se concrétise par la transcription de l'analyse. On est amené à déclarer les points d'entrées et de sorties. Ensuite, une écriture du comportement sous forme de primitives ESTEREL est élaborée.

La validation du programme ESTEREL peut se faire par trois moyens différents, qui sont mis à la disposition de l'utilisateur : la compilation, la preuve et la simulation.

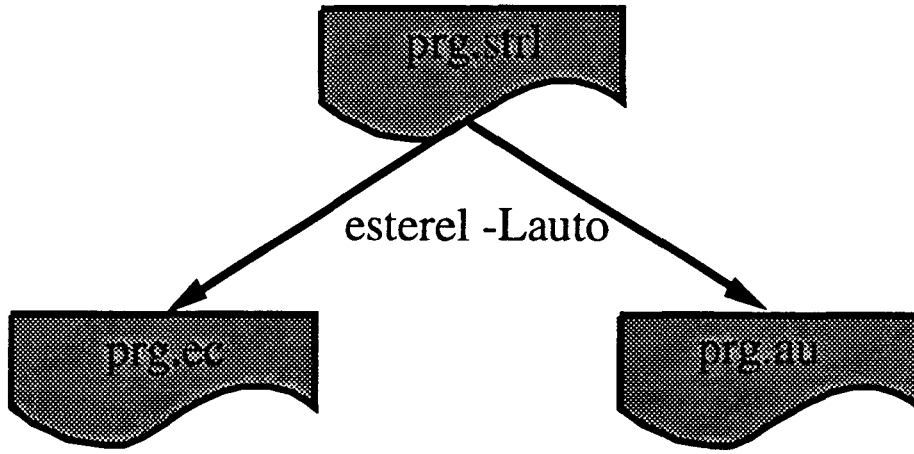


FIGURE III.2 : CONSTRUCTION D'UN AUTOMATE POUR AUTO.

La compilation permet la détection des cycles de causalité dans une application. Les inter-blocages conceptuels entre tâches sont ainsi éliminés. Il s'agit des cycles de causalité. Le code est ainsi traduit dans un langage hôte.

Les outils de preuves comme AUTO [VERGAMINI, 87] et AUTOGRAPH [ROY, 89] permettent de vérifier des propriétés des sorties d'un programme en fonction de ses entrées. Ces outils valident l'application ESTEREL compilée au préalable (FIGURE III.2). De plus, AGEL, le nouveau environnement de développement d'ESTEREL offre une simulation in vivo de l'automate [CHAILLOUX, 91B].

La simulation permet d'identifier les écarts de spécification. Le compilateur ESTEREL fournit un simulateur adapté à l'application (FIGURE III.3). Pour obtenir ce simulateur, il est nécessaire d'écrire en langage hôte la structure de données et les routines utilisées par l'application.

En fonction des entrées et de l'état de l'automate, le simulateur génère des sorties. Le concepteur a ainsi la possibilité de les observer et de localiser les écarts de comportement.

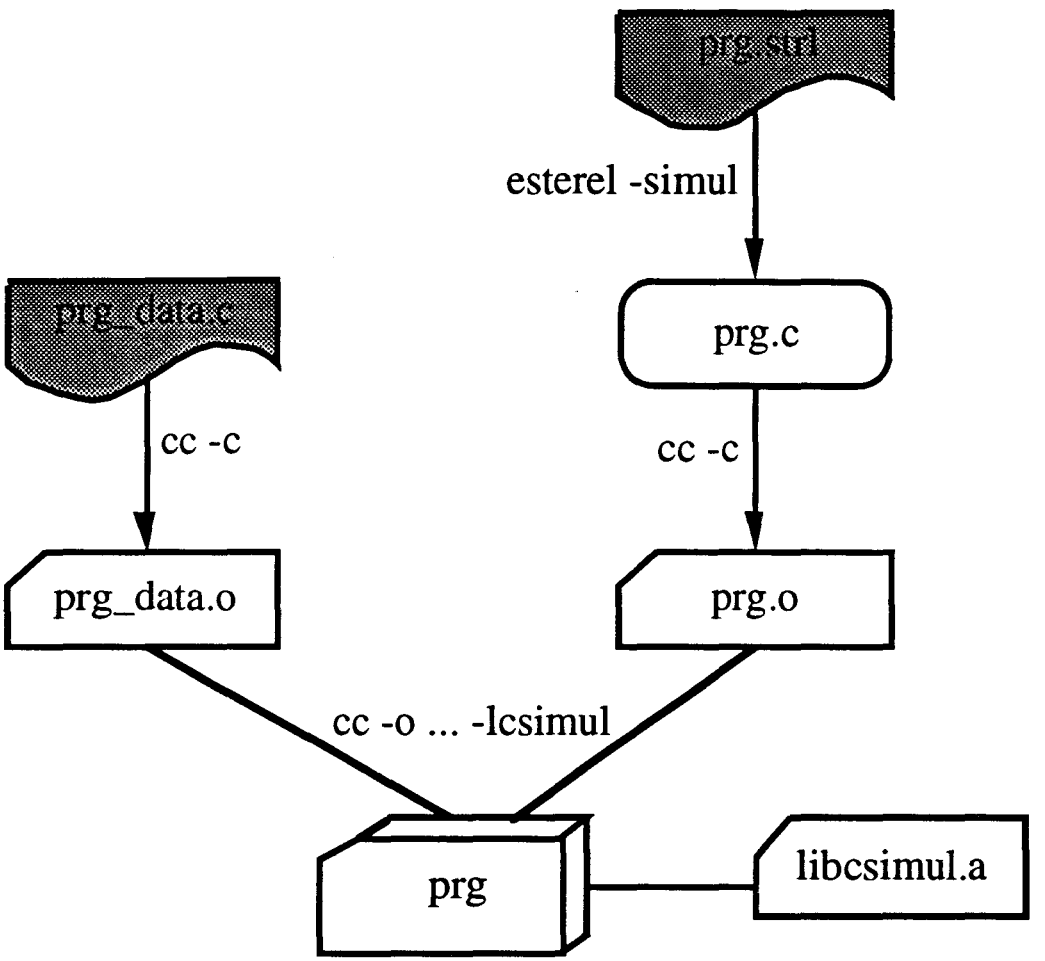


FIGURE III.3 : CONSTRUCTION D'UN SIMULATEUR D'UN PROGRAMME ESTEREL.

III.3. NOTION DE MODULE ET DE PROGRAMME.

Un module ESTEREL constitue l'unité de base (FIGURE III.4). Il se comporte comme étant un objet à part entière. Un programme (FIGURE III.5) est une composition de modules communiquant de façon synchrone.

La sémantique de communication est basée sur la diffusion instantanée de signaux. Cette sémantique apporte une facilité de modularité. Tous les modules ont une vision unique du système à tout moment. Cette idée s'apparente à celle de la diffusion radio. Tous les

récepteurs d'une station radio, ont une vision unique de l'environnement. En effet, la réception est instantanée, si l'on suppose que la célérité de l'onde est proche de l'infini.

Un module est composé de deux parties : déclarative et impérative. La partie déclarative concerne les signaux, les capteurs, les types et fonctions abstraits. La partie impérative est formée d'expressions classiques et temporelles. Cette partie dicte le comportement du module.

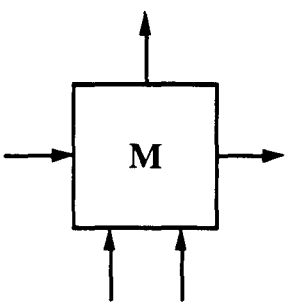


FIGURE III.4 : MODULE ESTEREL.

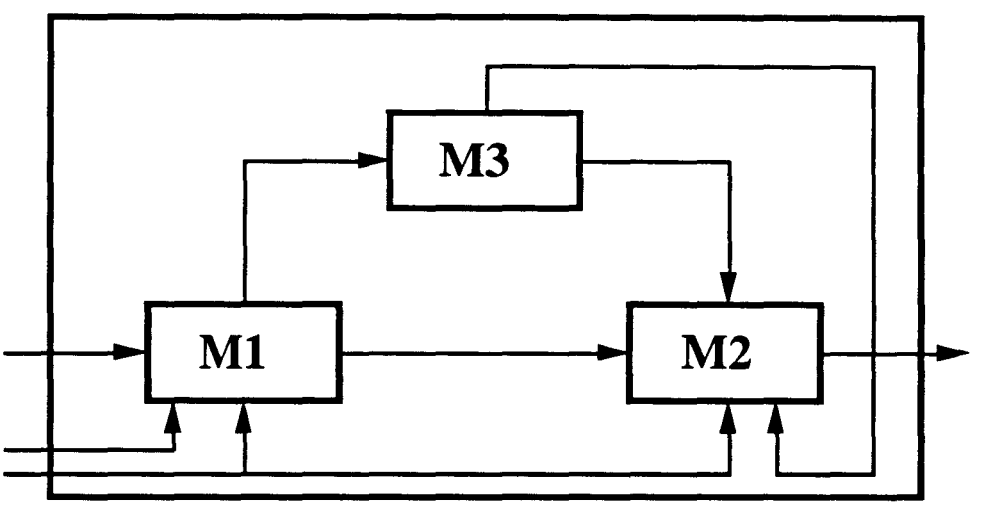


FIGURE III.5 : PROGRAMME ESTEREL.

La partie déclarative consiste en la déclaration des différentes ressources : partie hôte, partie locale et partie relationnelle. La partie hôte, concerne la déclaration des constantes, types, fonctions et procédures qui feront l'objet d'une interface. Ces éléments seront écrits dans un langage hôte (C, ADA, LELISP) et liés avec le code généré par le compilateur ESTEREL. La partie locale est constituée de la déclaration des signaux et des capteurs. La partie relationnelle consiste à définir les corrélations existantes entre signaux.

La structure d'un module en ESTEREL prend la forme suivante :

module <nom de module> :
% partie déclarative
% partie hôte
% partie locale
input <déclaration des signaux d'entrée>
output <déclaration des signaux de sorties>
inputoutput <déclaration des signaux d'entrées/sorties>
sensor <déclaration des capteurs>
% partie relationnelle

% partie impérative

. % fin de module

III.4. NOTION DE SIGNAL.

Un signal est un événement élémentaire (SECONDE, RESET). Les signaux peuvent être de quatre types : entrée (input), sortie (output), entrée/sortie (inputoutput), et signaux locaux. Nous disposons ainsi des mécanismes de *communication* élémentaires. Les signaux locaux assurent la communication inter-processus. Les occurrences d'un signal permettent la *synchronisation* des tâches.
Un signal peut être porteur d'une valeur. Il est dit *valué*. Un signal non-valué est qualifié de *pur*. Les signaux valués assurent l'échange d'information entre les modules ou avec l'environnement.

L'utilisation des signaux locaux est recommandée, puisqu'elle rend le programme plus clair. De plus, elle n'a aucune conséquence sur la taille de l'automate ou du code générés, par rapport à une programmation classique. Cependant, ils permettent d'expliciter les différents passages entre processus.

Dans le cas où le signal est de type inputoutput, il boucle sur le même module. A ce niveau, un problème d'identification du signal se pose, vue la diffusion instantanée. Ce type de problème concerne les signaux valués. Pour pallier ce problème, les concepteurs d'ESTEREL proposent au développeur de définir une fonction (*) de fusion, portant sur les valeurs du signal. Cette fonction doit être commutative et associative. A titre d'exemple,

prenons comme fonction de composition l'addition arithmétique, pour un signal S ayant comme domaine d'application des entiers. Ainsi, quand on sera en présence de deux valeurs V1 et V2, le signal portera comme valeur la somme de V1 et V2. Ce problème n'est pas lié spécifiquement aux inputoutput, puisque la diffusion simultanée d'un même signal valué par deux modules différents, génère une situation de conflit sur la valeur du signal.

La partie relationnelle permet de définir les relations entre les signaux. Ces relations peuvent être de deux types. La première sert à définir une exclusion entre deux signaux (symbole #). Ce qui implique que la simultanéité de ces deux signaux est impossible. La seconde permet de spécifier la simultanéité "unilatérale" de deux signaux (symbole ⇒). En effet, si on écrit S1⇒S2, alors l'apparition de S1 implique l'apparition simultanée de S2 et non l'inverse. A titre d'exemple, prenons le cas des signaux SECONDE et MINUTE. On peut alors écrire que MINUTE⇒SECONDE. Il convient d'être prudent lors de la déclaration de tels liens, puisque ces liens conditionnent très fortement l'interprétation du programme source par le compilateur. En effet, la spécification de ces liens permet une réduction considérable du nombre d'états de l'automate obtenu par compilation.

III.5. STRUCTURE DE CONTRÔLE.

ESTEREL offre les structures de contrôle d'un langage classique : la séquence, l'alternative et la répétitive. La séquence se matérialise par le symbole ";" qui sert de séparateur. L'alternative se construit grâce aux mots-clés **if, then, else** et **end**. L'exécution d'une telle structure est guidée par un prédicat booléen. La répétitive est assez pauvre, puisqu'il s'agit d'une boucle infinie.

Une instruction importante, liée aux systèmes concurrents, a été introduite : le parallèle (//). En effet, ce symbolisme permet la mise en parallèle de processus. Il permet en l'occurrence la synchronisation de processus. Il est indispensable de faire attention à l'utilisation du parallélisme. En effet, les branches parallèles peuvent occasionner une explosion du nombre d'états et des transitions de l'automate.
De plus, ESTEREL offre toute une gamme d'opérateurs logiques (**or, and, not**) et arithmétiques (**+, -, *, /, mod**), et des comparateurs arithmétiques (**<, <=, >, >=, =, <>**).

Une instruction d'appel de procédures et fonctions externes, est implémentée (**call**). Elle permet donc l'utilisation des langages classiques. Une instruction, assurant la

modularité et la simplicité d'écriture, est fournie (**copymodule**). Elle est paramétrable (signaux et constantes). Cette instruction ressemble à un appel de procédure classique, mais, en réalité, le compilateur effectue une copie du module concerné avec substitution des paramètres.

III.6. PRIMITIVES TEMPORELLES.

Ce langage est construit autour de primitives temporelles et de communication élémentaires. Il s'agit des instructions : **halt, emit, do...watching,** et **present**. A partir de ces primitives, il est possible d'exprimer tout comportement temporel.

L'instruction **halt** ne se termine jamais. Elle ne réalise aucun traitement. Cette instruction est à la base d'une instruction dérivée : *attente* de signal.

L'instruction **emit** permet l'envoi de signaux. Elle apporte ainsi le premier mécanisme de communication. L'émission peut être valuée. L'émission de S avec la valeur v, s'écrit alors comme suit : **emit S(v)**.

L'instruction **present** prend la forme d'une alternative. En effet, en fonction de la présence ou non de l'occurrence d'un signal dans l'événement courant, un traitement sera déclenché. La syntaxe de cette primitive est de la forme : **present** S **then** <inst1> **else** <inst2> **end**. Cette structure permet de spécifier les comportements exclusifs.

Enfin, nous avons l'instruction de préemption : **do** <inst> **watching** <occ>. En effet, c'est la primitive la plus forte de ce langage, car elle permet de "tuer" un traitement (<inst>) si l'occurrence (<occ>) se produit. La préemption peut se faire par l'apparition d'un signal ou par un compteur de signal.

Il est dommage qu'ESTEREL, n'offre pas la possibilité de préempter un traitement en fonction de la valeur d'un signal. En effet, il serait intéressant d'arrêter le traitement quand un seuil est atteint. Ce genre de comportement peut être résolu par la mise en place d'un processus parallèle qui se chargera d'émettre un signal indiquant que le seuil est dépassé. Ce peut être à titre d'exemple, un sprinkler (système d'extinction d'incendie). Le sprinkler est mis en marche, à partir du moment où la température atteint le seuil fixé.

ESTEREL propose des primitives temporelles dérivées construites à l'aide des primitives élémentaires. On trouve ainsi, le mécanisme de communication qui manquait : *attente* de signal. L'attente se fait par la primitive **await** qui est équivalente à **do halt watching** <occ>. Une structure d'attente multiple (**await case ...**) est implémentée. La priorité de branchement, en cas de simultanéité, est régie par l'ordre d'écriture des "case".

Le mécanisme de préemption est enrichi par l'ajout d'une clause **Timeout**. Elle permet de déclencher un traitement particulier en cas de préemption.

Des boucles temporelles ont été introduites pour expliciter le déclenchement périodique d'un traitement. Ces boucles préemptent le traitement, si le signal qui régule la boucle, apparaît.

Une primitive de traitement d'exception est proposée (**trap**). Les trap peuvent être paramétrés et recevoir ainsi des valeurs. Plusieurs possibilités d'exceptions dans un même traitement sont possibles.

III.7. EXTENSIONS.

Dans la version V3.0 d'ESTEREL, la notion d'horloge d'activation était inexistante. Par conséquent, il était impossible de coder des spécifications du genre :

> A chaque instant, Si le signal PRESENT n'est pas dans l'événement d'entrée, alors émettre le signal ABSENT.

La nouvelle version V3.20 apporte une solution à ce problème, en introduisant un signal d'entrée pré-défini *tick*. Ce signal toujours présent correspond à l'horloge d'activation du programme. Il devient alors possible d'écrire :

```
every tick do
        present PRESENT else emit ABSENT end
end
```

Une instruction corollaire de tick, *sustain*, permet de rendre un signal présent en permanence.

IV. LUSTRE ET SIGNAL.

SIGNAL et LUSTRE s'inscrivent dans la catégorie des langages synchrones déclaratifs. Le développeur est amené à spécifier les relations entre les données sous forme d'équations dynamiques : *orientés états*. Il décrit donc les fonctions, qui appliquées aux entrées fourniront les sorties. Ils permettent de prédire statiquement le comportement temporel du programme synchrone. Ils offrent aussi la possibilité de détection des contraintes sur les entrées et des blocages (deadlock).

Les programmes construits autour de ces langages, reçoivent et engendrent des suites infinies de valeurs. Ces valeurs sont typées. Les indices de ces valeurs ont une interprétation temporelle. Nous retrouvons dans ces langages la notion de flot de données (dataflow). Généralement, dans un langage flot de données, tout programme peut être caractérisé par un système d'équations dont les opérations sont continues, sur des domaines munis d'un ordre partiel complet. Les données produites par le programme forment le plus petit point fixe du système d'équations équivalent [PHAN, 92].

IV.1. NOTIONS ATTACHÉES AUX FLOTS DE DONNÉES.

IV.1.1. Horloge.

Les programmes écrits en SIGNAL ou LUSTRE, sont vus comme une boucle répétée infiniment, dont chaque pas est synchronisé sur une horloge globale. L'horloge globale est considérée comme une expression dont les valeurs sont booléennes. A toute notion d'horloge est associée une notion d'instant. Les instants de l'horloge déterminent les instants de sa propre horloge pour lesquels sa valeur est vrai. Une variable est représentée par une suite infinie de valeurs, et une suite d'instants associés appelés son horloge. Le temps est ainsi lié au rythme des données dans les flots. A un instant donné, le système a une vision globale des variables. Par conséquent, si l'horloge d'une variable prend la valeur faux, la variable est indisponible. On peut donc énoncer la définition suivante :

> *"Une horloge est une expression booléenne dont les instants sont ceux de sa propre horloge pour lesquels sa valeur vaut vrai."*

Ces langages équationnels flots de données, vérifient trois principes fondamentaux (la causalité, la substitution et la définition) et imposent des contraintes vis à vis des opérateurs et des opérandes.

IV.1.2. Le principe de causalité.

Ce principe stipule que le terme de rang n d'une suite ne dépend que des termes de rang inférieur ou égal à n des entrées. Le terme de rang n d'une variable donnée X peut s'écrire de la forme :

$$Xn = f(X0,...,Xn,Y0,...Yn,Z0,...,Zn,...)$$

IV.1.3. Le principe de substitution.

Ce deuxième principe, énonce que l'écriture de X=E signifie qu'il y a identité totale entre la variable X et l'expression E à tout instant.

IV.1.4. Le principe de définition.

Ce dernier principe affirme que le contexte d'utilisation d'une expression E ne peut avoir aucune influence sur le comportement de cette expression. En particulier, aucune information ne peut être inférée sur les entrées. Ainsi, X=E définit X comme étant la suite des valeurs de l'expression E.

IV.1.5. Opérateurs et opérandes.

Les fonctions prédéfinies ou définies sont considérées comme des opérateurs instantanés. Les opérandes d'un même opérateur doivent absolument être synchronisées sur la même horloge (présence simultanée de tous les opérandes). Ce qui permet de garantir la réalisation de l'opération (recherche de point fixe d'un système d'équations).

IV.2. LUSTRE.

Un programme LUSTRE est un ensemble (nœud) de déclarations qui font appel éventuellement à d'autres ensembles. Toute variable ou expression LUSTRE est un flot sur une horloge logique binaire : la valeur est soit présente (True), soit absente (False). Ce qui implique nécessairement une "horloge de base".

Les opérateurs temporels sont au nombre de quatre regroupés en deux classes. La classe des opérateurs de base permet de manipuler implicitement les instants. Cette classe est formée de deux opérateurs : **pre** (précédent) qui sert à mémoriser la valeur d'une

expression à l'instant t-1, et → (suivi de) qui permet de définir la valeur suivante d'une expression et en particulier une valeur initiale. La deuxième classe concerne les opérateurs multi-synchrones : **when** (quand) qui sert à filtrer un flot par un flot booléen et **current** qui permet la projection instantanée d'une expression sur l'horloge la plus rapide.

La modularité s'exprime par la notion de nœud. Un nœud est un opérateur créé par le développeur et introduit par le mot réservé **node**. Un programme est vu comme un réseau d'opérateurs communicant par les flots d'entrées et de sorties.

La compilation assure le contrôle de la cohérence des types et des horloges, ainsi que l'absence de blocages [GLORY, 90]. Elle produit un code "OC (Object Code)" commun à ESTEREL et LUSTRE. LUSTRE, a été utilisé dans le domaine de la description des propriétés de sûreté de logiciel vérifiables par des méthodes de démonstration (outil LESAR) du type "model checking".

IV.3. SIGNAL.

SIGNAL est relativement proche de LUSTRE. Il diffère avant tout du point de vue de l'horloge associée à un signal. En effet, l'originalité de SIGNAL réside dans le calcul effectif d'horloge dans le corps de Galois (Z/3Z). Les notions de VRAI, FAUX et ABSENT, sont plongées dans Z/3Z et sont respectivement associées aux valeurs 1, 2 (ou -1) et 0[1]. Le calcul effectif d'horloge, permet de détecter les erreurs de synchronisation et d'organiser le contrôle du programme.

Les objets manipulés sont des signaux. Les opérateurs temporels sont au nombre de quatre : le retard ($), le sous-échantillonnage (**when**), l'entrelacement (**default**) et la composition. D'autres opérateurs peuvent être définis à partir de ceux-ci. Il est possible de faire opérer des éléments n'ayant pas la même horloge par le biais d'opérateurs temporels (composition) et non fonctionnels (calcul).

La modularité en SIGNAL s'exprime au travers des processus qui sont l'équivalent des nœuds de LUSTRE. Les flots d'entrées et de sorties doivent être synchrones.

---

[1] Si "a" est un signal, alors "a^2" peut être considéré comme son horloge (au sens de Lustre). Si a est associé à VRAI ou FAUX alors a vaut 1 ou -1 et a^2 vaut 1. Sinon, a est associé à ABSENT, alors $a=a^2=0$.

Le compilateur vérifie la cohérence globale des relations entre signaux. Le calcul d'horloges explicite celles-ci pour permettre au compilateur de synthétiser une horloge plus rapide que toutes les autres : horloge de l'environnement.

L'utilisation principale de SIGNAL se situe dans les domaines continus du traitement de SIGNAL (segmentation de la parole) et de l'automatique des systèmes à états continus (industrie laitière).

V. STATECHARTS.

Les STATECHARTS est un formalisme graphique destiné à la description comportementale des systèmes complexes. HAREL [85 ET 87], les présentent comme étant une extension des automates à états finis et de leur mode de représentation (les diagrammes états-transitions).

Les principales extensions portent sur trois notions fondamentales [SAHRAOUI, 91]:

- la hiérarchisation des états (profondeur),
- l'expression du parallélisme (orthogonalité),
- la communication par diffusion.

Un diagramme STATECHARTS est un graphe dont les sommets représentent les états du système, et les arcs les transitions entre états. Les arcs sont étiquetés par l'événement déclencheur de la transition. Les états sont symbolisés par des rectangles à coins arrondis (FIGURE III.6). Ce modèle est enrichi par certaines notions. Il s'agit de :

- l'état initial : état d'entrée à l'activation,
- l'historique : mémorisation du dernier état actif,

Les STATECHARTS sont considérés comme un langage hiérarchique. Ils offrent deux possibilités de constructions. La première de type **OU** (FIGURE III.6A) permet de spécifier des états exclusifs d'un même STATECHART. L'autre construction (orthogonale) permet de mettre en évidence l'aspect parallèle de l'application : construction de type **ET** (FIGURE III.8).

Le passage d'un état à autre se fait par les événements qui étiquettent les arcs. Quand un état est actif, et un événement étiquetant un arc sortant de cet état est présent, l'état actif après franchissement, est l'état pointé par ce même arc.

Le diagramme de la FIGURE III.6 décrit un STATECHART à deux états *exclusifs*. L'événement α permet de passer de l'état A à l'état B.

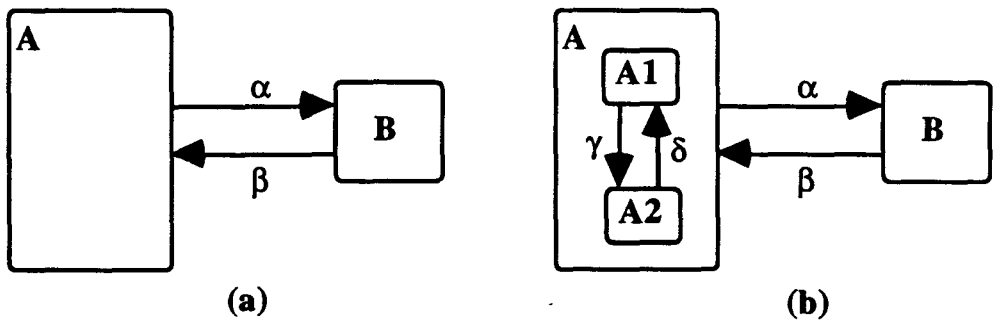


FIGURE III.6 : CONSTRUCTION DE TYPE OU (A) ET HIÉRARCHIE (B).

Supposons que l'état actuel est A. Si α et β sont présents simultanément, alors l'état final est A en passant par B (cf plus loin la diffusion).

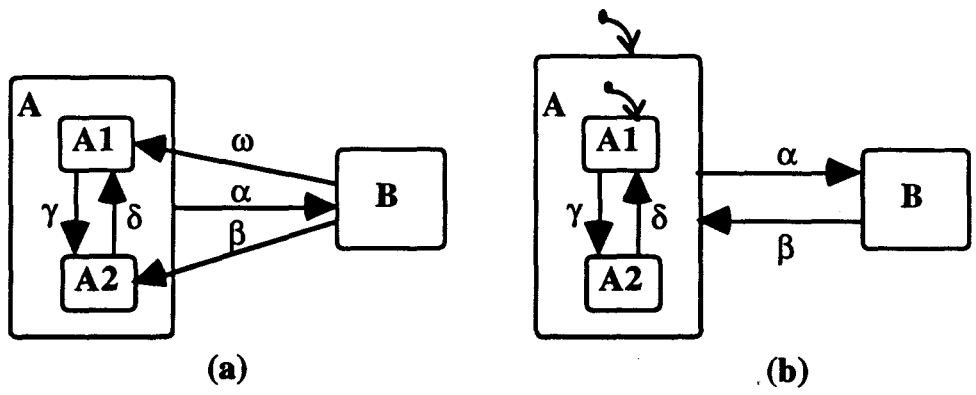


FIGURE III.7 : ETAT PAR DÉFAUT.

La hiérarchie permet de procéder à une spécification par affinement (FIGURE III.6). Ainsi, un état (A) représente un regroupement d'états (A1 et A2) ayant des propriétés communes. La hiérarchie n'altère en rien le comportement du diagramme. Cependant, elle rend la description souple (affinement), et fine (niveau de hiérarchie). La hiérarchie peut donc s'adapter aussi bien à une approche descendante (affinement) qu'à une approche ascendante (abstraction du comportement).

L'événement α (FIGURE III.7) déclenche le passage vers B, quelque soit le sous-état actif de A. Nous avons donc une agrégation des transitions A1 vers B et A2 vers B en une

seule transition. Ce concept permet d'alléger le graphe. Il est à noter que les transitions ne se font pas obligatoirement avec le même niveau d'abstraction. Il est ainsi possible d'avoir accès directement à des niveaux plus fins (FIGURE III.7A). A titre d'exemple, l'événement β déclenche la transition de B vers A2.

La notion d'état par défaut (FIGURE III.7B) a été introduite afin de déconnecter un état de son environnement. De plus, elle formalise l'état initial à activer. Ce mécanisme illustré par la FIGURE III.7B montre que la "flèche pointée" arrivant sur A désigne l'état initial entre A et B. A l'intérieur de A, l'état initial (A1) est spécifié par le même formalisme.

Notons la notion d'historique qui permet de réactiver le dernier état actif du groupement. Il est symbolisé par un arc se terminant sur un cercle contenant la lettre H. Il est ainsi possible de mémoriser l'état. Cette notion pourrait être d'une grande utilité en cas de reprise.

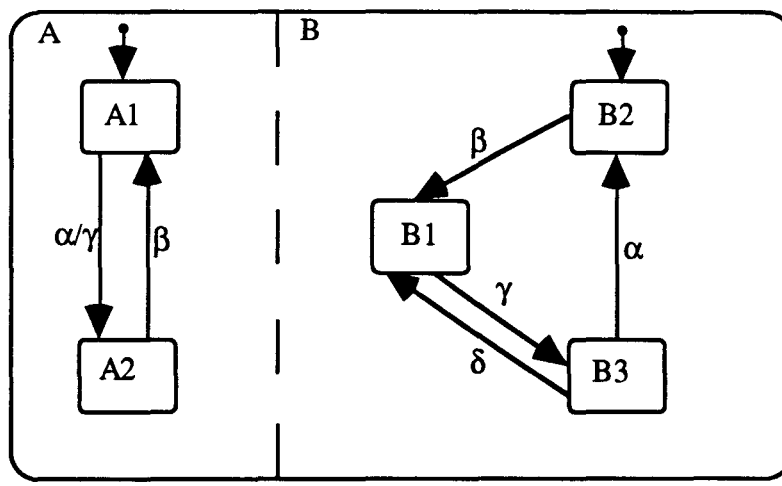


FIGURE III.8 : CONSTRUCTION DE TYPE ET.

Le diagramme de la FIGURE III.8 indique que le STATECHART A et le STATECHART B sont en parallèle. Le parallélisme est symbolisé par les pointillés scindant un groupement d'états en deux. Ce qui engendre une non connexité des ensembles d'états. Chacun des STATECHARTS possède un et un seul état actif à un instant donné. L'état A et B forment implicitement un état du niveau supérieur.

Une description équivalente (FIGURE III.9) aux deux diagrammes parallèles (FIGURE III.8) consiste à réaliser le produit des deux automates et à résoudre le synchronisme au niveau des actions. Notons ici, la possibilité d'une explosion combinatoire

du nombre d'états de l'automate produit, dans le cas d'un fort parallélisme.

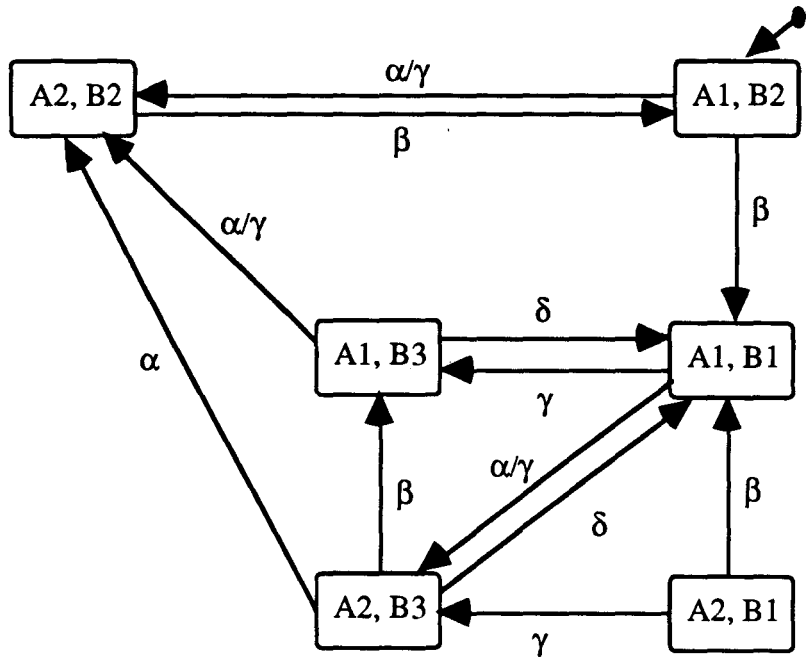


FIGURE III.9 : MISE À PLAT DE DEUX DIAGRAMMES PARALLÈLES.

L'évolution d'un STATECHART est définie par une sémantique associée aux transitions qui prend la forme suivante : **événement(s)/action(s)**

L'évolution d'un diagramme STATECHART est donc conditionnée par les événements engendrés par son environnement : entrées. Un événement présent est vu de la même manière à tout niveau de description (diffusion instantanée). Les actions activées par les transitions sont vues comme des sorties et deviennent à leur tour des entrées pour d'autres STATECHARTS. La dualité événement/action permet de synchroniser des processus parallèles : une action émise est considérée comme un événement (réaction en chaîne). Supposons que pour le diagramme de la FIGURE III.8, les états actifs soient A1 et B1. Quand l'événement α sera présent, alors les états actifs deviennent A2 et B3. La transition de A1 vers A2 engendre l'événement γ qui active la transition de B1 vers B3, dans ce même instant.

La synchronisation des processus peut être réalisée par référence à l'activation d'un état (primitive **IN(état)**).

On remarquera qu'il n'existe aucune séquence d'événements mettant le diagramme de la FIGURE III.9 dans l'état (A2, B1) : inaccessibilité de cet état.

La construction de type ET est très efficace, néanmoins des problèmes de blocages peuvent en résulter. Ils sont, de plus, très difficiles à localiser. Afin de valider le modèle, une mise à plat est nécessaire. Prenons le cas de la FIGURE III.8. Remplaçons l'événement β par in(A1), pour le passage de B2 à B1. Cette modification amène deux transformations sur le modèle mis à plat : disparition de la transition (A2,B2) vers (A1,B2) et transformation de l'étiquette de l'arc (A1,B2) vers (A1,B1) qui devient in(A1). Cette transformation pose deux problèmes :

- l'état (A2,B2) devient un état puits (blocage),
- un asynchronisme est constaté au niveau de la transition (A1,B2) vers (A1,B1) (problème de synchronisation).

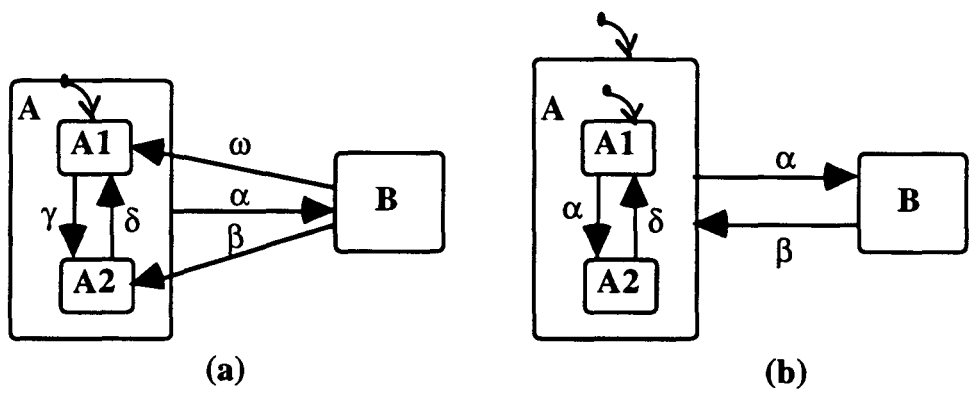


FIGURE III.10 : EXEMPLES D'INDÉTERMINISME.

Outre les problèmes de blocages et de synchronisation, les indéterminismes sont fréquents. Sur la FIGURE III.10A, l'état actif étant B, l'événement β met le modèle dans un état indéterministe : est-ce que l'état actif est A1 (par défaut) ou A2 (état activaté par la transition). Un autre cas d'indéterminisme peut être illustré par l'exemple de la FIGURE III.10B. Supposons que l'état actif soit A, et le sous-état actif est A1. L'apparition de l'événement α engendre un indéterminisme.

Cet outil étant de nature non déterminisme, n'est pas, par conséquent, adapté à notre éxigence de forte synchronicité, qui conduit à la nécessité du déterminisme du modèle retenu. Indiquons, de plus que les STATECHARTS sont plus orientés vers les tâches de spécification et qu'il est encore difficile de prouver de bonnes qualités de comportement de

ces modèles et d'en déduire une possibilité d'implantation évitant a priori toute erreur de transposition. Notons, l'intérêt qui est porté à ce dernier aspect, et qui a conduit à un début de développement d'outil d'aide dans ce sens. Il s'agit de l'outil STATEMATE [VALLOTTON, 91].

CONCLUSION.

Nous nous situons dans le cadre d'un système à événements discrets de type réactif. Le but de notre application, est de pouvoir détecter les défaillances dès leur apparition. Cette détection se fait par le traitement des événements générés par le procédé. Considérons cette citation [CHAILLOUX, 91A] : *"On les (les langages synchrones) trouve dans les systèmes où la réaction à un événement extérieur est primordiale ..."* Notre choix portera donc sur un langage synchrone. De plus, un programme en langage synchrone est considéré comme une description du comportement attendu d'un système réactif [ANDRÉ, 92] : approche événementielle.

Le modèle que nous recherchons doit posséder les caractéristiques suivantes (cf CHAPITRE IV) :

- expression explicite du temps,
- le comportement doit être déterministe,
- le modèle doit avoir un comportement réactif.

Parmi les langages, notre choix s'est porté sur ESTEREL, puisque c'est l'outil qui répond le mieux à nos exigences. En effet, il est particulièrement apte à programmer les systèmes à événements discrets qui doivent réagir à des séquences de stimuli [ANDRÉ, 92].

Notre choix est confirmé par cette citation de J. CHAILLOUX, puisque l'outil en question est destiné à rendre le système de commande fiable et sûr de fonctionnement :

> *"Ce langage (ESTEREL) offre aux développeurs d'applications aux contraintes temporelles fortes, un formalisme rigoureux sur des fondements mathématiques prouvés, et apporte enfin une solution aux problèmes les plus cruciaux de ce type d'applications, c'est-à-dire la sûreté et la fiabilité de fonctionnement." [CHAILLOUX, 91A]*

CHAPITRE IV.

LA CONCEPTION DU CONTRÔLE DE COMMANDE.

INTRODUCTION.

Dans ce chapitre, nous nous intéresserons au Contrôle de Commande (CC), et plus particulièrement aux aspects conception et mise en œuvre du système de contrôle.

En premier lieu, nous exposerons le concept de base du contrôle, ainsi que l'approche choisie pour sa mise en œuvre.

Dans un second temps, nous présenterons l'organisation du système de contrôle, ainsi que les raisons qui nous ont conduit à une telle organisation.

Dans un troisième temps, nous aborderons la démarche de conception. Notamment, les différentes étapes du cycle de conception, depuis la spécification jusqu'à l'implantation, seront présentées. Nous illustrerons notre approche à l'aide d'un exemple.

En conclusion, les apports de cette approche, ainsi que les perspectives d'évolution du système de contrôle, seront exposés.

I. PRÉSENTATION DU MODULE DE CONTRÔLE DE COMMANDE.

La fonction principale du Module de Contrôle de Commande (MCC) est d'assurer la *fiabilité* du SC, d'accroître la *disponibilité* matérielle. La fiabilité d'un système définit le degré d'*accomplissement* de sa mission dans des délais prescrits sans défaillance.

Nous proposons d'exposer le principe du contrôle, pour conférer au système de production cette propriété de fiabilité. Nous aborderons par la suite, le problème de la mise en œuvre de cette idée : approche choisie. Enfin, nous terminerons ce volet, par une discussion sur les besoins de la réalisation de l'approche.

I.1. IDÉE GÉNÉRALE.

Le MCC aura pour mission d'accroître la fiabilité de fonctionnement du SC et la disponibilité matérielle.

Pour satisfaire cette proposition, nous proposons d'introduire de la redondance. En effet, la construction d'un système de commande fiable passe par l'incorporation de *redondances explicites*. Le caractère important de cette redondance est confirmé par cette citation du *Dr D. Lardner* dans l'article "*Babbage's calculating engine* " :

> *«Le contrôle le plus sûr et le plus efficace des erreurs qui surviennent dans le processus de calcul consiste à faire les mêmes calculs par des calculateurs indépendants et séparés, ce contrôle est rendu encore plus décisif s'ils (les calculateurs) effectuent leurs calculs selon des méthodes différentes» [AVIZIENIS, 79].*

Cette remarque s'adresse aux systèmes informatiques. Néanmoins, elle peut être adaptée aux SED de type manufacturiers, en considérant un "*processus de fabrication*" au lieu d'un "*processus de calcul*". Intéressons-nous plutôt au caractère de la *redondance* qui est mis en relief dans cette citation. On relève l'idée principale selon laquelle, le même traitement doit être réalisé par des organes *indépendants* et *séparés*. Dans notre domaine d'application, une pièce ne pourrait être usinée deux fois. En conséquence, nous proposons de simuler le comportement du procédé pendant que celui-ci réalise l'usinage. Nous utilisons ainsi, des

moyens différents et indépendants. Cette approche reste en conformité avec la citation puisque nous proposons une méthode différente : l'une est informatique (simulation du traitement) alors que l'autre est réelle (réalisation effective).

Comme, nous l'avons proposé ci-dessus, cette citation peut être adaptée. En effet, nous pouvons énoncer que :

> *«le contrôle le plus sûr et le plus efficace des erreurs qui surviennent dans un traitement, consiste à réaliser le même traitement par des organes indépendants et séparés (la simulation d'un traitement est considérée comme étant identique à ce traitement). Ce contrôle est rendu plus décisif si les organes en question effectuent leurs traitements selon des méthodes différentes».*

Nous sommes donc, amenés à contrôler la réalisation des actions par le procédé, par une simulation du comportement. Cette simulation est basée sur l'idée de redondance. Nous distinguons deux types de redondances, *explicite* et *implicite*. Cette différenciation est très importante, puisque l'une est volontaire alors que l'autre est due à une mauvaise conception.

Le rôle de la redondance explicite est d'empêcher qu'une *erreur* ne conduise à une *défaillance*. Par opposition, la redondance implicite peut avoir un effet identique mais inattendu. Cette "fausse" redondance doit être éliminée puisqu'elle peut masquer des erreurs et évite donc, que des *erreurs latentes* ne surviennent de manière intempestive [LAPRIE, 85].

En résumé, nous proposons de construire un système redondant qui permet de *détecter les écarts de fonctionnement* du procédé. Il faut souligner que ces écarts sont liés à la réalisation des actions. Par conséquent, le système de contrôle est en relation directe avec le système de commande. Notre but est de construire un système de *détection réactif*. En effet, la *réactivité* permet la détection instantanée des défaillances, et confère au système la possibilité d'*éviter* ainsi une *propagation de la défaillance* d'un composant vers d'autres composants. Cette caractéristique rend le diagnostic plus facile et plus rapide, puisque le composant défaillant est "*pré-défini*".

I.2. APPROCHE PROPOSÉE.

Pour répondre à ce besoin de réactivité, nous avons choisi l'approche *synchrone*. Nous utiliserons donc, une simulation du comportement du procédé par la *programmation synchrone* [BERRY, 85; CASPI, 87; LEGUERNIC, 86] (cf Chapitre III, pour une

description de ces langages). Les caractéristiques d'une telle programmation, nous ont conduit à adjoindre au système de commande un *module réactif redondant.*

La raison principale de ce choix, est basée sur la capacité des langages synchrones à exprimer le *parallélisme* et la *synchronicité*. D'après *G.* BERRY dans son article "*Programmation synchrone des systèmes réactifs*" [BERRY, 87] :

> *«L'hypothèse de synchronisme se heurte pourtant à une objection sérieuse : elle n'est pas directement implémentable ... le tout est de rapporter la notion de synchronisme à l'utilisateur final du système, ou, dans le formalisme de MEIJE [BOUDOL, 85], à son observateur : si sa perception est que le système fonctionne "comme si" les entrées étaient synchrones aux sorties, alors l'hypothèse est justifiée».*

L'hypothèse de synchronicité permet de rendre les langages *déterministes* [BERRY, 87]. Nous aurons ainsi un système *réactif.* Rappelons qu'un système réactif (terme introduit par A. PNUELI) désigne des *systèmes réagissant à des entrées provenant de façon répétitive de leur environnement en produisant eux-mêmes des sorties vers cet environnement* [CARDELLI, 85].

Le contrôle réactif permet d'avoir une détection instantanée des erreurs de fonctionnement. Cette caractéristique d'instantanéité du modèle généré confère au système de production deux propriétés nécessaires à son bon fonctionnement :

- *fiabilité* (tolérance aux fautes avec recouvrement),
- *disponibilité* (évitement des erreurs de propagation à l'issue d'une défaillance).

Les entrées sont captées directement sur le procédé. C'est une méthode qui dépend donc de la disposition et de la nature des capteurs. Pour que cette méthode puisse être sûre, il est nécessaire de supposer que les capteurs soient non défaillants : si on veut remédier à cette insuffisance, on sera amené à utiliser des sources d'informations différentes (un autre type de redondance).

Parmi les langages synchrones, SIGNAL [LEGUERNIC, 86], LUSTRE [CASPI, 87], ESTEREL [BERRY, 85; BERRY, 87], nous avons choisi de travailler en ESTEREL pour diverses raisons. Tout d'abord, ce langage n'impose pas d'unité de temps (notion de *temps multiforme* [BERRY, 87]), ce qui donne une certaine puissance d'expression. De plus, il est doté de *primitives temporelles* assez puissantes : *préemption, attente (même multiple)*

d'événements, chien de garde, traitement d'exception [BERRY, 87]. Un autre caractère important de ce langage est son orientation *activités* (SIGNAL et LUSTRE sont orientés états). D'autre part, il existe un environnement de développement autour d'ESTEREL : outils de preuve et de simulation graphique [ROY, 89; VERGAMINI, 87]. Ce qui nous permet de valider et de simuler notre modèle plus facilement, afin de vérifier sa conformité avec la description qui en a été faite.

I.3. RÉALISATION.

L'envoi d'une commande au procédé active de manière *simultanée* le contrôle de la commande ainsi que le processus physique, associé à celle-ci. D'une part, nous assurons la gestion de la réalisation physique de l'action. D'autre part, le MCC simulera le comportement associé à cette réalisation.

Les actions émises par le système de commande sont d'abord *filtrées* par le Module des Filtres de Commande. Ainsi, nous serons assurés de la conformité de la commande avec l'état réel du procédé : on qualifiera la commande comme étant *validée*.

Le système de commande envoie donc des consignes au procédé physique pour lui imposer un comportement particulier afin de répondre à une demande de l'utilisateur. Pour un système à événements discrets, une commande (une séquence d'actions) consiste donc à faire évoluer le procédé, d'un état vers un autre, selon une séquence d'actions prédéterminée.

A ce niveau, une question se pose : quelles sont les informations nécessaires pour avoir le meilleur contrôle ? Pour chaque commande élémentaire, trois éléments doivent être contrôlés :

- le bon déroulement de la commande,
- les conditions de sa réalisation,
- et ce, dans des délais fixés au préalable (Chien de garde).

En effet, ces éléments conduisent à une vision assez fine du comportement de la commande. Ils seront l'objet d'une spécification du procédé.

Illustrons cette proposition par un exemple simple. Prenons le cas du transfert d'une pièce palettisée d'un poste Pi à un poste Pj. Le transfert est réalisé par un convoyeur à

activation continue (FIGURE IV.1). Chacun des postes est équipé d'une butée, permettant l'arrêt de la palette, et d'un capteur. Ce capteur nous renseigne sur l'état de la butée (fermée ou ouverte). Le poste Pi est muni, en sortie, d'un capteur C_Pi_S_po, qui sert à nous informer du positionnement de la palette. Le poste Pj est équipé, en entrée, d'un capteur détectant le passage d'une palette : C_Pj_E_pa.

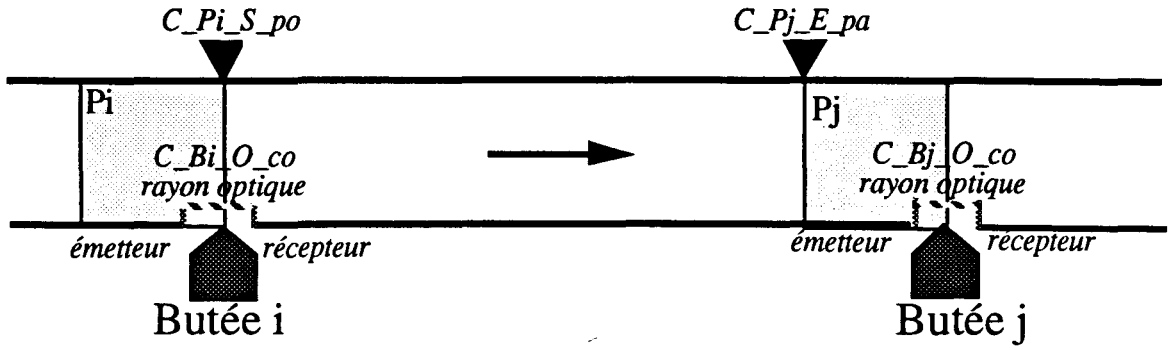


FIGURE IV.1 : UN EXEMPLE DE PROCÉDÉ.

Le transfert d'une palette de Pi à Pj est implicitement lié à la commande de la butée. En effet, en ouvrant la butée Bi, la palette est libérée et commence donc son cheminement vers le poste Pj. En parallèle, il faut lancer la commande fermeture de la butée Bj afin de stopper la palette au poste suivant. Il reste néanmoins à spécifier une grandeur importante : le temps nécessaire au transfert de la palette depuis Pi jusqu'à Pj. Nous supposerons que ce temps a pour valeur, N incréments de temps.

Après cette description du comportement du procédé, nous pouvons effectivement nous intéresser au contrôle du transfert.

Le bon déroulement du transfert est conditionné par la bonne réalisation des commandes des butées. Les conditions du transfert consistent en la vérification de l'état des différents capteurs. En effet, C_Pi_S_po doit être à UN, au début du transfert, et à ZERO, le reste du temps. Alors que, C_Pj_E_pa doit être à ZERO pendant tout le transfert et à UN à la fin de celui-ci. Le temps de transfert doit être le même que celui écoulé entre le passage de C_Pi_S_po à ZERO et le passage de C_Pj_E_pa à UN. Pour avoir une meilleure compréhension du comportement du procédé, nous proposons un chronogramme de ces événements (FIGURE IV.2).

Ce chronogramme, permet de décrire le comportement normal du procédé. Afin de pouvoir affirmer que le transfert s'est réellement bien déroulé, il est nécessaire de prendre en

compte une autre information, qui consiste à nous renseigner sur la position de la palette. Cette information est obtenue par la consultation d'un capteur, placé à la sortie de Pj, du type C_Pj_S_po. Nous considérons que tout écart du comportement défini par le chronogramme, est non conforme aux spécifications. Dans ce cas, une erreur de fonctionnement est générée. Supposons par exemple, que l'événement de transfert (passage de C_Pi_E_pa à un) n'apparaît pas alors que N est écoulé.

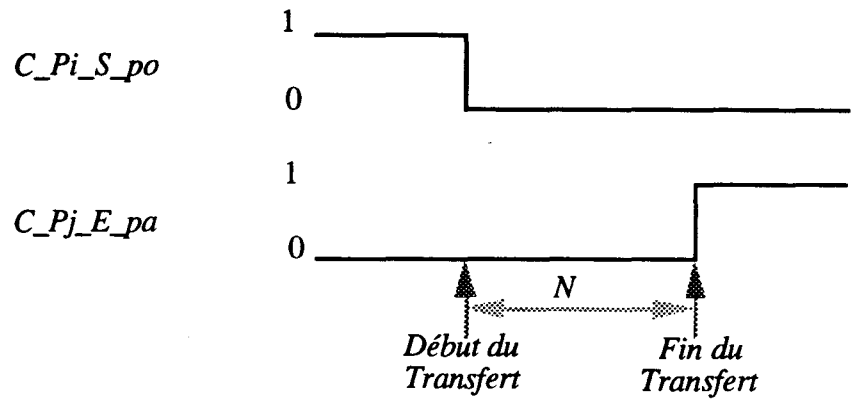


FIGURE IV.2 : CHRONOGRAMME DES ÉVÉNEMENTS.

Le *diagnostic* de l'origine de l'erreur est l'étape qui doit suivre toute détection. Nous cherchons ici à définir quelle est la *cause* qui a engendré cette erreur. Cet aspect du traitement d'erreurs n'est pas abordé dans ce mémoire. Les travaux de A.K.A. TOGUYENI [TOGUYENI, 90], traitent ce problème.

II. ORGANISATION DU MODULE DE CONTRÔLE.

II.1. DÉCOMPOSITION EN SOUS MODULES DE CONTRÔLE.

Toute action active, de manière simultanée, un processus de fabrication et un contrôle de celui-ci. Cette action est adressée à un composant physique unique. De ce fait, nous associons à chaque composant un Sous Module de Contrôle (**SMC**) : *modularité* de l'approche.

Prenons par exemple, le cas de deux butées d'un convoyeur d'un atelier flexible. Le contrôle est générique pour chacune des butées. Cependant, les deux contrôles ne seront pas regroupés. Nous aurons donc, deux modules de contrôle, chacun sera associé à une butée et une seule.

Cette décomposition [ELKHATTABI, 91 ET 92B] (FIGURE IV.3), permet d'éviter l'inconvénient majeur des langages synchrones, en effet, la mise à plat du parallélisme rend la combinatoire des automates générés, exponentielle. Il est donc plus intéressant d'avoir, si possible, dès la conception des modules séparés et répartis. On évite ainsi toute explosion *combinatoire* du modèle. Cette décomposition amène donc, une *réduction du nombre d'états* de chaque automate caractérisant le fonctionnement des différents composants.

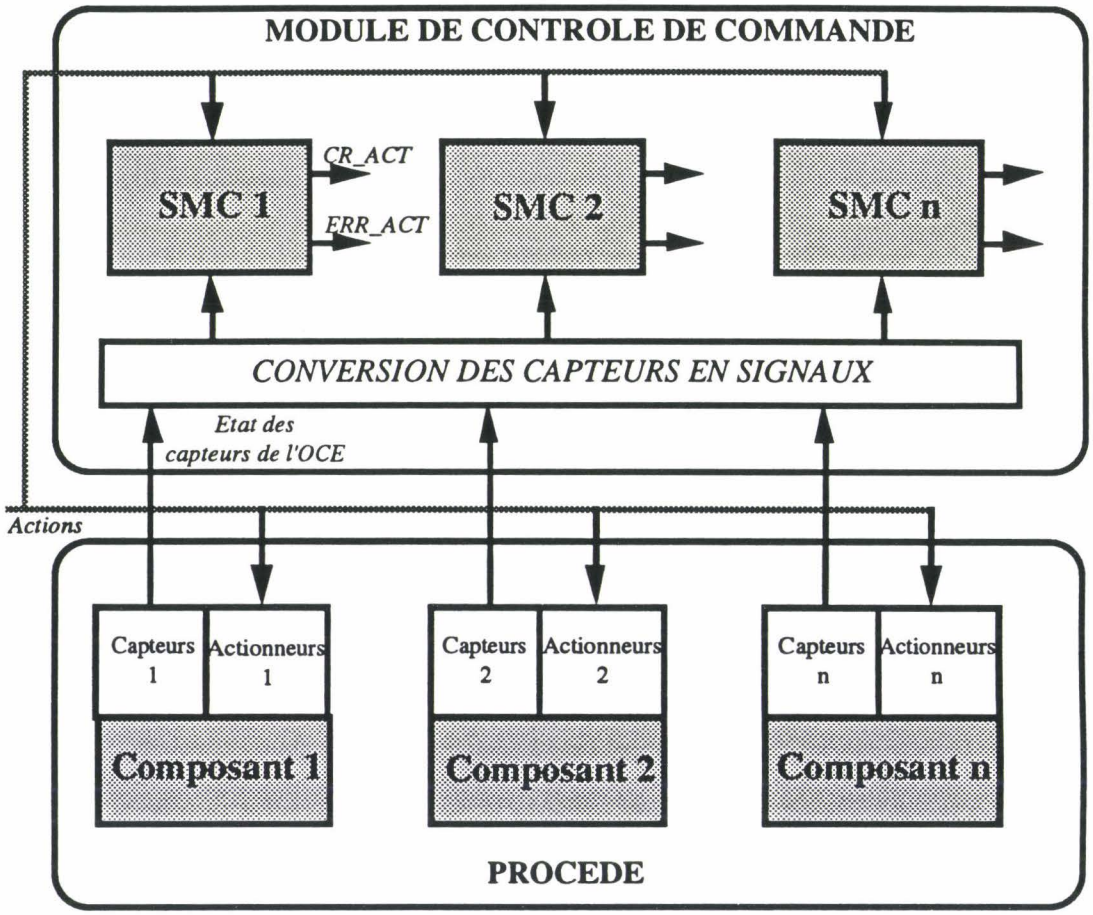


FIGURE IV.3 : INTERACTIONS ENTRE SMC ET COMPOSANTS.

D'autres avantages incontestables de cette approche, commme la modularité et la généricité, permettent d'avoir des modèles faciles à générer, à valider et à simuler.

Cette décomposition ne pose aucune difficulté puisque les composants sont *indépendants* entre eux, du point de vue des actions, conduisant à une indépendance entre les SMC correspondants. C'est la principale caractéristique qui a motivé cette approche. En effet, elle permet de rendre la réactivité du modèle possible.

II.2. CAPTEURS/ACTIONNEURS.

Pour chacun des composants, il est nécessaire d'identifier les *actionneurs* et les *capteurs*, qui lui sont associés. Il faut donc avoir une connaissance fine des différents composants d'un atelier. Les capteurs ont pour mission de traduire l'état du système en évaluant des grandeurs physiques. Ils permettent ainsi une interprétation du comportement du système. Ils constituent alors, des *comptes-rendus bruts*. Les actionneurs, quant à eux, ont un rôle dual. Ils convertissent les *consignes*, en provenance du système de commande, en grandeurs physiques compréhensibles par le procédé.

A titre d'exemple, la mise d'un bit à UN par le système de commande, activera le positionnement d'un aiguillage.

Les capteurs et les actionneurs constituent ainsi une interface entre le procédé et le système de commande. Ils permettent de traduire les consignes du système de commande en actions physiques (actionneurs) et l'état du procédé (capteurs) en compte-rendu compréhensible. Les comptes-rendus ne sont pas interprétés. Les SMC procèdent à une interprétation de l'état du système. En fonction du contexte, les SMC génèrent soit un *compte-rendu* de bon fonctionnement, soit une *erreur*. L'information ainsi générée est transmise au MFC.

Il est à noter que le choix des capteurs est très important, puisque c'est grâce à eux que le comportement doit pouvoir être analysé, en vue d'une détermination immédiate de dysfonctionnement. Il faut donc choisir les bonnes grandeurs à mesurer permettant ainsi au système de contrôle d'avoir les bonnes informations pour juger du fonctionnement du procédé. Il est par ailleurs inutile d'en avoir trop.

II.3. CONVERSION DES CAPTEURS.

La conversion des valeurs des capteurs en signaux, est une nécessité du fait du langage utilisé (ESTEREL). D'une part, ce langage ne permet pas de *préempter* un traitement par le changement de valeur d'un capteur mais par la réception d'un signal. D'autre part, les capteurs ne peuvent qu'être consultés.

Suite à ces deux remarques, et aux besoins futurs de procéder à la préemption de traitement, il est donc nécessaire de convertir l'état des capteurs en signaux au sens

d'ESTEREL. Nous distinguons deux types de capteurs : les capteurs *purs* et les capteurs *valués*. Les capteurs purs ne portent aucune valeur.

Pour ce qui est des capteurs purs, ils sont de nature *booléenne*. Quand un capteur est à ZERO, alors le signal correspondant est absent. A l'inverse, le signal est présent pour un capteur à UN. Ce type de signal est pur (non valué).

Pour ce qui des capteurs valués, ils portent la *valeur* de la grandeur mesurée. Nous devons donc traduire ce type de capteurs par des signaux valués. La présence du signal représente l'activation du capteur. La valeur du signal associé à un capteur activé est celle mesurée par celui-ci. L'algorithme de conversion de l'état des capteurs par des signaux est donné par la FIGURE IV.4A.

Cet algorithme comporte deux boucles. L'une synchronisée sur le signal tick et l'autre régulée par la liste des capteurs. A chaque pas, on consulte l'état d'un capteur de la liste. Si le capteur est actif, alors on émet le signal correspondant. De plus, si le capteur est valué, la valeur mesurée sera portée par le signal correspondant.

A chaque tick **faire**
  Pour chaque Capteur **faire**
    Si Capteur pur **alors**
      Si Capteur activé **alors**
        émettre signal associé
      Sinon émettre signal associé avec valeur du capteur

FIGURE IV.4A : ALGORITHME DE CONVERSION.

Le signal tick est toujours présent [CHAILLOUX, 91A]. La boucle principale est donc infinie. La scrutation infinie des capteurs permet de renseigner le système de contrôle de tout changement intervenant au niveau du procédé.

Cet algorithme en pseudo-code est transcrit en langage ESTEREL. Le module synchrone faisant l'objet du module de conversion est représenté par la FIGURE IV.4B. Les capteurs en ESTEREL sont forcément typés. Ils seront déclarés de type booléen pour les capteurs purs. La valeur d'un capteur pur activé est "true". Le test se fera donc par rapport à cette valeur, pour les capteurs purs.

```
module CONVERSION_DES _CAPTEURS_EN_SIGNAUX :
    output Si(type);     %liste des signaux associés aux capteurs
    sensor Ci(type);     %liste des capteurs à convertir

every tick do
%Conversion des capteurs purs
    if ?C1 then emit S1 endif
    •••
%Conversion des capteurs valués
    emit Sj(?Cj)
    •••
end
.
```

FIGURE IV.4B : MODULE DE CONVERSION DES CAPTEURS EN SIGNAUX.

II.4. SOUS-MODULE DE CONTRÔLE.

Un sous module de contrôle est associé à chaque composant ou à un groupe de composants dépendants. La dépendance des composants est une conséquence des contraintes de fonctionnement du système modélisé.

En effet, si l'on considère un bras et une pince de Robot, on s'aperçoit que ces deux composants sont dépendants. Le déplacement du bras implique le déplacement de la pince. Ces deux composants seront donc considérés dans un même SMC.

Un SMC est concrétisé par la mise en place d'un *module* ESTEREL. Ce module reprend les spécifications de fonctionnement du composant. Il traduit ainsi, le fonctionnement de ce dernier, sous forme *temporelle*. ESTEREL est basé sur des primitives temporelles : **await** (attente de signal) et **emit** (émission de signal). En effet, les signaux sont le cœur de tout programme synchrone en ESTEREL, puisque ce langage est destiné à modéliser les *systèmes réactifs*.

En résumé, cette approche, basée sur les langages synchrones, permet d'avoir des modules de contrôle *indépendants*, *répartis* et *déterministes*. Les modèles obtenus sont des *automates à états finis*.

Après avoir abordé le problème de l'organisation du module de contrôle, nous allons nous intéresser à la conception de celui-ci. Nous proposons à cet effet une méthodologie de conception des différents sous-modules qui le composent.

III. CONCEPTION DES SMC.

Dans cette partie, nous nous intéressons à la *méthodologie de conception* des SMC [ELKHATTABI, 91 ET 92B]. Cette méthodologie prend en charge le *cycle* de conception depuis la *spécification* jusqu'à la phase finale d'implantation sur site.

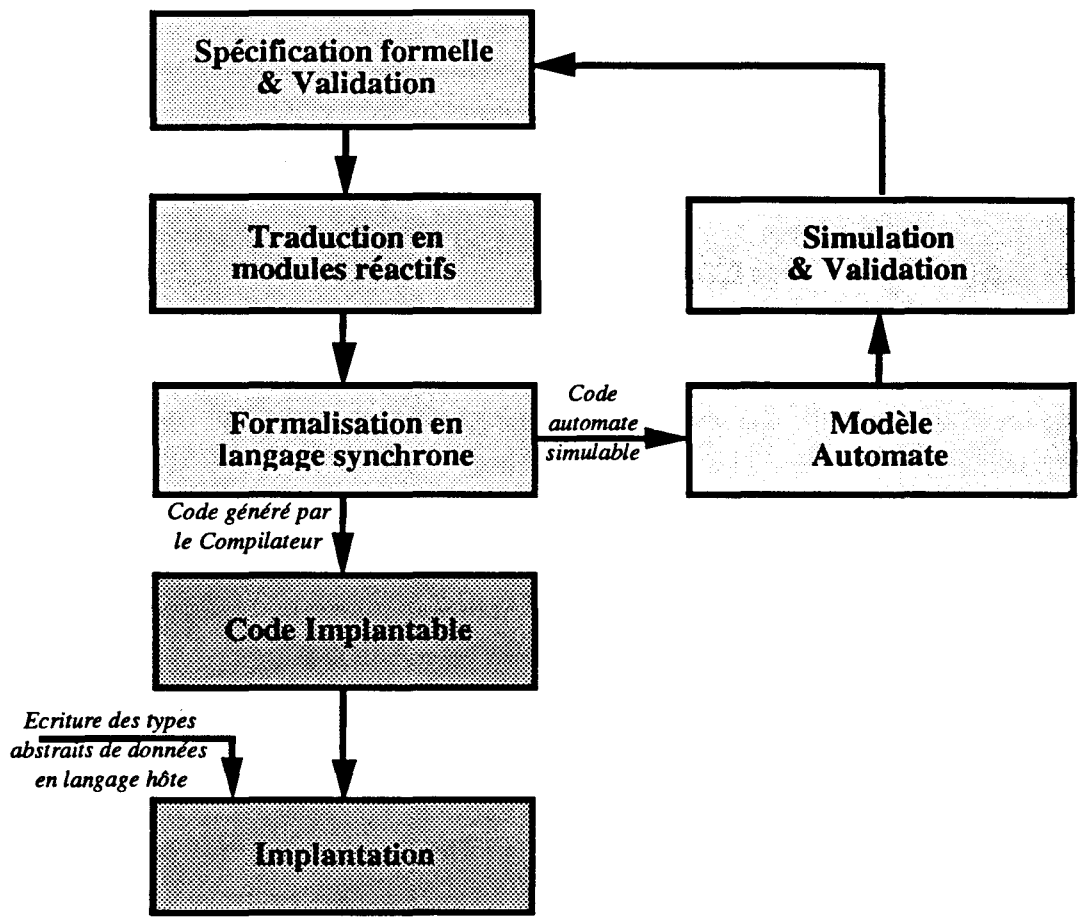


FIGURE IV.5 : MÉTHODOLOGIE DE CONCEPTION DES SMC.

La mise en œuvre d'une approche de conception nécessite une bonne connaissance du problème à modéliser, et des outils à utiliser. Notre but est donc de proposer une méthode de conception *systématique*. Nous proposons de développer les différentes étapes qui composent cette démarche.

La première étape de toute démarche de conception est évidement la spécification du problème à modéliser. Le but étant bien sûr de fournir un modèle implémentable. Dans le même esprit des travaux développés au LAIL dans le cadre de CASPAIM, la démarche de conception doit donc permettre le passage de la spécification à l'implantation.

Le cycle de vie se concrétise par l'élaboration de quatre phases : la spécification, la modélisation, la simulation et l'implantation.

III.1. LA PHASE DE SPÉCIFICATION.

Cette phase a pour objectif de fournir une description formelle validée du cahier de charges. En réalité, cette étape dépend de la modélisation structurelle du procédé physique, qui est notre point de départ. En effet, nous nous intéressons, ici à des composants élémentaires. La *décomposition structurelle* sera enrichie, en spécifiant, pour chacun des composants, les *actions* qu'il peut réaliser, les *conditions* et les *délais* de leur réalisation.

Cette étape est *rigoureuse* et *formelle*. Elle peut donc être validée par des outils de preuve. La validation de la spécification permet d'*éviter* des erreurs de spécification dès le début du cycle.

III.1.1. Présentation du modèle conceptuel.

Afin d'avoir une *spécification formelle* et cohérente, nous utilisons un modèle de données normatif et formalisé. Ce modèle *conceptuel* comporte trois concepts interdépendants, qui constituent les types de base de toute expression d'un schéma conceptuel de système d'information. Il s'agit des concepts *objet*, *opération* et *événement* [ROLLAND, 82 ET 86].

Un *objet* est un constituant concret ou abstrait du système modélisé. Une *opération* est une action qui peut être exécutée isolement et qui modifie l'état d'un des objets. Un *événement* est la constatation d'un changement d'état d'un objet par un détecteur. Il déclenche l'exécution d'une ou de plusieurs opérations et provoque ainsi un nouveau changement d'état. Le modèle est normatif, parce qu'il impose des contraintes dans le mode d'expression des représentations. La représentation normalisée retenue conduit à modéliser les objets, les opérations et les événements dans une forme *élémentaire* qui n'est plus *décomposable*. Cette forme élémentaire minimise le nombre des *inter-relations* entre les

éléments, aide à révéler d'éventuelles incohérences, lacunes ou redondances implicites, et conduit à une représentation complète, et condensée.

Les trois concepts sont définis formellement, dans le formalisme relationnel de CODD [CODD, 70], et intègrent des contraintes de normalisation. La rigueur de définition de ces concepts rend possible la *vérification stricte des qualités* d'un schéma conceptuel qui décrit les aspects *statiques* et *dynamiques*.

Les aspects *statiques* sont modélisés par des *objets* représentant les *entités* et les *associations d'entités* du modèle réel, qui sont développés suivant le formalisme de YOURDON [YOURDON, 89]. Ce formalisme dans sa dernière version [COAD, 92], intègre des contraintes de normalisation (exclusion, ...) et des relations de généralisation-spécialisation entre entités (approche objet).

Les aspects *dynamiques* sont modélisés par les *opérations* modélisant les actions élémentaires sur un des objets du système et par les *événements* exprimant les changements d'états élémentaires qui déclenchent l'exécution des opérations.Nous distinguons trois types d'événements :

- *externes* qui représentent l'arrivée ou le départ de message extérieur au système modélisé,
- *internes* qui rendent compte des changements d'états des objets dans le système,
- *temporels* qui traduisent des changements temporels déclenchant des opérations.

III.1.2. Spécification et validation d'un composant.

Tout composant physique sera représenté par un *modèle statique* (FIGURE IV.6). Une action n'est associée qu'à un seul composant. Un composant peut être activé par plusieurs actions. A chaque action est associée une liste de capteurs à UN, disjointe de la liste des capteurs à ZERO.

Ce modèle doit vérifier un certain nombre de propriétés pour être validé :

- La *consistance*,
- L'*unicité* de l'association Action → composant,
- L'*exclusion* de la liste des capteurs à UN et capteurs à ZERO pour une même action,

- L'*existence* nécessaire d'*au moins* une action pour tout composant spécifié,
- L'*existence* nécessaire d'au *moins* un capteur à UN pour tout composant spécifié (idem pour les capteurs à ZERO).

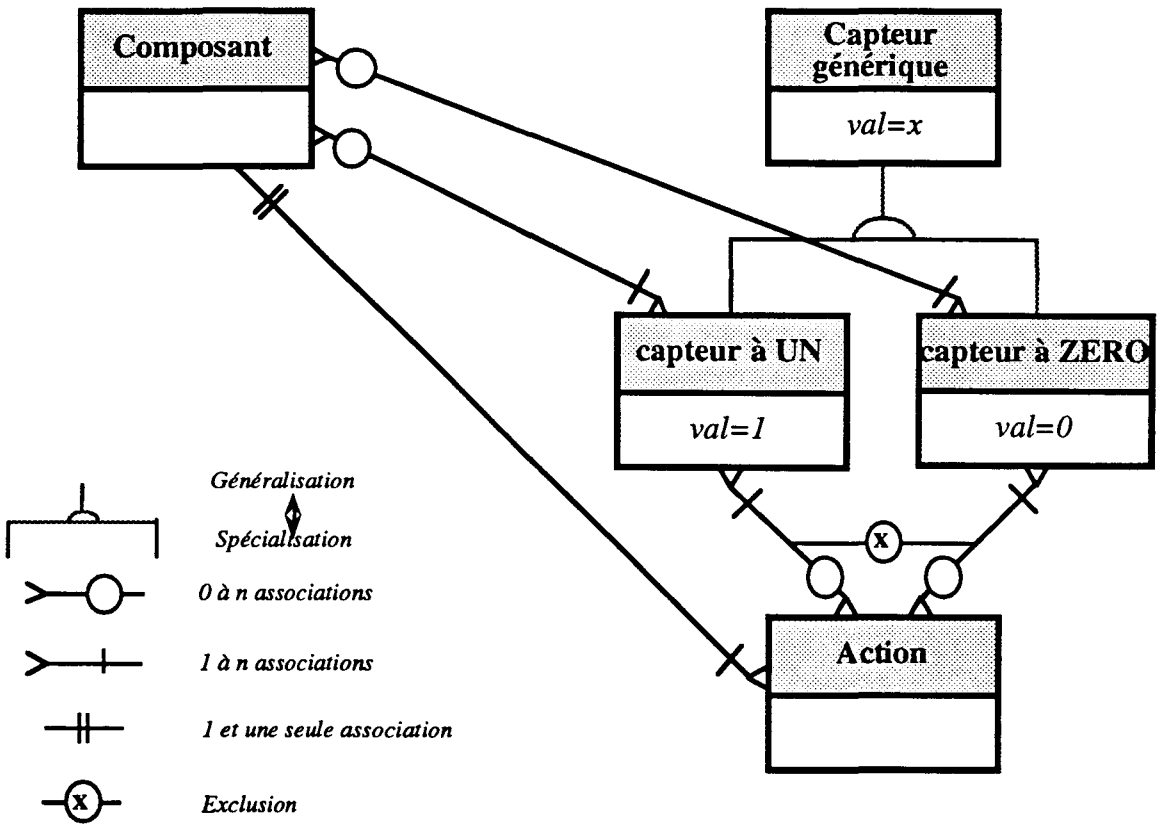


FIGURE IV.6 : SPÉCIFICATION STATIQUE DES SMC.

La validation de la spécification du système à modéliser, permet de passer à la phase de modélisation. Si l'étape de validation détecte une incohérence, un retour vers la spécification utilisateur, sera opéré. Ainsi, les incohérences de spécification peuvent être éliminées.

III.2. LA PHASE DE MODÉLISATION.

La phase de modélisation consiste à transformer les spécifications en *modules synchrones.*

En premier lieu, chaque composant est représenté par une *boîte réactive* (FIGURE IV.7), mettant en évidence ses points d'entrées et sorties. Cette représentation

permet aussi de mettre en relief les différents composants qui sont en liaison : contraintes de fonctionnement. En effet, prenons deux composants d'un Tour numérique : la broche et le sas. La fermeture du sas est conditionnée par le serrage de la broche. Ces interactions sont ainsi mises en évidence.

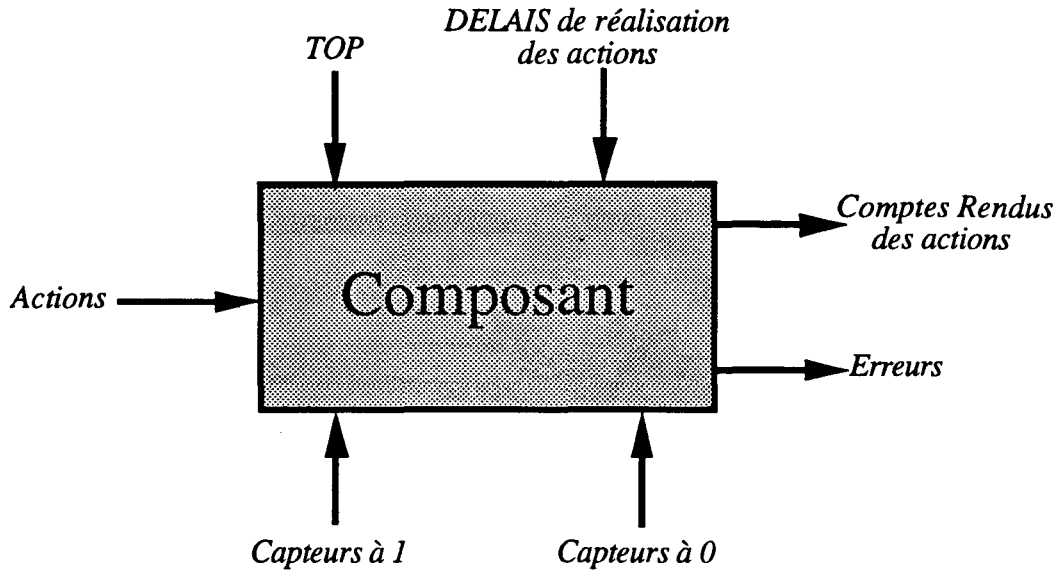


FIGURE IV.7 : MODÈLE RÉACTIF.

Dans un second temps, chacune des boîtes réactives sera traduite en langage cible (ESTEREL) : à chaque composant est associé un *programme générique* qui le modélise. La traduction consiste à puiser dans une bibliothèque le modèle correspondant au composant physique. Néanmoins, il est toujours possible de modifier ou d'enrichir le modèle synchrone, puisqu'en réalité, l'ossature du programme proposé est suffisante.

Les modules synchrones, écrits en ESTEREL, génèrent deux types de résultats suivant la compilation adoptée (Cf. FIGURE IV.5) :

- un *code automate* qui peut être *simulé* et *validé*,
- un code automate *implémentable*.

Le premier type de code, sert essentiellement à faire de la simulation et de la validation. D'une part, la simulation du modèle permet de vérifier sa conformité aux spécifications. D'autre part, la validation passe en revue les propriétés mathématiques du modèle.

III.3. LA PHASE DE SIMULATION.

Cette phase a comme point d'entrée, les automates générés par la compilation des modules synchrones. Elle est une étape nécessaire pour la mise en évidence des *écarts de comportement* du système modélisé.

Durant cette phase, nous procédons à une *validation formelle* des propriétés de l'automate généré. Nous utilisons à cet effet, un outil de preuve appelé AUTO [VERGAMINI, 87]. Cet outil permet de valider les propriétés des sorties d'un programme en fonction de ses entrées. Il permet ainsi de valider la description du programme ESTEREL.

La simulation du modèle consiste en l'exécution du simulateur généré par le compilateur ESTEREL. Elle est interactive et pas à pas. A chaque pas de l'exécution, une séquence de signaux est saisie. Le système peut donc évoluer. Les résultats consistent en l'énumération des éléments suivants : état actif, signaux présents et signaux attendus.

Cette simulation permet en effet, de passer en revue, les différentes possibilités du comportement du modèle. Nous pouvons ainsi valider la conformité du modèle avec la description utilisateur.

Après validation et simulation du modèle obtenu, deux possibilités se présentent :

- le modèle est valide formellement et son comportement est conforme aux attentes du concepteur, il est alors, possible d'envisager l'implantation du modèle.
- le modèle n'est pas valide ou non conforme à la spécification, alors, une remise en cause de la phase de spécification ou de la modélisation est envisagée.

III.4. LA PHASE D'IMPLANTATION.

La phase de simulation permet de valider le modèle généré. L'implantation devient alors possible. Cette phase a comme points d'entrée les modules synchrones validés. Nous effectuons une compilation de ces modules en langage hôte (C, ADA, LeLisp, ...). Le code ainsi obtenu, doit être complété par les interfaces écrites en langage hôte. En l'occurrence, il s'agit des procédures de calcul ou de traitement, et des interfaces de communication.

A ce niveau, le problème de la répartition des modèles se pose. Il est à noter que l'organisation du module de contrôle qui a été choisie, permet la prise en compte de cet aspect dès le début de la conception. Nous obtenons donc, des modèles déjà répartis. Deux possibilités d'implantation s'offrent à nous : machines distribuées avec superviseur ou une seule machine. Ce choix de répartition sera fonction du support informatique du système de commande et de son organisation [CRAYE, 89], [RIAT, 92].

En général, l'architecture matérielle est descendante, où le support informatique est formé d'un organe de pilotage et d'un ensemble d'automates programmables industriels. Nous nous retrouvons ainsi dans le premier cas. Ces modèles seront donc implantés sur des machines réparties. Quel serait donc le critère de répartition ? Vue l'organisation du MCC, où chaque SMC est associé à un composant physique, il serait donc judicieux d'implanter le SMC sur le même support que sa commande.

Nous terminons ce chapitre par la présentation d'une application complète de cette démarche.

IV. EXEMPLE.

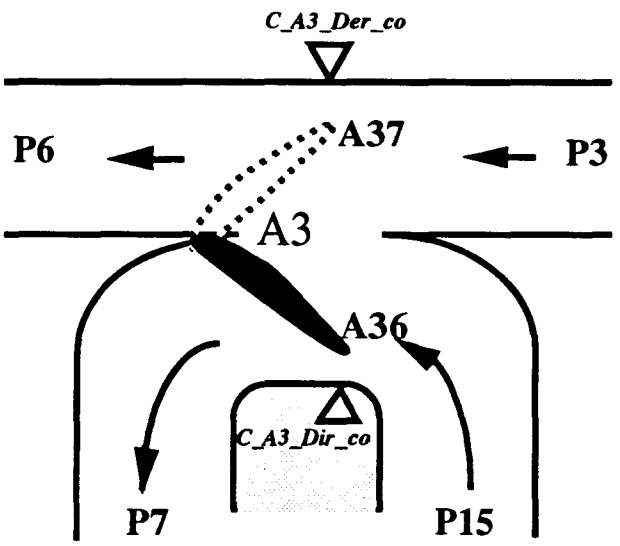


FIGURE IV.8 : EXEMPLE D'AIGUILLAGE.

Illustrons notre méthodologie de conception par un exemple. Considérons un aiguillage d'un atelier flexible (FIGURE IV.8).

L'exemple représente un aiguillage de l'atelier flexible de l'Ecole Centrale de Lille. L'action A36 (respectivement A37) permet d'aiguiller les palettes vers le poste de travail P6 (resp. vers P7). La provenance des palettes est soit P3, soit P15.

Pour chaque position (correspondante à une action) de l'aiguillage, un capteur à contact permet de valider cette position : C_A3_Dir_co pour l'action A36 et C_A3_Der_co pour l'action A37. Inversement, ces capteurs permettent de détecter la non-réalisation des actions : C_A3_Der_co à 1 pour l'action A36.

IV.1. SPÉCIFICATION DE L'EXEMPLE.

De la description précédemment donnée de l'aiguillage, nous pouvons dresser le tableau suivant :

Actions	Composant	Capteurs à 1	Capteurs à 0	Temps d'action
A36	A3	C_A3_Dir_Co	C_A3_Der_Co	N
A37	A3	C_A3_Der_Co	C_A3_Dir_Co	N

TABLE 1 : SPÉCIFICATION DE L'EXEMPLE.

Dans ce tableau, on voit apparaître les différentes actions avec leur spécifications respectives. On remarquera que ces actions sont duales. Cette spécification est conforme aux objectifs du modèle normatif qui a été présenté en IV.1. Le temps de réalisation de ces actions est du même ordre.

L'algorithme de conversion des capteurs en signaux est donnée par la FIGURE IV.9.

IV.2. MODÈLE RÉACTIF DE L'EXEMPLE.

Le résultat direct de la spécification est sa transposition en boîte réactive (FIGURE IV.10). On voit apparaître, en entrée, les différentes actions réalisables par le

composant, la liste des signaux associés aux capteurs à UN et à ZERO pour chacune des actions et le délai de réalisation de chacune des actions.

```
module CONVERSION_A3 :
    output S_A3_Dir_Co, S_A3_Der_Co;
    sensor C_A3_Dir_Co(boolean), C_A3_Der_Co(boolean);

every tick do
%Conversion des capteurs purs
    if ?C_A3_Dir_Co then emit S_A3_Dir_Co endif;
    if ?C_A3_Der_Co then emit S_A3_Der_Co endif
end
.
```

FIGURE IV.9 : MODULE DE CONVERSION DES CAPTEURS POUR A3.

Le signal TOP est l'unité temps considéré dans cet exemple. Les actions A36 et A37 doivent donc, être réalisées dans un délai de N TOP.

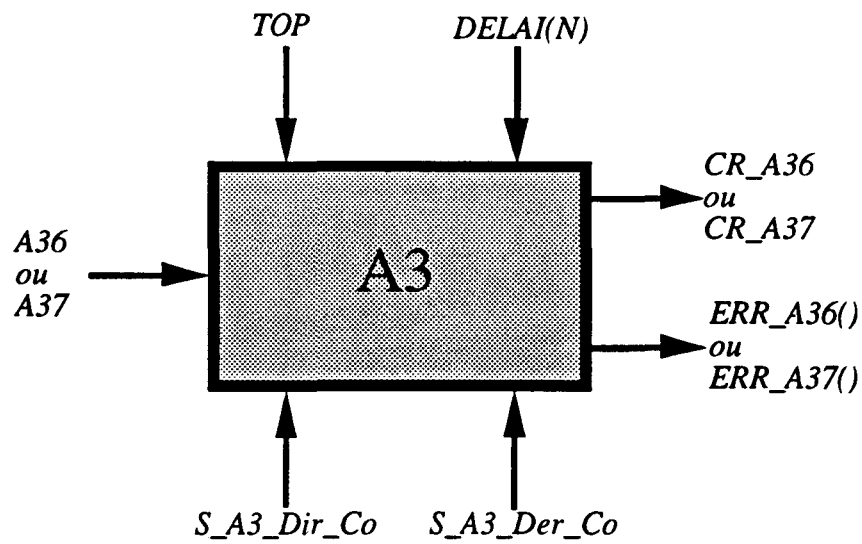


FIGURE IV.10 : MODÈLE RÉACTIF DE L'EXEMPLE.

En fonction de l'état du procédé, et de l'action qui a activé le contrôle, un compte-rendu, ou une erreur sera généré. L'erreur qui est générée peut être de deux types. Le signal d'erreur sera donc valué ("ERR_A36()").

Considérons le cas de l'action A36. Trois possibilités de terminaison du contrôle sont considérées :

- S_A3_Dir_Co est présent, S_A3_Der_Co est absent, N TOP non écoulés, alors le compte-rendu CR_A36 est émis,
- S_A3_Der_Co est présent, S_A3_Dir_Co est absent, N TOP non écoulés, alors une erreur de type 1 ERR_A36(1) est émise,
- S_A3_Der_Co est absent, S_A3_Dir_Co est absent, N TOP écoulés, alors une erreur de type 2 ERR_A36(1) est émise,

Après cette description du comportement du sous module de contrôle de l'aiguillage, nous présentons l'algorithme correspondant.

IV.3. ALGORITHME DU SMC EN LANGAGE ESTEREL.

Dans notre algorithme (FIGURE IV.11), nous retrouvons deux parties : déclarative, impérative. En effet, nous sommes amenés à déclarer tous les signaux d'entrées (input), de sorties (output).

L'algorithme se présente sous la forme d'une boucle infinie. Un pas de la boucle est constitué de deux étapes : attente de l'action à contrôler, et déclenchement du contrôle.

Durant le contrôle, le signal S_A3_Dir_Co permet de valider l'action A36 (CR_A36). A l'inverse, S_A3_Der_Co déclenche une erreur de réalisation de A36 de type 1 (ERR_A36(1)). Si aucun de ces signaux n'est présent et si le délai de réalisation de l'action est écoulé, alors une erreur de type 2 est déclenchée. La description est conforme à la spécification.

IV.4. MODÈLE AUTOMATE DE L'EXEMPLE.

La compilation du module de comportement de l'aiguillage en ESTEREL génère un *automate* qui décrit le fonctionnement (FIGURE IV.12). Ainsi, on retrouve les trois états principaux suivants : attente de l'action déclenchant le contrôle (e2), réalisation de l'action A36 (e3), et réalisation de l'action A37 (e4).

```
module A3 :

input  A36, A37, S_A3_Dir_Co, S_A3_Der_Co,TOP, DELAI(integer);

output CR_A36, CR_A37, ERR_A36(integer), ERR_A37(integer);

await DELAI
loop
    await
        case A36 do
            await tick
            do
              do
                  await ?DELAI TOP
                  emit ERR_A36(2)
              watching S_A3_Der_Co
              timeout
                  emit ERR_A36(1)
              end
            watching  S_A3_Dir_Co
            timeout
                  emit CR_A36
            end

        case A37 do
            ....
    end
end
.
```

FIGURE IV.11 : ALGORITHME DU SMC.

L'évolution se fait par réception d'événements (actions ou changement d'état des capteurs). La réception d'événements génère à son tour des événements par émission (compte rendu ou erreur) en fonction de l'état de l'automate.

L'automate est rendu déterministe en affectant une priorité aux arcs. Ainsi, pour chaque état, l'arc ayant le poids le plus faible est le plus prioritaire.

La simulation de ce modèle est conforme à la spécification qui en a été faite. Nous utilisons l'écriture CSP [HOARE, 78] pour les étiquettes des arcs.

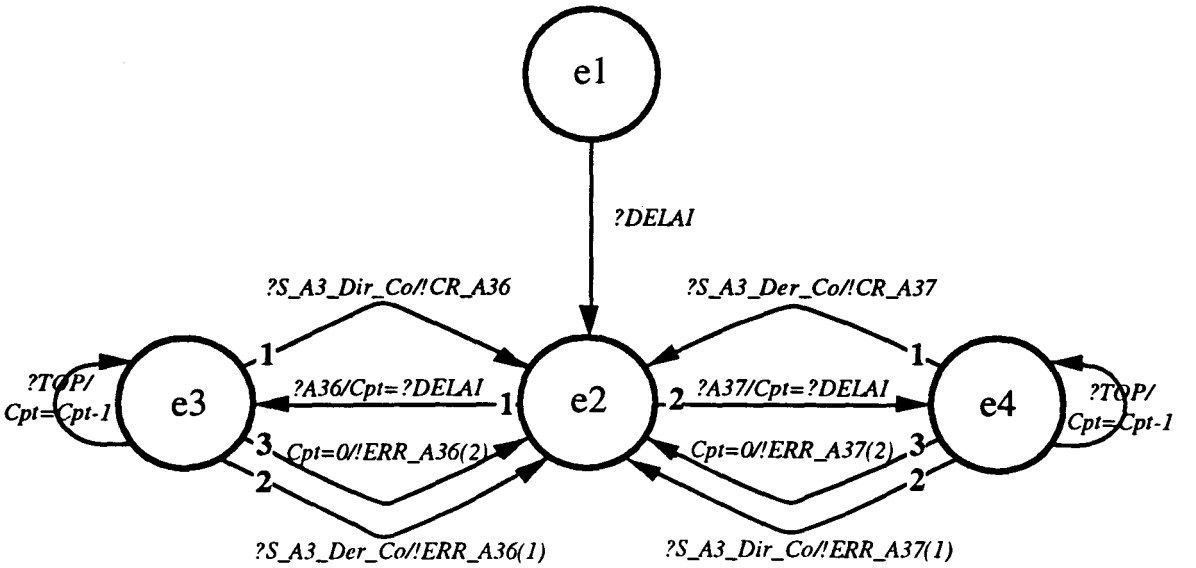


FIGURE IV.12 : AUTOMATE À ÉTATS FINIS DE L'EXEMPLE.

CONCLUSION.

Cette approche permet d'intégrer les aspects de sûreté de fonctionnement dans un système de production, jusqu'à lors, très peu traités dans les SED. L'idée de base est de *surveiller* la réalisation des actions en *temps réel.* D'une part pour mettre en œuvre l'aspect surveillance, il est nécessaire d'introduire de la *redondance explicite* qui permet de détecter les erreurs conduisant à une défaillance (évitement de la propagation des erreurs). D'autre part, pour ce qui est de l'aspect temps réel, la *programmation synchrone* est la plus appropriée. Cette démarche permet donc d'assurer un contrôle réactif, notamment la détection instantanée des erreurs de fonctionnement.

Néanmoins, un apport important en capteurs reste nécessaire. Cette approche étant basée sur l'information acquise auprès des capteurs, il est nécessaire de supposer qu'ils sont sûrs pour valider la démarche. L'un de nos soucis sera de pallier ce problème en utilisant d'autres sources d'informations.

La décomposition du module de contrôle en SMC associés aux composants, confère au système un certain nombre de propriétés intéressantes :

- modularité,
- indépendance entre SMC,
- déterminisme de la réaction par rapport aux événements d'entrée,

- lisibilité et facilité de validation,
- intégration aisée au système de commande.

La démarche de conception du contrôle de commande exposée, est générique. Elle permet d'avoir une conception modulaire et itérative. Elle permet donc, de prendre en charge tout le cycle de conception du système de contrôle. La méthodologie proposée permet en effet de :

- avoir une spécification formelle avec validation,
- puiser dans une bibliothèque d'aide à la formalisation des spécifications en ESTEREL,
- enrichir les modules synchrones,
- générer des automates simulables validant ainsi les spécifications,
- générer des codes implémentables répartis.

Notons que cette approche suit les étapes principales de conception du domaine du génie logiciel [BOOCH, 88].

PARTIE III.

LES FILTRES DE COMMANDE.

CHAPITRE V.

UN OUTIL POUR LES FILTRES DE COMMANDE : LES OBJETS COMMANDABLES.

INTRODUCTION.

Dans ce chapitre, nous proposons de présenter un modèle : les Objets Commandables (OC). Les Objets Commandables sont considérés ici comme un modèle exprimant le comportement d'un composant physique.

Nous allons nous intéresser aux différentes notions mathématiques régissant ce modèle, ainsi qu'à ses propriétés. Nous présenterons également la représentation graphique des OC. Le but est de construire un modèle graphique formel. Pour répondre à ces besoins, nous utiliserons la théorie des graphes.

Après la formalisation mathématique du modèle, nous aborderons des problèmes d'agrégation et de hiérarchisation. Les aspects de communication sont également traités pour interfacer le modèle avec l'environnement existant ou extérieur.

Ce modèle peut être utilisé en industrie manufacturière, pour la validation de la commande en phase de conception ou pour la prévision du comportement et le filtrage des actions en phase d'exploitation.

Nous terminerons ce chapitre, en présentant un exemple d'application de cet outil. Il s'agit de la modélisation du comportement d'une télécommande de téléviseur. Cet exemple a été choisi pour des fins pédagogiques, et permet de montrer toutes les notions liées aux OC.

I. PRÉSENTATION DU MODÈLE.

I.1. CONTEXTE.

Dans le contexte de la modélisation des systèmes de *contrôle/commande*, les *Réseaux de Petri* (RdP), modèles graphiques, par essence, sont souvent utilisés [BRAMS, 83A]. Ils constituent le modèle le plus approprié et le plus complet pour la modélisation de ces systèmes. Ces modèles permettent en effet, de décrire le comportement du système composé de sous-systèmes, contraints à fonctionner en parallèle, de manière dynamique et formelle. Ces modèles ont largement été étudiés, et particulièrement au LAIL [BOUREY, 88; KAPUSTA, 88]. Les obstacles majeurs liés à l'exploitation de ces modèles, sont dus [BRAMS, 83B] :

- à leur *combinatoire explosive*, qui rend difficile leur validation et la vérification de leurs propriétés,
- et à l'*indéterminisme* de fonctionnement engendré par le parallélisme inhérent au système modélisé qui amène des situations critiques : blocage du système, conflits structurels ...

Un autre modèle les STATECHARTS, extension du concept à machine à états, permettent de réduire l'explosion combinatoire des états [HAREL, 87]. Les extensions portent sur la hiérarchisation, le parallélisme et la synchronisation [SAHRAOUI, 91].

Pour toutes ces raisons, nos travaux se sont orientés vers la mise en place d'un outil qui permet de pallier ce problème d'indéterminisme : les *Objets Commandables*. Cet outil doit apporter une solution à l'indéterminisme des systèmes de commande. Ainsi, nous aurons un outil complémentaire des RdP pour la conception des systèmes de commande. Nous proposons dans ce chapitre de présenter la notion d'*Objets Commandables*, Objets qui se veulent *formels* par leurs définitions et *propriétés mathématiques* (avantage très important par rapport aux STATECHARTS). Dans un premier temps, nous nous intéresserons aux termes nécessaires à la définition des OC. Dans un second volet, nous développerons les propriétés des OC, ainsi que quelques notions d'algèbre linéaire adaptées à ceux-ci. Ensuite,

nous mettrons en œuvre des aspects de communications, pour faciliter leur intégration aux autres modèles. Nous terminerons ce chapitre par une mise en œuvre des OC sur un exemple concret.

I.2. NOTIONS GÉNÉRALES.

I.2.1. Description informelle d'un Objet.

Dans un premier temps, nous proposons de fournir une définition informelle d'un OC. Un *Objet* est un constituant physique, d'un atelier flexible, par exemple. Tout Objet réalise un certain nombre de tâches spécifiques (fonctionnalités) en fonction des ordres qu'il reçoit (*actions* ou *commandes*) du système qui le commande. Une action est une commande *élémentaire*, par opposition à une commande non élémentaire qui est une séquence d'actions. Ainsi, une action est non décomposable en actions plus élémentaires.

Nous nous intéressons à un Objet du point de vue des actions, sachant que toute commande non élémentaire est décomposable en actions.

Pour illustrer la notion d'Objet, considérons le cas d'un interrupteur électrique à impulsion. L'interrupteur peut mettre le circuit dans deux positions, une position "ouvert" et une position "fermé". On distingue alors des états qui caractérisent les *états physiques stables* de l'Objet. Ces états sont de type *discret*. La réalisation d'une action modifie l'état de l'Objet, traduite par le passage d'un état stable vers un autre état stable. Dans notre exemple, la seule action réalisable est "activer le bouton poussoir". Pendant la réalisation d'une action, nous considérons que l'Objet est en *mode transitoire*.

Dans cette description, nous voyons apparaître trois notions fondamentales :

- la notion d'*état*, qui représente l'état physique stable de l'OC à un instant donné,
- la notion d'*action*, qui permet de passer d'un état à un autre, de façon instantanée,
- la notion de *mode transitoire*, qui se matérialise par la réalisation d'une action par l'Objet.

Après avoir donné une définition d'un Objet, nous allons tenter d'exprimer la *commandabilité*.

L'Objet sera dit ***commandable***, s'il vérifie deux propriétés :

- à partir d'un état quelconque, nous accédons à tous les autres états en réalisant une suite d'actions. Ce qui définit l'*accessibilité* d'un état de tout autre état,
- l'Objet doit avoir un comportement *déterministe* vis-à-vis de toutes les actions réalisables par l'Objet.

En effet, un interrupteur électrique à impulsion est bien un OC, puisqu'on peut passer d'un état à un autre et qu'il a un comportement déterministe vis-à-vis des actions qui lui sont appliquées.

Après cette description informelle des termes et notions utilisés, donnons à présent une définition rigoureuse de ceux-ci.

I.2.2. Notion d'Objet [ELKHATTABI, 92A].

Un Objet est défini par deux ensembles : l'ensemble des états et l'ensemble des actions. Les états peuvent donc être représentés par un ensemble englobant tous les *états discrets stables* possibles de l'Objet. Cet ensemble sera noté E. L'ensemble des états caractérisant l'Objet est *fini*. De plus, il est *complet*(1).

L'ensemble des actions regroupe toutes les actions élémentaires réalisables par l'Objet. Cet ensemble permet de représenter les fonctionnalités de l'Objet. Il sera noté A. A est lui aussi *fini* et *complet*(2). Le passage d'un état vers un autre se caractérise par l'exécution d'une action. A est donc un ensemble d'*applications* (définies sur E), de E vers E.

Définition 1. Un Objet est un constituant physique défini par un couple O = <E, A> où :
E est un ensemble fini d'états discrets stables pour lequel on note n=card(E),
A est un ensemble fini d'applications de E vers E pour lequel on note m=card(A).

Exemple 1. Un interrupteur électrique à impulsion peut être défini par le couple O_1 = <E, A> avec:
E = {e_1="ouvert", e_2="fermé"},
A = {a="activer le bouton poussoir"},
$a(e_1)=e_2$ et $a(e_2)=e_1$.

(1) E contient tous les états physiques stables possibles par construction.
(2) A contient toutes les actions réalisables par l'objet spécifié.

Afin de faciliter la compréhension de toute notion, une représentation graphique est souvent nécessaire. Nous essayerons donc, d'élaborer une représentation permettant de refléter le comportement de l'Objet modélisé.

I.2.3. Graphe associé à un Objet.

Nous allons nous intéresser à la représentation graphique de notre modèle afin de permettre l'assimilation rapide et aisée du fonctionnement d'un Objet. A cet effet, nous avons emprunté la définition d'un graphe citée par SAKAROVITCH [84].

Définition 2. Un graphe G est défini par :
- deux ensembles X et U dits ensembles de sommets et d'arcs respectivement,
- deux applications I et T : U $\rightarrow$ X qui associent à chaque arc son extrémité initiale et son extrémité terminale respectivement.

Suite à cette définition, un Objet peut être considéré comme un graphe orienté où E est identifié à X et A à l'ensemble dont les éléments étiquettent les arcs. Quant à l'arc, il est défini par son état initial, l'action l'étiquetant et son état terminal. Un arc est donc un élément de E×A×E.

Définition 3. Le graphe G = <E, U, I, T> associé à l'Objet O = <E, A> est défini par :
$U = \{(x,a,y) \in E \times A \times E \ / \ a(x)=y\}$[3],
$a(x)=y \Leftrightarrow I(xay)=x$ et $T(xay)=y$.
On note I_X (resp. T_X) l'ensemble des extrémités terminales (resp. initiales) des arcs ayant x comme extrémité initiale (resp. terminale) et A_X l'ensemble des étiquettes des arcs dont l'extrémité initiale est x.

Représentation. Différentes représentations sont possibles : [KAUFMANN, 68]
- représentation en casier,
- représentation en matrice booléenne,
- représentation en matrice latine,
- représentation par correspondance,
- représentation sagittale.

(3) On notera par abus u=(xay) l'arc défini par le triplet (x,a,y).

Nous avons choisi la représentation sagittale que nous avons étendue. En effet, les états seront représentés par des cercles et les arcs par des liaisons orientées étiquetées. Un arc ainsi défini, mettra donc en jeu un état initial, un état terminal et une action étiquetant celui-ci. La notion de mode transitoire définie précédemment, apparaît de façon implicite sur le graphe associé à l'Objet par le biais des arcs. Nous verrons plus loin, la représentation du mode transitoire

Le graphe associé à un Objet est donc un graphe à états/transitions (automate à états finis, Réseau de Petri à états). Les RdP à états permettent de mettre en évidence l'aspect dynamique. En effet, l'une des extensions possibles du modèle serait l'intégration d'un marquage permettant l'expression de la dynamique.

Etat *Arc* étiqueté par l'action *a*

FIGURE V.1 : REPRÉSENTATION DES ÉTATS ET DES ARCS.

Exemple 2. Reprenons le cas de l'interrupteur, le graphe G_1 associé se présente comme suit :

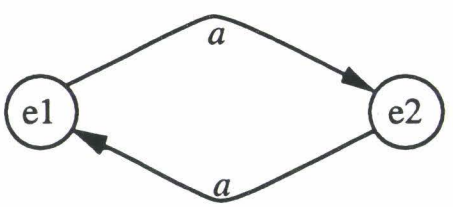


FIGURE V.2 : GRAPHE G_1 ASSOCIÉ À L'OBJET O_1.

II. OBJET COMMANDABLE (OC).

Supposons qu'un Objet O = <E, A> se trouve dans un état e. Si pour chaque action a de A, il n'existe qu'un seul état e' de E tel que la réalisation de l'action a mette O dans l'état e', on dira que le comportement est déterministe complètement spécifié pour l'état e. Maintenant, si cette propriété est vérifiée pour tous les éléments de E alors l'Objet O a un comportement déterministe complètement spécifié.

Définition 4. *Un Objet* $O = <E, A>$ *est* ***déterministe complètement spécifié*** *s'il vérifie la propriété suivante :*

$\forall e \in E \quad \forall a \in A \quad \exists! e' \in E \ / a(e)=e'$

Propriété 1. *Si l'Objet* $O = <E, A>$ *est déterministe complètement spécifié alors :*

$\forall e \in E \quad A_e = A.$

Exemple 3. Soit O_2 l'Objet <E, A> avec E = {x, y, z} et A = {a, b}.
l'Objet O2 est déterministe. En effet, on s'aperçoit très vite que $A_x = A_y = A_z = A$.

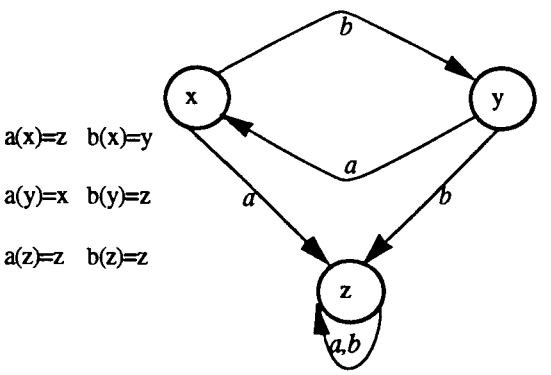


FIGURE V.3 : GRAPHE G_2 ASSOCIÉ À L'OBJET O_2.

O1 est déterministe : $A_{e1} = A_{e2} = A$

Soient e et e' deux états quelconques de E. On dira que e' est accessible depuis e, s'il existe une séquence d'actions de A menant de e à e'. Si cette propriété est vérifiée pour tout couple (e,e') de E×E, on dira que tous les états sont accessibles depuis n'importe quel état.

Définition 5. *Soit un Objet* $O = <E, A>$, *et deux états e et e' de E* ($e \neq e'$).

On dit que e est ***accessible par*** *e' si* $\exists \alpha \in A^+ / \alpha(e')=e$, *où* A^+ *est l'ensemble des séquences formées par la composition des actions.*

On dit que e est ***accessible*** *si e est accessible par tous les autres éléments de E.*

Définition 6. *Un Objet* $O = <E, A>$ *est* ***accessible*** *si tous les états de E sont accessibles.*

La notion de connexité d'un graphe permet de mettre en évidence l'accessibilité des sommets. Elle est utilisée notamment pour vérifier la connexité d'un réseau électrique, d'un

réseau téléphonique, etc [GONDRAN, 79]. L'intérêt de cette caractéristique est de prouver de manière graphique les propriétés mathématiques du modèle. C'est pour cette raison que nous utilisons cette propriété des graphes. Avant tout, donnons en une définition citée dans SAKAROVITCH [84].

Définition 7. *Deux sommets x et y d'un graphe G sont entre eux dans la relation de* ***connexité*** *(resp.* ***connexité forte****) si x=y ou bien s'il existe une chaîne joignant x et y (resp. un chemin de x à y et un chemin de y à x).*

Propriété 2. *Deux états x et y sont* ***fortement connexes*** *si et seulement si ils sont* ***mutuellement accessibles.***

Propriété 3. *Le graphe associé à un Objet O est* ***fortement connexe*** *si et seulement si O est* ***accessible.***

Le graphe G_1 (Interrupteur à impulsion) est fortement connexe, puisque ses deux états sont mutuellement accessibles. Par contre, le graphe G_2 n'est pas fortement connexe, puisque y et x ne sont pas accessibles depuis z. En effet, O_2 n'est pas accessible.

Après avoir défini de façon mathématique tous les termes nécessaires à la définition d'un OC, on peut donc énoncer la définition suivante d'un OC.

Définition 8. *Un Objet est dit* ***commandable*** *si et seulement si il est déterministe complètement spécifié et accessible.*

Reprenons l'exemple de l'interrupteur. Il est déterministe et accessible puisque tous les états sont accessibles. Donc, un interrupteur électrique à impulsion est un OC. Par contre, l'Objet O_2 n'est pas commandable, puisqu'il n'est pas accessible.

III. OBJET COMMANDABLE ELÉMENTAIRE (OCE).

Un OC peut être vu comme une composition d'OC communiquant entre eux. Il peut donc être décomposé en plusieurs OC. Le but de cette décomposition est d'arriver à des OC non décomposables. Ces OC non décomposables, sont appelés Objets Commandables Elémentaires (**OCE**). Cette démarche doit permettre de faciliter la phase de modélisation. Cette phase doit pouvoir être automatique. Dans l'industrie manufacturière, il est quasiment

impossible d'avoir des modèles génériques du fait de la diversité des organes qui composent un atelier flexible.

La recherche des OCE qui forment un Objet ou un OC, se fait selon un critère de connexité directe. Cette décomposition fonctionnelle fait donc intervenir deux notions qui sont à exprimer de façon rigoureuse :

- la notion de connexité directe,
- la notion d'OCE.

Un OC non décomposable est dit *élémentaire*. C'est une entité physique commandable atomique. Ce type d'Objet a plusieurs caractéristiques, notamment :

- il est déterministe complètement spécifié,
- tous ses états sont accessibles entre eux avec un α de *longueur unitaire, unique.*

Pour un OCE, le passage d'un état quelconque à tout autre état se fait donc par l'exécution d'*une et une seule* action. On parle alors d'*accessibilité directe.* Après l'énoncé de cette propriété, définissons de manière mathématique la notion d'accessibilité directe et la notion d'OCE.

Définition 9. *Soit un Objet $O = <E, A>$, un état e de E est* ***accessible directement*** *ou* ***1-accessible*** *si :*
$\forall e' \in E,\ e \neq e',\ \exists a \in A\ /\ a(e')=e.$
Si l'existence de a est unique, on dira que e est ***1-accessible parfait.***

Définition 10. *Un Objet est dit* ***commandable élémentaire,*** *s'il est déterministe complètement spécifié et tous ses états sont 1-accessibles parfaits.*

Soient l'Objet O = <E, A> et e un élément de E. Si O est un OCE et e accessible directement alors l'existence de a (cf définition 9) est unique. Nous pouvons alors énoncer la propriété suivante :

Propriété 4. *Si $O = <E, A>$ est 1-accessible alors :*
$\forall e \in E\ T_e = E$ *ou* $T_e = E-\{e\}.$

Propriété 5. *Si $O = <E, A>$ est 1-accessible alors :*
$\forall e \in E\ I_e = E$ *ou* $I_e = E-\{e\}.$

Un certain nombre de notions de la théorie des graphes, en l'occurrence les degrés des sommets sont nécessaires pour poursuivre notre étude. Ainsi, nous proposons d'en énoncer les définitions, citées par GONDRAN [79].

Définition 11. *Le **demi-degré extérieur** (resp. **intérieur**) du sommet x, noté $d^+(x)$ (resp. $d^-(x)$) est le nombre d'arcs ayant x comme extrémité initiale (resp. terminale).*
*Le **degré** du sommet x, noté d(x) est le nombre d'arcs ayant x comme extrémité, et on a :*

$d(x) = d^+(x) + d^-(x)$.

La définition 9 et la propriété 1, nous permettent d'affirmer que pour tout état x d'un OCE, on a :

$d^+(x)=m$ (où m est le cardinal de A).

Or $d^+(x)$, peut prendre deux valeurs. En effet, s'il n'existe pas d'action ayant pour élément neutre x, alors, $d^+(x) = n-1$ (où n est le cardinal de E).
Cette affirmation découle de l'accessibilité directe de tous les états (propriété 5). Par contre, si x est un élément neutre d'une action, alors $d^+(x) = n$. On peut donc affirmer que pour tout état x d'un OCE, on a :

$n-1 \leq d^+(x) \leq n$.

Par conséquent, nous avons alors $n-1 \leq m \leq n$.

Propriété 6. *Si $O = <E, A>$ est un OCE alors $card(E)-1 \leq card(A) \leq card(E)$.*

Propriété 7. *Soit un OCE O. Le graphe associé à O est **plein** si $card(A)=card(E)$ et réciproquement.*

Reprenons l'exemple de l'interrupteur électrique à impulsion. O1 est un OCE puisque tous ses états sont voisins. Cependant, son graphe n'est pas plein car $card(A) \neq card(E)$. Par contre, un interrupteur électrique "classique" est Commandable Elémentaire et son graphe est plein. En effet, card(A)=card(E).

Si le graphe associé à un OCE est plein, l'ensemble des actions réalisables par l'Objet forment une *relation d'équivalence*. Dans le cas contraire, la relation formée par les actions est symétrique, transitive et non-réflexive.
Les OCE sont les Objets que nous aurons le plus souvent à modéliser car en général, les Objets sont de type "tout ou rien".

IV. DIFFÉRENTS ÉTATS D'UN OBJET ET MODE TRANSITOIRE.

IV.1. AGRÉGATION DES ÉTATS DISCRETS.

Tous les exemples qui ont été traités jusqu'à lors sont à états discrets. Un Objet peut avoir plusieurs états discrets représentant chacun un même état physique mais avec des valeurs discrètes différentes. Dans ce cas, une agrégation de tous ces états, en un seul état paramétré par une variable d'état, s'impose pour plusieurs raisons :

- lisibilité et facilité de représentation,
- réduction du nombre d'états,
- facilité de validation.

Néanmoins, cette agrégation ne peut se faire que sous certaines conditions :

- les états impliqués dans l'agrégation représentent la même information,
- les actions appliquées à chaque état impliqué sont les mêmes.

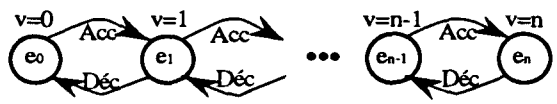


FIGURE V.4 : MODÈLE NON AGRÉGÉ DU MOTEUR.

FIGURE V.5 : MODÈLE AGRÉGÉ DU MOTEUR(4).

En effet, prenons le cas d'un moteur. Les actions réalisables par cet Objet sont "Acc" (Accélération) et "Déc" (Décélération). L'état de marche de celui-ci est représenté par autant d'états que la vitesse de rotation a de valeurs (0 à n). Pour décrire le fonctionnement, il est donc suffisant d'utiliser un seul état de marche paramétré par la vitesse de rotation (les graphes de la FIGURE V.4 et la FIGURE V.5 sont identiques).

(4) Le "•" représente le ET logique.

Après agrégation, nous nous retrouvons avec un seul état paramétré par la vitesse. Pour notre exemple, nous aurions pu isoler le cas particulier où la vitesse est nulle. En effet, ce cas permet de mettre en évidence l'état arrêt (moteur sous tension).

L'agrégation rend le modèle plus concis et plus lisible. Nous essayerons donc d'agréger les modèles dès que les conditions nécessaires pour le faire seront réunies. Cette étape vient après une modélisation directe depuis la spécification de l'Objet.

IV.2. MACRO-ÉTAT.

Un niveau d'abstraction est nécessaire afin d'avoir une analyse descendante du système : la *hiérarchisation*. Ce niveau est important si on veut avoir une vision plus détaillée du comportement. Cette abstraction apparaît au niveau des états : notion de *macro-état* qu'on notera *M-état*. Un M-état est un état commun à un ensemble d'états. Pour un Objet complexe, un M-état peut regrouper plusieurs OC évoluant de façon parallèle.

Afin d'illustrer cette notion, reprenons le cas du moteur. Il se caractérise par un état discret (Arrêt) et un M-état (Marche). L'état de marche englobe ainsi, deux "états paramétrés" (En accélération et En décélération) paramétrés par la vitesse du moteur.

FIGURE V.6 : REPRÉSENTATION D'UN MACRO-ÉTAT.

Un M-état permet aussi de représenter la préemption, notion fondamentale pour la surveillance des systèmes. Cette propriété de préemption, différencie la notion de M-état de la macro-place ou de la substitution des transitions en RdP hiérarchisés [HUBER, 90]. En effet, cette notion apporte un avantage certain : priorité des événements par hiérarchie. Ainsi, un M-état peut "préempter" tous les OC du niveau inférieur. Afin d'illustrer cette notion, prenons le cas d'un atelier flexible. Un signal d'arrêt d'urgence, mettra tous les composants de l'atelier hors tension. En résumé, on distingue trois entités permettant de représenter les différents états : un état discret, un état agrégé paramétré avec variable d'état discrète, et un M-état regroupant plusieurs OC.

IV.3. MODE TRANSITOIRE.

La notion de transitoire est directement liée aux actions, puisqu'elle exprime la réalisation de celles-ci. Pour mettre en relief cette notion, nous allons nous intéresser aux actions. Une action est une commande élémentaire qui ne peut être décomposée en d'autres actions. Une action permet de passer d'un état vers un autre. Sur le plan de la modélisation, ce passage d'un état vers un autre se fait de manière instantanée. Cependant, du point de vue de l'implantation, cette caractéristique n'est plus valable. En effet, la réalisation d'une action induit une durée de changement d'état. Ce délai est dit temps de réalisation de l'action. Il doit être borné afin d'assurer à l'Objet une vivacité : non blocage de l'Objet. Durant ce temps de réalisation, on dira que l'Objet est en mode transitoire.

Dans de nombreuses situations, la fin d'une action est accompagnée d'un compte-rendu. Une action peut alors être décomposée en trois étapes plus au moins abstraites :

- début d'action,
- mode transitoire : action en cours,
- fin d'action.

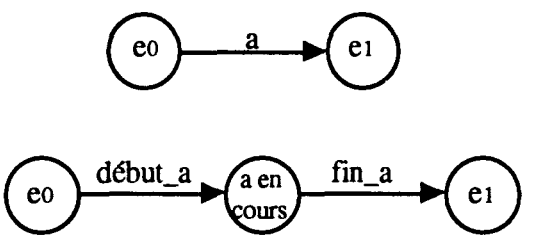


FIGURE V.7 : REPRÉSENTATION DU MODE TRANSITOIRE.

Le changement d'état peut être alors caractérisé par un état intermédiaire dit "action en cours". Ainsi, un arc sera décomposé en un arc "début d'action" débouchant sur l'état "action en cours" et un autre arc "fin d'action" accédant à l'état final. Ce changement d'état est représenté graphiquement par la FIGURE V.7 (les deux graphes sont équivalents).

V. PRIMITIVES DE COMMUNICATION.

Tout Objet est amené à être intégré dans un environnement. Il a besoin de communiquer avec l'environnement extérieur ou d'autres Objets. Il est donc nécessaire de mettre en place un mécanisme de réception/émission de signaux. Dans notre formalisme,

nous assimilons ce mécanisme à des actions de réception/émission. Une émission ou une réception de signal est vue comme une action. Néanmoins, il est à noter que l'émission d'un signal ne modifie jamais l'état de l'Objet en question. Par contre, une réception modifie l'état du système. Le symbolisme utilisé pour représenter ce mécanisme de communication peut être le suivant [HOARE, 78] :

?Sig : Réception (ou attente) de Sig

!Sig : Emission de Sig.

La réception (resp. émission) peut être limitée à un signal, mais peut être étendue facilement à une réception (resp. émission) simultanée de plusieurs signaux. Nous avons ainsi la possibilité d'attente multiple et d'émission multiple de signaux. Cette notion de simultanéité sera représentée comme suit :

{? ou !} (Sig1•Sig2•...•Sign) : Réception ou émission de n signaux(5).

A toute réception de signal peut être associée l'émission d'un autre signal ou l'exécution d'une action spécifique. Ceci s'écrit alors sous la forme suivante :

?Sig1/!Sig2 ou ?Sig/Action.

Les aspects de communication qui ont été introduits jusque là, seront mis en œuvre à titre d'illustration dans l'exemple présenté ci-après.

VI. EXEMPLE D'UNE TÉLÉCOMMANDE DE TÉLÉVISEUR.

Afin d'illustrer les notions présentées, nous allons modéliser le fonctionnement des Objets qui composent un téléviseur. Nous allons nous intéresser plus précisément à la télécommande d'un téléviseur. Le choix d'un tel exemple, non manufacturier, a été motivé par le souci de faciliter la compréhension.

La télécommande représente le système de commande et le téléviseur le système commandé. On distingue sur cette FIGURE plusieurs groupes d'actions. Chacun de ces groupes s'adresse à un Objet Commandable spécifique. On peut les classer comme suit :

- boutons, touches programmes et VIDEO : sélection des canaux,
- touches volume et bouton annulation son : réglage du niveau son,
- boutons couleur : réglage du niveau couleur,
- boutons image : réglage du niveau luminosité,

(5) Le "•" représente le ET logique.

- attente, TV TURN ON et le bouton ON/OFF se trouvant sur le téléviseur : arrêt/mise en marche/veille du téléviseur.

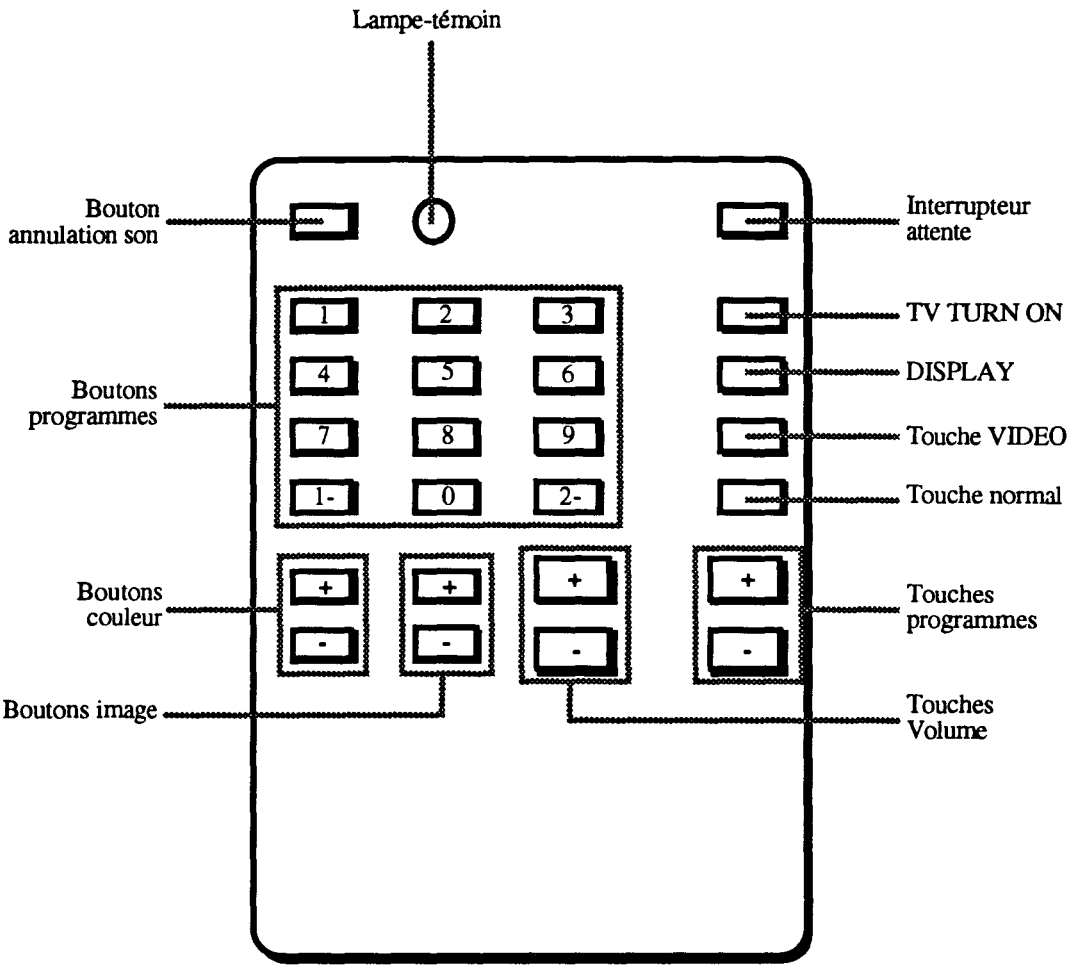


FIGURE V.8 : TÉLÉCOMMANDE D'UN TÉLÉVISEUR.

Il est à noter que les différents modes (sélection des canaux et les différents réglages) ne sont possibles que si le téléviseur est en état de marche : l'état "marche" est un M-état. On distingue ainsi deux niveaux hiérarchisés :

- un niveau de mise en marche/arrêt/veille du téléviseur,
- un niveau inférieur regroupant toutes les autres fonctionnalités du téléviseur.

Le téléviseur peut donc être représenté au premier niveau par un modèle décrivant son fonctionnement, par deux états discrets ("arrêt" et "veille") et un M-état ("marche"). Ce modèle est présenté par la FIGURE V.9. A ce niveau, le modèle n'est réceptif qu'à un nombre limité d'actions ("ON", "OFF", "attente" et "TURN ON").

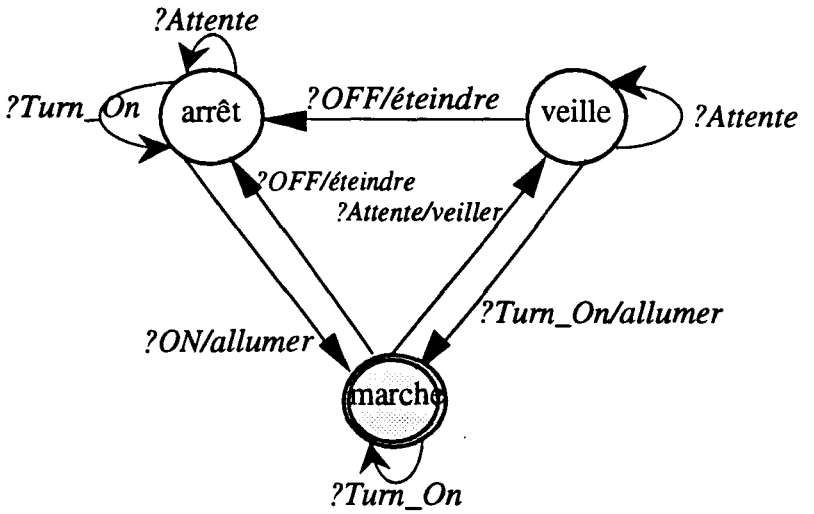


FIGURE V.9 : MODÈLE COMPORTEMENTAL DU TÉLÉVISEUR.

Une transition est représentée par la réception d'un signal (appui sur une touche) et l'action "effective" associée qui sera réalisée par l'OC. Une action "effective" est exprimée sous la forme d'un verbe ("éteindre", "allumer", "veiller"). Cette décomposition a été introduite pour mettre en évidence les arcs auxquels aucune action "effective" n'est associée : le signal reçu est tout simplement ignoré.

Le modèle associé au premier niveau (FIGURE V.9) représente bien un OC, puisqu'il apparaît clairement qu'il est accessible (tous les états sont voisins) et déterministe complètement spécifié. Cette FIGURE permet d'illustrer la notion de macro-état et de mettre en évidence un des aspects de communication.

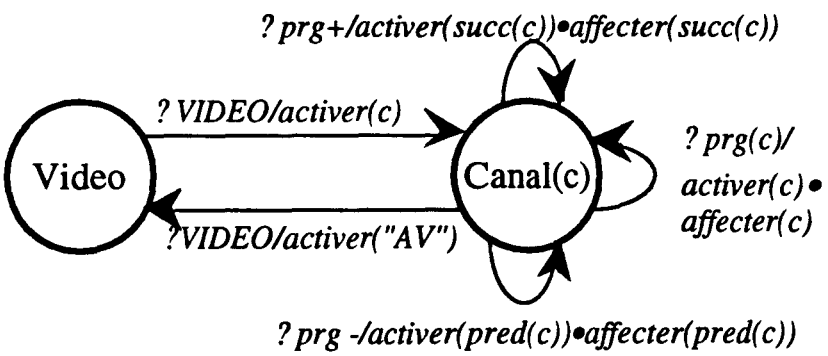


FIGURE V.10 : MODÈLE COMPORTEMENTAL DE LA SÉLECTION DES CANAUX.

Au second niveau de hiérarchisation, nous allons modéliser les autres Objets. Nous allons nous intéresser en premier lieu à la sélection des canaux, afin de montrer l'intérêt d'un état agrégé. Le modèle fourni tient compte de l'agrégation de tous les canaux (sauf le canal VIDEO) dans un même état discret agrégé "canal(c)" où c est une variable prenant ses valeurs dans un ensemble discret C = {0,1,2,...,c_max}, avec c_max+1 le nombre de canaux possibles. L'ensemble C est muni d'une relation d'ordre où le successeur de c_max est 0 et le prédécesseur de 0 est c_max. La sélection des canaux peut être représentée par un état discret ("VIDEO") et un état agrégé ("canal(c)"). Le choix d'un canal par les boutons programmes ou les touches programmes entraîne l'activation du canal désiré ("activer(c)") et son affectation à la variable c ("affecter(c)"). Le modèle est représenté FIGURE V.10.

CONCLUSION ET PERSPECTIVES.

L'approche de modélisation présentée dans ce chapitre, basée sur les OC permet de résoudre un bon nombre de problèmes rencontrés lors de la conception des systèmes de contrôle/commande. Dans un premier temps, elle permet de fournir un modèle comportemental très concis des ressources du système commandé, ainsi que, de modéliser le comportement et de le prédire de manière certaine (déterminisme). Elle offre, dans un deuxième temps, la possibilité de détection des états puits (accessibilité), évitant ainsi, dès la phase de conception, des situations de blocage du système commandé. Ce type de modèle peut donc être utilisé à deux niveaux : durant la conception, pour valider le système de commande, et en phase d'exploitation pour prédire le comportement et filtrer les actions incompatibles avec le système commandé. Le deuxième aspect permet d'assurer une cohérence au système de contrôle/commande.

Une autre notion a été introduite : la hiérarchie. Cette notion permet de faciliter et d'affiner la modélisation. Elle permet également de représenter de manière claire la préemption. La hiérarchie, présente dans les RdP, n'est pas sensible à la préemption. Ainsi, il est possible de mettre hors service un ou plusieurs Objets en cas de dysfonctionnement.

Nos recherches sur les OC portent sur deux points : le premier traite l'intégration de la dynamique et l'étude approfondie des propriétés mathématiques. Le deuxième point, plus avancé, concerne la mise en place d'une méthodologie de conception assistée des modèles, appliquée aux ateliers flexibles dans l'industrie manufacturière. Un outil graphique mettant en œuvre cette méthodologie est présentée dans le chapitre suivant.

CHAPITRE VI.

LA CONCEPTION DES FILTRES DE COMMANDE.

INTRODUCTION.

Dans ce chapitre, nous allons nous intéresser à la conception des **F**iltres de **C**ommande (**FC**). En premier lieu, nous exposerons l'idée de base du filtrage, l'approche de conception et l'organisation d'un **M**odule de **F**iltres de **C**ommande (**MFC**). L'intégration des contraintes de sécurité et de fonctionnement entre composants, introduit la notion de **C**omposant **F**onctionnel **L**ogique (CFL).

Dans un second temps, nous aborderons la démarche de conception. Notamment, les différentes étapes du cycle de conception, ainsi que les modèles obtenus. Nous illustrerons notre approche à l'aide d'un exemple. La faisabilité de la phase de modélisation est réalisée par un programme PROLOG.

Dans un troisième temps, nous présenterons une application informatique destinée à la **C**onception des **O**bjets **C**ommandables **E**lémentaires (**COCE**).

I. PRÉSENTATION DU MODULE DE FILTRES DE COMMANDE.

I.1. IDÉE DE BASE.

La fonction du **M**odule de **F**iltres de **C**ommande (**MFC**) est d'assurer un séquencement strict des commandes qui respecte l'état du procédé et des contraintes fonctionnelles, liant les différents éléments qui le compose. Garantir ce séquencement strict permet de conférer au système de commande disponibilité, et sécurité. Afin d'aboutir à ce résultat, les commandes doivent être systématiquement filtrées.

Ne réaliser une commande, que sous certaines conditions, évite de commander des composants dont l'état est incompatible avec la commande. La commande filtrée garantit donc la validité de la commande. La non réalisation d'une commande valide, est détectée de manière instantanée grâce au MCC. Ces deux caractéristiques permettent d'éviter la propagation d'erreurs et d'assurer par là même la disponibilité des équipements.

Le séquencement strict des commandes intégrant les contraintes fonctionnelles et de sécurité permet d'éviter les risques de collisions et d'accidents.

A cet effet, nous avons besoin d'une modélisation assez fine du procédé. Cette modélisation a une double vocation : éviter des confusions de spécifications et obtenir un modèle indépendant de l'exploitation qui en sera faite. Ce modèle peut donc être réutilisé dans d'autres contextes : recouvrement par prévision du comportement, pilotage par le séquencement des actions, simulation du système de commande en conception, etc.

Le principal objectif de ce modèle n'intégrant que les contraintes fonctionnelles et de sécurité, est de filtrer les commandes à réaliser par le procédé. Le MFC doit représenter le comportement strict des composants modélisés.

I.2. EXEMPLE D'APPLICATION.

Afin d'illustrer notre approche, nous travaillerons par la suite sur des composants d'un tour numérique. Ce type de tour est muni de sa propre commande numérique.

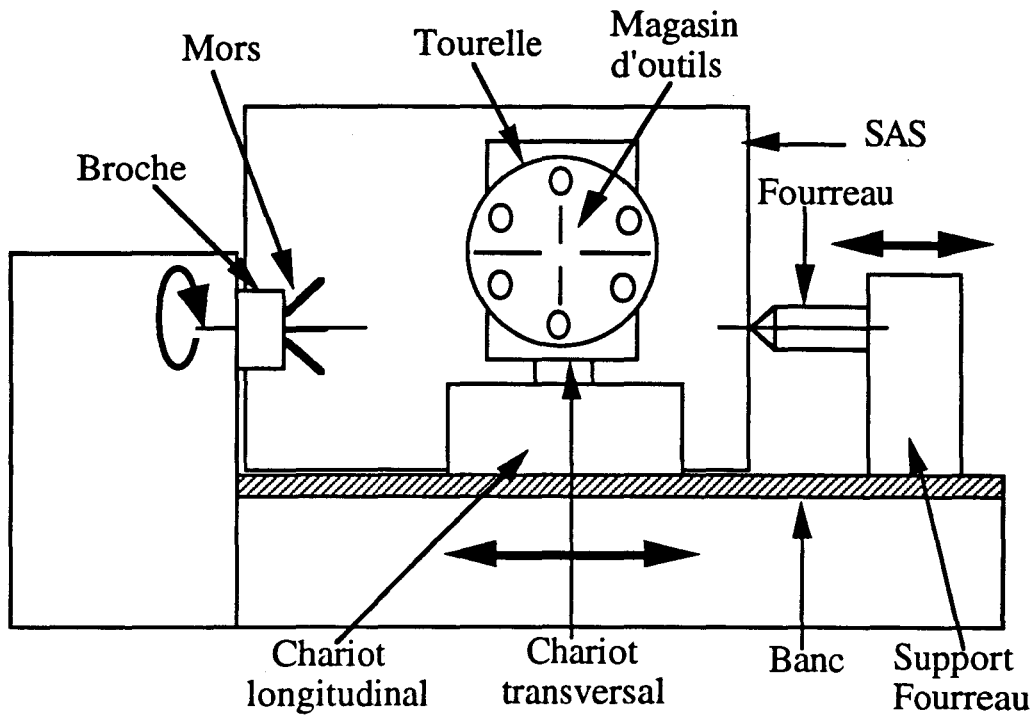


FIGURE VI.1 : TOUR A COMMANDE NUMÉRIQUE.

Généralement, nous retrouvons toujours les mêmes familles d'actionneurs dans un atelier flexible : vérins, moteurs électriques, moteurs pas à pas, aiguillages, butées, ... C'est pourquoi nous ne nous intéresserons pas directement aux actionneurs mais aux composants qu'ils activent. Prenons le cas de la broche. Quand le moteur est en rotation, nous dirons alors que la broche est en rotation.

I.3. APPROCHE DE CONCEPTION.

L'idée majeure est d'aboutir à un modèle indépendant de son utilisation ultérieure. Il est donc nécessaire d'identifier l'ensemble des opérations réalisables par un procédé, ainsi que leurs conditions de réalisation. A cet effet, la génération des composants qui constituent le système physique, s'impose : décomposition structurelle du procédé. Une spécification des opérations de chaque composant, complète cette décomposition. Une définition des modes d'évolutions de chaque opération, à partir des spécifications techniques du composant, permet de définir les comportements élémentaires.

La modélisation du comportement de chaque composant se fait par les Objets Commandables (OC). A un niveau de description assez fin, il s'agit d'OC élémentaires.

Cette étape de modélisation doit être validée : la spécification de chaque composant (indépendamment des autres) doit vérifier les propriétés de commandabilité et d'atomicité.

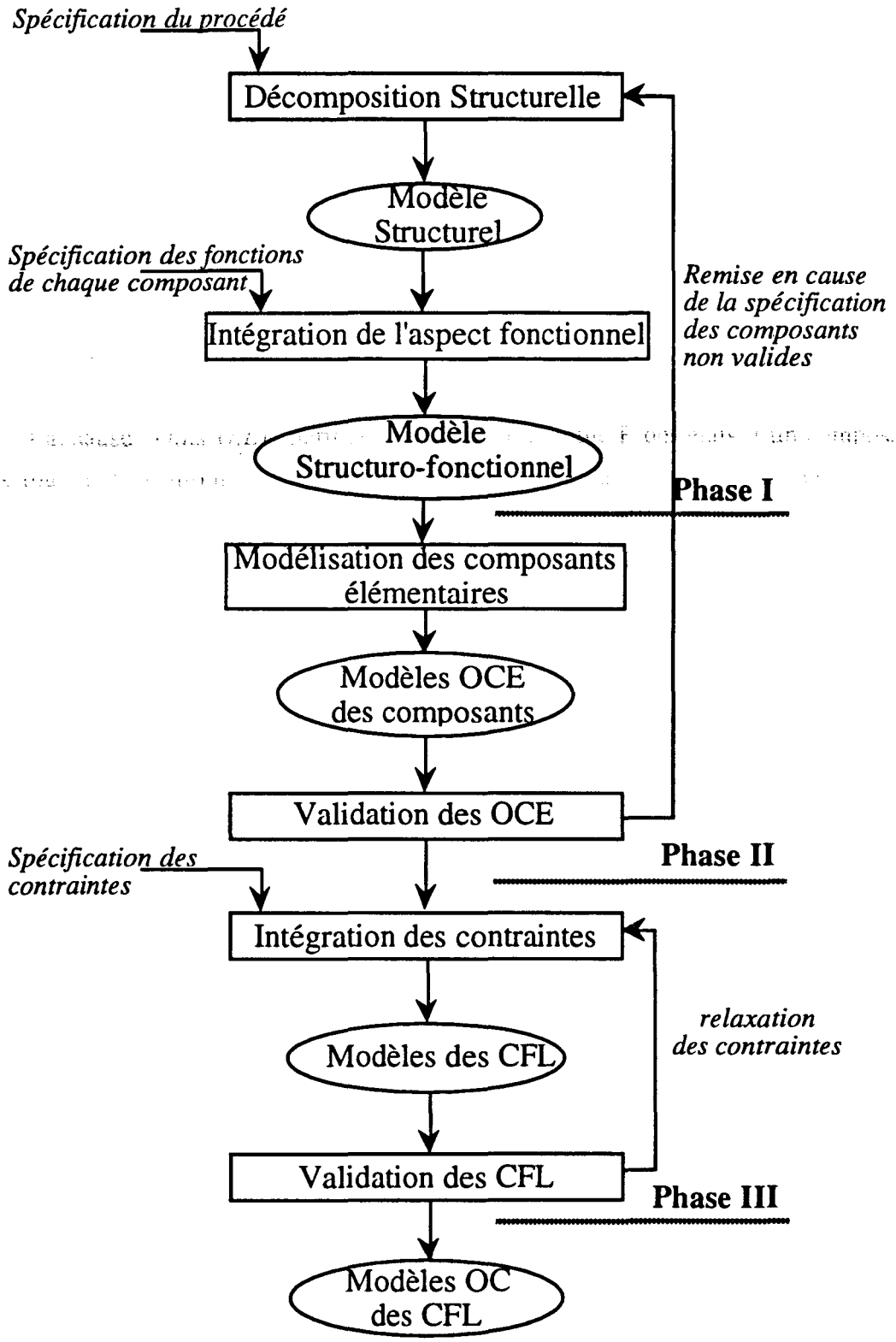


FIGURE VI.2 : DÉMARCHE DE MODÉLISATION.

A ce stade de la démarche, une spécification des contraintes fonctionnelles et de sécurité est à faire. Les contraintes sont associées aux composants concernés et permettent de regrouper les composants contraints. Ce regroupement donne naissance à des Composants Fonctionnels Logiques (CFL). Une étude des propriétés de ces modèles permet de valider ce regroupement ou de remettre en cause les contraintes spécifiées par relâchement.

I.4. NOTION DE COMPOSANT ET DE COMPOSANT FONCTIONNEL LOGIQUE.

Nous nous intéressons particulièrement aux composants élémentaires. Un composant élémentaire est un objet physique réalisant un certain nombre de fonctionnalités élémentaires. Il s'agit des actions. Puisque nous nous intéressons aux composants élémentaires, indépendamment de l'utilisation ultérieure, la flexibilité matérielle n'est nullement réduite.

Un Composant Fonctionnel Logique (CFL), est un objet virtuel. Il est composé de plusieurs composants élémentaires contraints. Le terme Fonctionnel désigne un regroupement des composants par contraintes fonctionnelles. Quant au terme "Logique", il identifie l'aspect virtuel du composant, par opposition à l'aspect physique.

Le regroupement de composants en CFL, implique le respect des contraintes par les composants. La synchronisation des composants devient implicite. Un autre avantage de ce regroupement, se situe au niveau de l'implantation. En effet, le modèle d'un CFL, doit être implanté sur un seul organe et ne peut être réparti.

Le respect de ces contraintes, a pour conséquence de garantir une sécurité matérielle et humaine. De plus, le séquencement des actions induit par ce regroupement, évite la propagation des fautes d'un composant sur les autres. Cette conséquence assure une fiabilité de la commande, une disponibilité du matériel et une facilité de recouvrement ou de maintenance.

I.5. ORGANISATION DU MFC.

La démarche de modélisation d'un système physique, conduit à des filtres comportementaux associés à des CFL. Les CFL sont indépendants sur le plan comportemental. En effet, réaliser une action ou une commande par un CFL, n'altère en rien

le comportement d'un autre CFL. Le modèle d'un CFL doit modéliser tous les comportements autorisés de ces composants. Les comportements interdits sont modélisés de manière implicite. En effet, interdire une évolution dans un état donné, revient à la disparition de l'action amenant cette évolution.

Prenons le cas de la broche et des mors du tour. Il est impossible de considérer le comportement de ces deux composants, séparément : ces deux composants sont contraints. Les mors ne peuvent être desserrés quand la broche est en rotation. Cette contrainte peut être vue comme une contrainte de sécurité (casse de l'outil, casse de la pièce, ...) ou de fonctionnement (le tournage ne peut être réaliser).

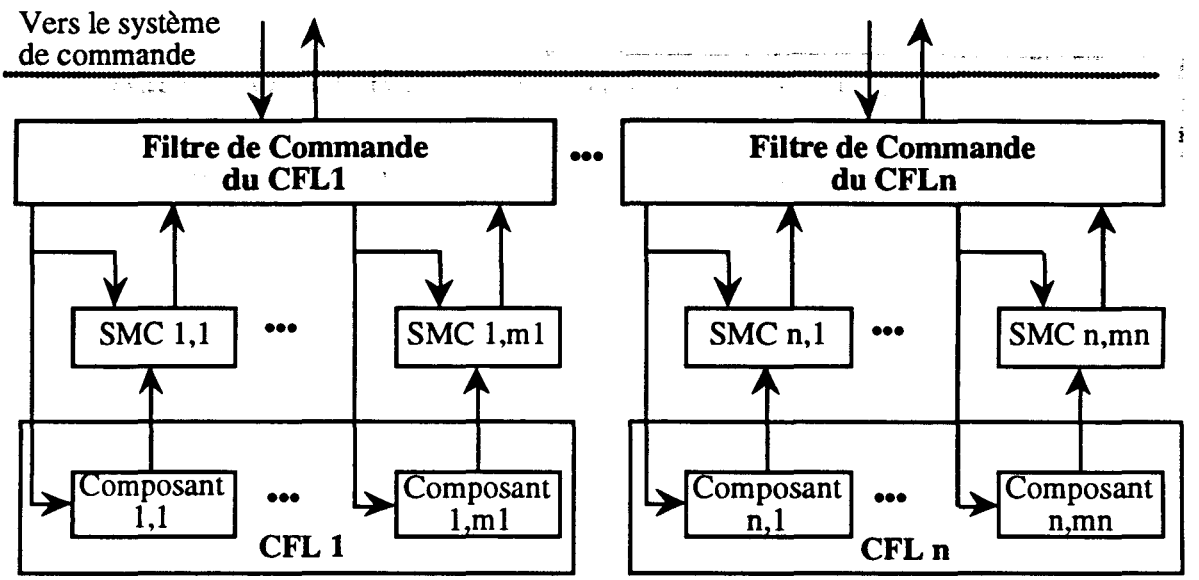


FIGURE VI.3 : INTERACTION ENTRE MFC ET MCC.

A chaque CFL, nous associons donc, un filtre de comportement. La répartition doit respecter cette association. En effet, le filtre est considéré comme indivisible et ne peut être réparti sur plusieurs organes d'implémentation.

II. CONCEPTION DES MFC.

Dans cette partie, nous allons nous intéresser aux différentes phases de conception des filtres. La démarche de conception [ELKHATTABI, 92A] prend en charge la modélisation d'un système physique, depuis sa spécification jusqu'à la génération automatique des filtres.

Une maquette de la conception est réalisée en langage PROLOG [O'KEEFE, 90], [STERLING, 86], afin de fournir une maquette de la conception (cf ANNEXE A). Nous exposerons au §III de ce chapitre, une application informatique qui prend en charge la génération automatique des filtres.

II.1. DÉCOMPOSITION STRUCTURO-FONCTIONNELLE.

Le comportement d'un système se limite à la vision extérieure de l'utilisateur. En effet, un système peut être considéré comme une boite noire qui interagit ou susceptible d'interagir avec d'autres systèmes. Le comportement du système est tout simplement ce qu'il fait. Une décomposition structurelle permet de développer la boite noire. D'après [AMAR, 90], cette démarche consiste à mettre en évidence les différents organes opératifs élémentaires constituant un système.

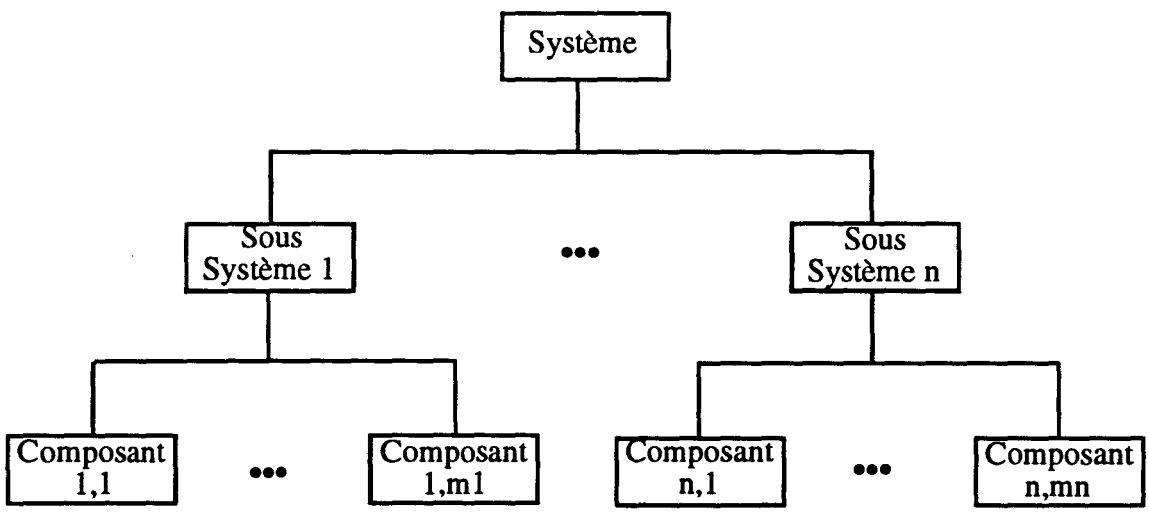


FIGURE VI.4 : DÉCOMPOSITION STRUCTURELLE D'UN SYSTÈME.

Cette approche est de nature descendante, elle est organisée en niveaux d'abstractions. Un système est ainsi décomposé en composants. Chaque composant est à son tour considéré comme un système (FIGURE VI.4). La décomposition s'arrête lorsque le système devient atomique. Le critère d'arrêt de la décomposition peut être défini par l'analyse prévisionnelle de la sûreté de fonctionnement. Cette démarche permet de mettre en évidence les relations d'interactions entre les différents composants de niveaux adjacents. Ces relations sont statiques, elles sont généralement physiques. Dans le cas d'un atelier flexible, il s'agit de caractériser les moyens de fabrication et de transport qui le constituent. Afin de répondre à notre objectif de modélisation, le niveau atomique est celui de

l'actionneur. En effet, la modélisation comportementale par OCE nécessite la connaissance des actions élémentaires que réalise chaque composant. Considérons l'exemple du tour. La décomposition structurelle du tour, génère le modèle suivant :

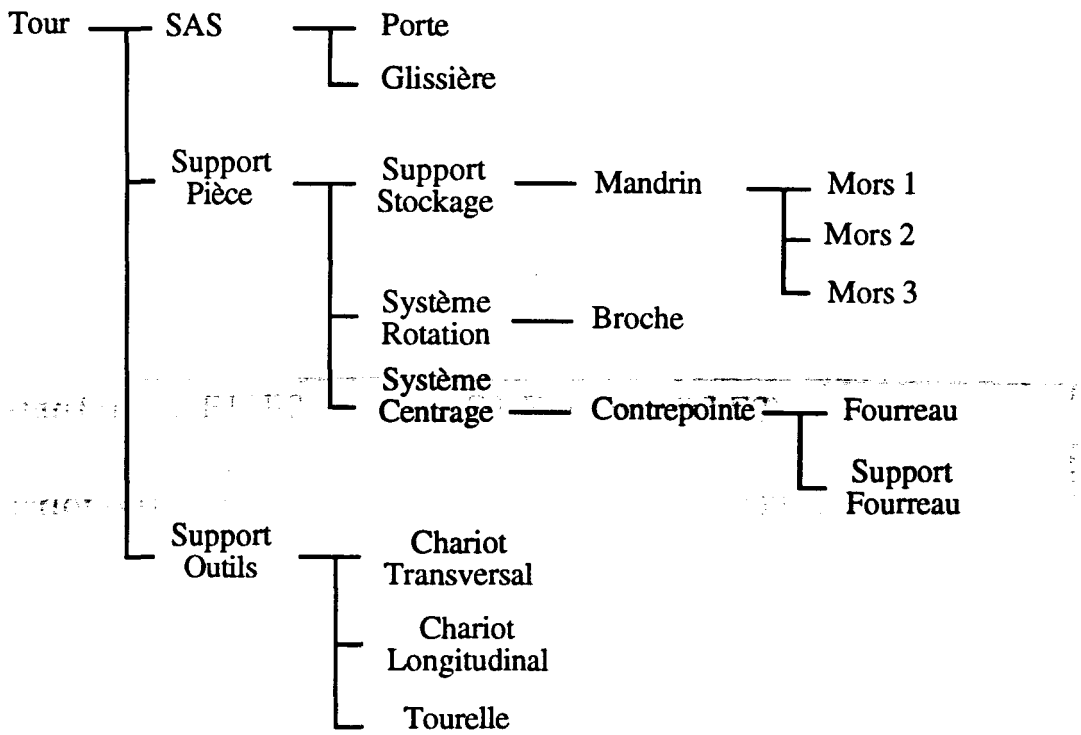


FIGURE VI.5 : DÉCOMPOSITION STRUCTURELLE DU TOUR.

L'aspect fonctionnel est intégré en associant à chaque composant élémentaire une ou plusieurs fonctions. S'agissant des actionneurs des composants, les fonctions seront des actions.

La définition d'un mode d'évolution représentant la dynamique de chaque action , détermine le comportement des composants élémentaires. Le comportement d'un composant est défini par la mise en relation des comportements élémentaires des actions qui lui sont associées. Une approche ascendante définit le comportement global d'un système par association des composants de chacun de ses constituants [AMAR, 92].

Cette décomposition, basée sur un critère opérationnel, permet de mettre en évidence les fonctionnalités de base pouvant être assurées par un composant élémentaire, définissant ainsi les actions de bas niveau réalisables par le procédé.

II.2. MODÉLISATION DU COMPORTEMENT DES COMPOSANTS.

A partir de la décomposition structuro-fonctionnelle, chaque composant élémentaire (actionneur) sera caractérisé par un ensemble d'états physiques stables (E) et un ensemble d'actions qu'il réalise. La spécification des comportements élémentaires détermine le comportement global du composant..

Les composants sont modélisés par des OCE. L'utilisation de cet outil comme modèle permet la validation de propriétés mathématiques : commandabilité, accessibilité, déterminisme.

Le modèle comportemental est constitué d'un ensemble d'états/transition où chaque transition est étiquetée par une action (FIGURE VI.8). Nous verrons au § III.5 l'aspect intégration avec le module de contrôle de commande.

La spécification d'un composant se fait par des clauses Prolog. Une clause sans corps, est un fait. A cet effet, nous avons défini quatre prédicats :

- **objet** (o) : o est un composant,
- **etat** (o,e) : e est un état du composant o,
- **action** (o,a) : a est une action du composant o
- **fonction** (o,e_i,a,e_f) : o est un composant. L'action a étiquette l'arc entre l'état e_i et l'état e_f.

Considérons le cas de la porte du tour. Les actions réalisables par le vérin de positionnement de la porte sont : "*avance-avant*" ("*fermer*" porte) et "*avance-arrière*" ("*ouvrir*" porte). Ces actions mettent la porte respectivement dans l'état "*fermée*" et "*ouverte*". Un prototype Prolog de la spécification du composant "porte", est donné FIGURE VI.6.

L'interprétation logique des deux dernières clauses est la suivante : quelque soit l'état E de la porte, l'action "ouvrir" (resp. "fermer"), la met dans l'état "ouverte" (resp. "fermée"). Après résolution, chacune de ces clauses, génère deux solutions puisque E peut prendre deux valeurs.

Une spécification des autres composants est réalisée de la même manière. Nous ne les présentons pas ici, afin d'alléger la lecture du mémoire.

```
objet (porte).

etat (porte, ouverte).
etat (porte, fermée).

action (porte, ouvrir).
action (porte, fermer).

fonction (porte, E, ouvrir, ouverte) :- etat (porte, E)
fonction (porte, E, fermer, fermée) :- etat (porte, E)
```

FIGURE VI.6 : SPÉCIFICATIONS PROLOG DU COMPOSANT "PORTE".

La clause "*etats(O,E)*" permet de définir l'ensemble E des états d'un composant O, accessibles par au moins un état de O (sauf par lui-même). "*Actions(O,A)*" détermine l'ensemble des actions réalisables par un composant O. L'ensemble U des arcs d'un composant O, est défini par la clause "*arcs(O,U)*". Ces trois ensembles (E, A et U) permettent de définir formellement un composant O.

```
arrive(O,E)        :- fonction(O,E1,_,E), not (E1==E).

etats(O,E)         :- objet(O), setof(UnEtat,arrive(O,UnEtat),E).

actions(O,A)       :- setof(UneAction,action(O,UneAction),A).

arc(O,(X,A,Y))     :- fonction(O,X,A,Y).
arcs(O,U)          :- setof(UnArc,arc(O,UnArc),U).
```

FIGURE VI.7 : ETATS, ACTIONS ET ARCS D'UN COMPOSANT.

Une modélisation OCE de l'objet "*porte*" est représenté par la FIGURE VI.8. Ce modèle paraît trivial, néanmoins, après intégration des contraintes, les modèles seront plus complexes.

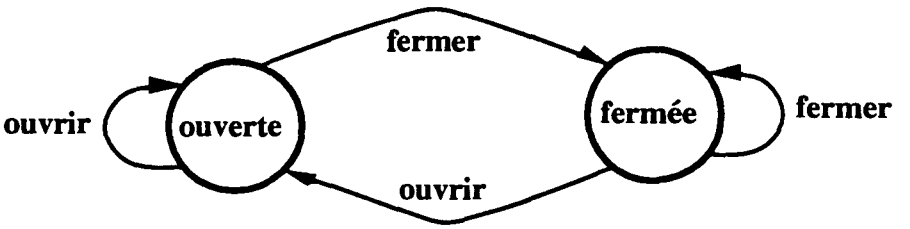


FIGURE VI.8 : MODÈLE OCE DE L'OBJET "PORTE".

II.3. VALIDATION DE LA MODÉLISATION DES COMPOSANTS.

La validation de la modélisation des composants élémentaires revient à vérifier les propriétés de commandabilité et d'atomicité (élémentaire). En effet, étant donné, que les composants modélisés sont élémentaires, leurs modèles doivent être commandables (accessible et déterministe) et élémentaires. Si, une de ces propriétés n'est pas vérifiée pour un composant, la décomposition structurelle est incomplète ou la spécification du composant est erronée. Une remise en cause des étapes précédentes s'impose.

```
act(O,X,A)          :- fonction(O,X,A,_).
acts(O,X,A)         :- setof(UneAction,act(O,X,UneAction),A).

det1(O,E)           :- actions(O,A1), acts(O,E,A2), A1==A2.
det2(O,[]).
det2(O,[X|L])       :- det1(O,X), det2(O,L).
det(O)              :- etats(O,E), det2(O,E).

par1(O,E)           :- etats(O,E1), terminals(O,E,E2), E1==E2.
par2(O,[]).
par2(O,[X|L])       :- par1(O,X), par2(O,L).
par(O)              :- etats(O,E), par2(O,E).

oce(O)              :- det(O), par(O).
```

FIGURE VI.9 : CLAUSES DE VÉRIFICATION DES PROPRIÉTÉS DES COMPOSANTS.

Cette démarche de validation des propriétés formelles des composants, est illustrée par un ensemble de clauses Prolog (FIGURE VI.9). A cet effet, nous avons utilisé

l'instruction "*setof*" [STERLING, 86] qui permet de déterminer l'ensemble des solutions possibles pour une clause. Ainsi, la clause "*acts*" définit l'ensemble A des "*UneAction*" vérifiant la clause "*act(O,X,UneAction)*".

La clause "*oce*" permet de valider les spécifications formelles d'un composant O : un composant sera dit commandable et élémentaire si "*oce(O)*" est vrai. La résolution de cette clause nécessite la définition de deux clauses "*det*" ("*det*" permet de valider la propriété de déterminisme complètement spécifié de O) et "*par*" ("*par*" détermine si le composant O est accessible parfait).

Un objet O est déterministe complètement spécifié si chaque état E de O l'est. La validation pour chaque état de cette propriété se fait par la clause "*det1*". Il suffit de vérifier l'égalité entre l'ensemble A1 des actions de O et l'ensemble A2 des actions réalisables depuis l'état. L'ensemble A des actions réalisables depuis un état E d'un composant O, est donné par la clause acts(O,E,A). La validation de la propriété d'accessibilté parfaite, suit le même raisonnement.

La validation de ces propriétés permet de remettre en cause les spécifications en cas de non conformité. Cette non conformité peut avoir deux raisons : la décomposition structurelle n'était pas atomique, ou les spécifications des composants sont incomplètes.

II.4. INTÉGRATION DES CONTRAINTES.

Si l'étape de validation est réalisée avec succès, le concepteur peut alors aborder la phase d'intégration des contraintes. Cette phase permet de spécifier les composants élémentaires contraints fonctionnellement. Ces contraintes de fonctionnement ont comme objectif d'assurer la sécurité matérielle et humaine, et la coopération du système de production.

Considérons le cas de la broche et du mandrin du tour. L'évolution parallèle et libre de ces composants, ne peut être autorisée. Prenons le cas où la broche est en rotation et le mandrin en cours de déchargement. Il y a un risque évident d'accident. L'évolution parallèle, doit être contrainte dans ce genre de situations. Cette contrainte réduit la flexibilité potentielle des moyens au minimum autorisé. L'intégration d'une telle contrainte permet d'interdire une action dans une configuration matérielle donnée. Nous pouvons donc garantir une évolution exempte d'accidents.

Les contraintes se formulent par l'interdiction de réaliser une action d'un composant quand un autre composant se trouve dans un état particulier. Une spécification en Prolog d'une contrainte se fait par le prédicat : contrainte(o1,e_o1,o2,a_o2). La contrainte de l'exemple précédent s'écrit comme suit : contrainte(broche,rotation,mandrin,desserer). Les mors du mandrin ne peuvent être desserrés tant que la broche est en rotation.

A titre d'exemple, nous donnons une liste des contraintes fonctionnelles entre les composants d'un tour :

- broche à l'arrêt lors du chargement/déchargement du mandrin,
- broche à l'arrêt lors de l'ouverture de la porte,
- contre-pointe à l'arrêt lors de l'ouverture de la porte,
- mandrin serré lors de l'ouverture de la porte,
- support outils (chariot transversal, chariot longitudinal et tourelle) à l'arrêt lors de l'ouverture de la porte,
- broche à l'arrêt lors du mouvement de la contre-pointe.

Ces contraintes conduisent à un regroupement des composants contraints en CFL. Le regroupement de deux composants O1 et O2 conduit à :

- la création d'un nouveau composant virtuel O1*O2 (CFL),
- la propagation des contraintes mettant en liaison l'objet O1 (resp. O2) avec un autre objet différent de O2 (resp. O1) sur le nouveau composant,
- la suppression des prédicats objet(O1) et objet(O2).

Le regroupement des composants se fait par paire. Un CFL est considéré à son tour comme un composant. Le regroupement s'arrête lorsqu'il n'y a plus de contraintes à exploiter.

La définition des clause "etat", "action" et "fonction" sont étendues aux CFL. Nous définissons ces clauses de manière générale. L'ensemble des états d'un CFL est le résultat d'un produit cartésien des ensembles des états des composants initiaux et l'ensemble des actions est une union des deux ensembles initiaux d'actions.

Seules les fonctions caractérisant les changements d'états, intègrent les contraintes. La première clause indique qu'il existe un arc reliant un état E1*E2 à un état E3*E2, étiqueté par A, si A de O1 permettait le passage de E1 à E3 et que A n'est contrainte avec aucun état E2 de l'objet O2. La deuxième clause suit le même raisonnement pour le composant O2.

La clause "*contraints*" permet de définir les états contraints d'un CFL. Ainsi, l'ensemble des états d'un CFL est réduit de ces états contraints. L'explosion de la combinatoire (produit cartésien des états), se trouve réduite. Nous pensons étudier dans l'avenir, de manière formelle, l'incidence des contraintes sur la réduction de la combinatoire. Nous allons illustrer ce regroupement par contraintes. Il s'agit de deux composants mandrin et porte. Le mandrin est soit "serré", soit "desserré". Les actions du mandrin sont au nombre de deux : "serrer" et "desserrer". Pour les spécifications de la porte se rapporter à la FIGURE VI.7. Les contraintes de fonctionnement entre ces deux composants sont les suivantes :

- **contrainte**(porte,fermée,mandrin,desserer)
- et **contrainte**(mandrin,desserré,porte,fermer).

```
etat(O1*O2,E1*E2)   :- etat(O1,E1), etat(O2,E2).

action(O1*O2,A)     :- action(O1,A) ; action(O2,A).

fonction(O1*O2,E1*E2,A,E3*E2) :- fonction(O1,E1,A,E3),
                    etat(O2,E2), not(contrainte(O2,E2,O1,A)).
fonction(O1*O2,E1*E2,A,E1*E3) :- fonction(O1,E2,A,E3),
                    etat(O1,E1), not(contrainte(O1,E1,O2,A)).

non_arrive(O,X)     :- objet(O), etat(O,X), not(arrive(O,X)).
contraints(O,EC)    :- objet(O), setof(X,non_arrive(O,X),EC).

regroupe            :- contrainte(O1,_,O2,_), atom(O1),
                    not(objet(O1*O2)), assertz(objet(O1*O2)).
```

FIGURE VI.10 : REGROUPEMENT DES COMPOSANTS CONTRAINTS.

Compte tenu des spécifications faites (objet porte, objet mandrin et contraintes entre porte et mandrin), la clause *regroupe* permet de générer un nouveau fait *objet(porte*mandrin)*. Ce fait indique la création d'un nouveau objet (CFL) *porte*mandrin*. Le résultat des interrogations "*actions(porte*mandrin,A)*", "*etats(porte*mandrin,E)*", et "*contraints(porte*mandrin,EC)*", est le suivant :

- *A=[ouvrir, fermer, serrer, desserrer],*
- *E=[ouverte*serré, ouverte*desserré, fermée*serré, fermée*desserré],*
- *EC=[fermée*desserré].*

A partir de ces résultats, nous pouvons donc représenter le modèle du CFL "*porte*mandrin*" par la FIGURE VI.11. Ce modèle est commandable. Nous remarquerons l'absence de l'état "*fermée*desserré*". En effet, cet état est un état interdit. Par conséquent, le filtrage des actions destinées à la porte et au mandrin ne peuvent être réalisées que si elles sont sensibles à un instant donné. Ainsi, la réalisation de l'action "*desserrer*" quand la porte est "*fermée*" et le mandrin "*serré*", est interdite et par conséquent, elle sera dite non conforme à l'état du procédé.

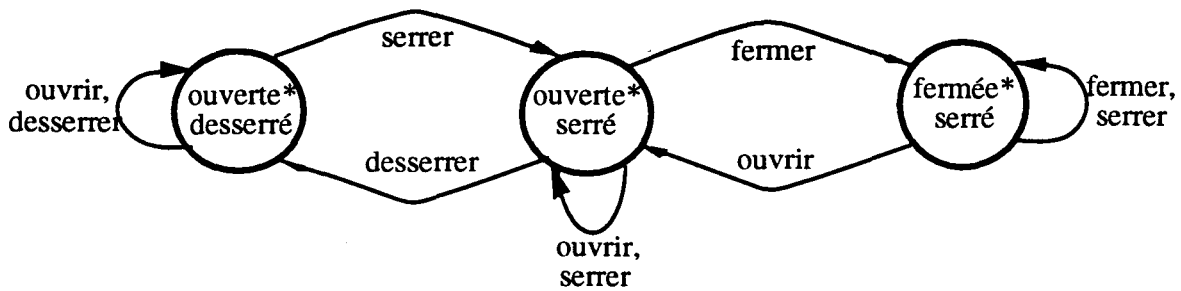


FIGURE VI.11 : MODÈLE DU CFL PORTE*MANDRIN.

Nous constatons que tous les composants qui constituent le tour, sont tous contraints entre eux par transitivité. Le tour forme donc un CFL. Le modèle comportemental du tour ne peut donc être réparti en plusieurs filtres : un filtre unique regroupe tous les comportements possibles des différents composants du tour. Ce filtre intègre implicitement tous les états interdits par soustraction au produit cartésien des états initiaux des différents composants.

II.5. VALIDATION DES MODÈLES DES CFL.

Il s'agit ici de valider les propriétés de commandabilté des CFL et surtout la propriété d'accessibilité.

Nous utiliserons l'exemple du Tour afin d'illustrer la validation du regroupement. La FIGURE VI.12 donne une spécification complète des différents composants du CFL Tour. Elle donne la liste des états et actions élémentaires de chaque composant, ainsi que les états d'arrivée suite à la réalisation d'une action. Les cases contenant un C, indiquent qu'il existe une contrainte dans cette situation. Le mandrin ne peut être desserré ("dma"), si la porte est fermée ("PF").

		Porte		Mandrin		Fourreau		Support Fourreau		CT		CL		Tourelle		B	
		PO	PF	MAS	MAD	FA	FR	SFA	SFR	CTA	CTR	CLA	CLR	TP	TR	BA	BT
P	Ouvrir (op)	PO	PO			C		C		C		C		C			C
	Fermer (fp)	PF	PF		C												
M	Serrer (sma)			MAS	MAS												
	Desserrer(dma)		C	MAD	MAD	C		C		C		C		C			C
F	Avancer (af)	C			C	FA	FA		C	C		C		C			C
	Retirer (rf)					FR	FR										C
SF	Avancer (asf)	C			C	C		SFA	SFA	C		C		C			C
	Retirer (rsf)					C		SFR	SFR								C
CT	Avancer (act)	C			C		C		C	CTA	CTA		C	C			C
	Retirer (rct)									CTR	CTR			C			C
CL	Avancer (acl)	C			C				C	C		CLA	CLA	C			C
	Retirer (rcl)									C		CLR	CLR				C
TO	Positionner (pt)	C			C		C		C		C		C	TP	TP		C
	Retirer (rt)													TR	TR		
B	Arrêter (ab)	C														BA	BA
	Tourner (tb)	C			C		C		C		C		C		C	BT	BT

FIGURE VI.12 : SPÉCIFICATIONS DU TOUR.

Le modèle comportemantal du CFL Tour, obtenu après regroupement, est représenté FIGURE VI.13. Les états e_i sont une combinaison des états des différents composants (cf. ANNEXE A3). Nous discuterons au §III de la réduction du modèle. Ce modèle est accessible et déterministe. Néanmoins, il n'est pas complètement spécifié. Ceci est une conséquence directe du regroupement qui implique l'élimination d'arcs contraints.

La propriété d'accessibilité est déterminante pour la validation du modèle. En effet, la non vérification de cette propriété, engendre une situation critique qui rendrait impossible la réalisation de la fonction principale du CFL (tournage pour notre exemple). De plus, l'existence d'états puits, amènerait indiscutablement à des situations de blocages indésirables. Cette situation peut être due à l'une des raisons suivantes :

- une mauvaise spécification, amenant à une mauvaise modélisation,
- une mauvaise conception matérielle des composants.

Le modèle du Tour garde la fléxibilité du séquencement des actions, possible. Cette flexibilité apparaît clairement sur le modèle. Atteindre l'état "e3" à partir de "e10" peut se faire de deux maières : réaliser "rt" suivie de "ab" ou "ab" suivie de "rt". Cette flexibilité n'est nullement altérée par le regroupement des composants. Cependant, elle élimine une flexibilité qui mettrait le système dans une situation indésirable.

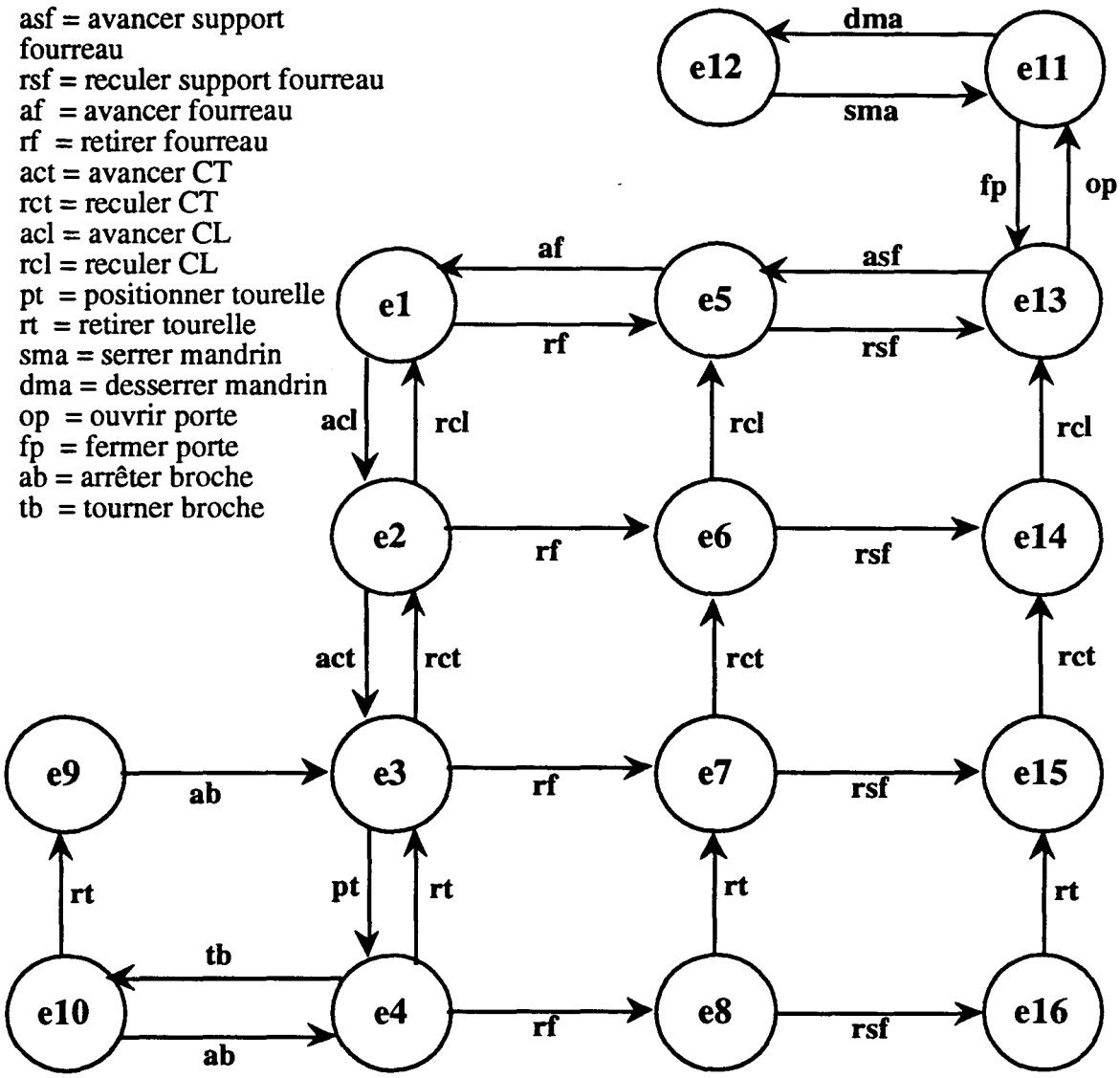


FIGURE VI.13 : MODÈLE DU CFL TOUR.

La non validation de l'accessibilité du CFL, remet en cause la spécification des contraintes par une relaxation de celles-ci. Si cette remise en cause de la spécification, ne conduit pas à l'obtention d'un modèle valide, nous serons donc amenés à remettre en cause l'architecture matérielle ou la nature des composants.

III. PRÉSENTATION DE L'APPLICATION COCE.

III.1. INTÉRÊTS.

La conception des filtres de commande, présente des risques d'erreurs et des lourdeurs de calcul dans le cadre des regroupements des composants contraints en CFL. Le cas du Tour illustre assez bien ces situations. Nous sommes en présence de 8 composants, à 2 états et 2 actions, liés par 64 contraintes (ANNEXE A2). Un regroupement engendrerait une combinatoire de 2^8 états (256 états) et 16×256 transitions (4096).

Une conception manuelle d'un tel filtre conduirait inévitablement à une erreur humaine de conception. C'est pour cette raison principale, que nous avons décidé de développer une application informatique destinée à la Conception des Objets Commandables Elémentaires (**COCE**) [ELKHATTABI, 93B].

COCE a été développé aussi dans le but d'offrir des fonctionnalités au concepteur afin de l'assister. Le cycle de développement doit être assuré par le concepteur, en lui permettant de revenir en arrière à tout instant.

COCE étant un outil de conception, notre objectif n'était pas de disposer d'un outil rapide en temps de réponse. Néanmoins, l'optimisation a été constamment prise en compte dans l'écriture des algorithmes. Un soin particulier a été réservé à la qualité du logiciel.

Le langage de développement de COCE, qui a été choisi est LELISP [ILOG, 89]. Ce choix a été motivé par deux raisons :

- l'existence d'une sur-couche graphique, appellée AIDA [ILOG, 92B], qui présente un ensemble de routines assez riche,
- le développement de l'interface graphique de CASPAIM, a été développé en LELISP et AIDA [HERBRARD, 92B].

III.2. FONCTIONNALITÉS.

Actuellement, l'application tourne en mode interprété. L'un de nos objectifs serait de la compiler afin de minimiser l'occupation mémoire du système. Le lancement de l'application active une fenêtre à l'écran représentée par la FIGURE VI.14.

L'application se compose de quatre zones :

- une barre de titre indiquant le nom du fichier de travail et son état (modifié ou non),
- une barre de menus donnant accès aux fonctionnalités de l'application,
- une zone centrale d'affichage de la spécification actuelle du système à modéliser,
- et une zone d'historique des opérations effectuées. Cette zone offre aussi la possibilité d'exécuter les fonctionnalités du système en mode intéractif.

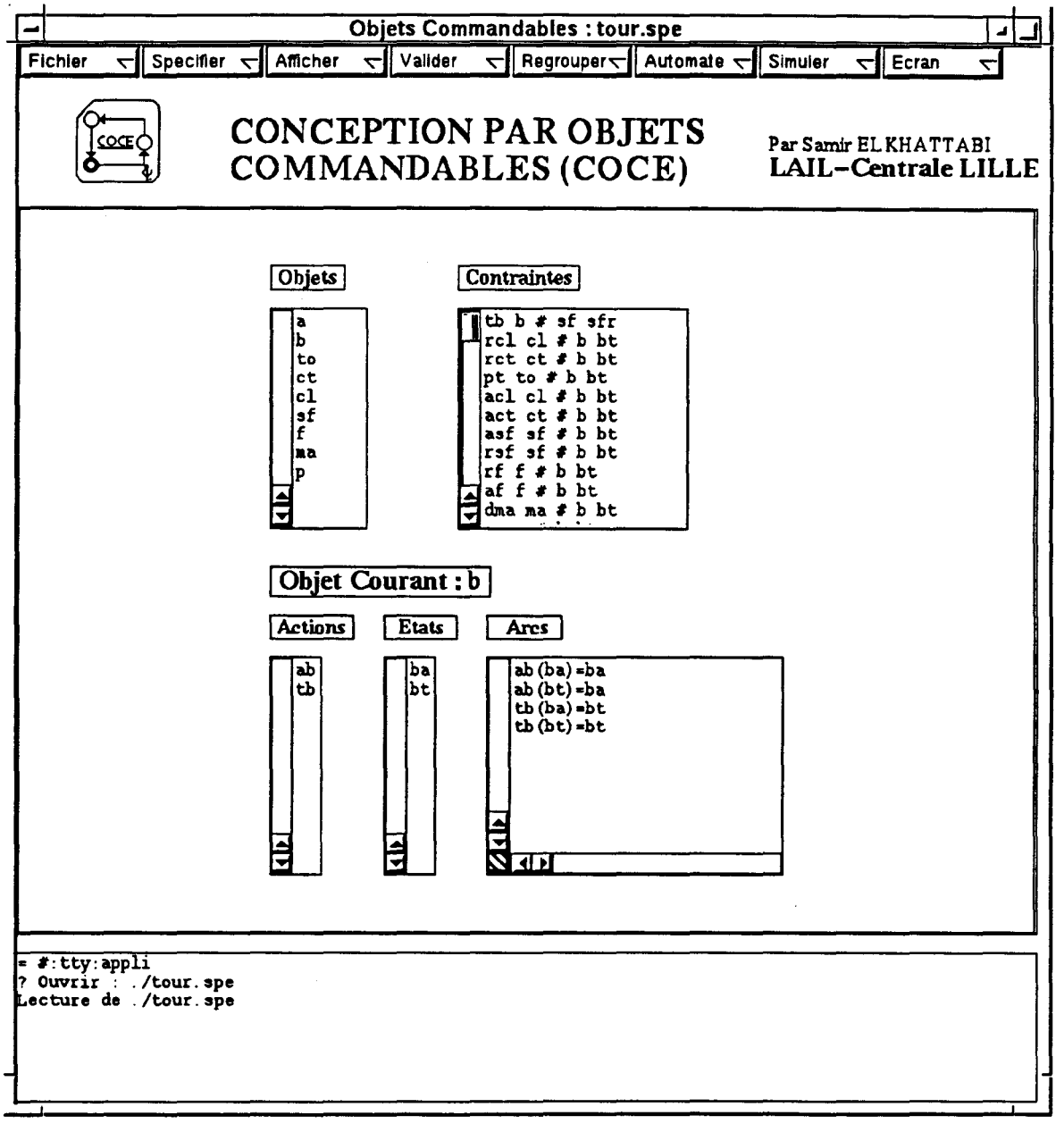


FIGURE VI.14 : APPLICATION COCE.

Les fonctionnalités de l'application sont regroupées par catégorie (Fichier, Spécifier, ...), formant une barre de menus (FIGURE VI.14). Chaque menu se compose de plusieurs items. Pour le menu "Fichier", on accède à toutes les fonctions d'entrée/sortie de l'application (ouvrir, fermer, enregistrer, imprimer, ...). La FIGURE VI.15 donne sous forme de tableau, une liste de tous les items, ainsi que les fonctionnalités assurées par ceux-ci.

Menu	Item	Fonctionnalité
Fichier	Nouveau	Création d'un nouveau environnement
	Ouvrir	Lire les données d'un fichier
	Fermer	Fermer l'environnement courant
	Enregistrer	Enregister les données dans un fichier
	Enregistrer Sous...	Enregister les données dans un fichier, sous un nouveau nom
	Imprimer	Imprimer les spécifications actuelles
	Quitter	Quitter l'application
Spécifier	Créer un Objet	Créer un nouveau Objet
	Modifier un nom d'Objet	Modifier le nom d'un Objet existant
	Supprimer un Objet	Supprimer un Objet existant
	Objet Courant	Choisir un Objet courant (Modification état, action, arc)
	Créer/Supprimer Etat	Créer ou Supprimer un état de l'Objet courant
	Créer/Supprimer Action	Créer ou Supprimer une action de l'Objet courant
	Modifier Arc	Modifier un arc existant
	Créer une Contrainte	Créer une contrainte entre deux Objets
	Supprimer une Contrainte	Supprimer une contrainte entre deux Objets
Afficher	Spécif d'1 Objet Courant	Afficher la spécification de l'Objet courant
	Objets	Afficher la liste des Objets créés
	Contraintes	Afficher la liste des Contraintes
Valider	Déterministe	Vérifier la propriété de déterministe complètement spécifié
	Accessible	Vérifier la propriété d'Accessibilité
	1-Accessible	Vérifier la propriété d'Accessibilité Directe
	Commandable	Vérifier la propriété de Commandabilité
	Commandable Elémentaire	Vérifier la propriété de Commandabilité Elémentaire
Regrouper	2 Objets Contraints	Regrouper deux Objets contraints en un CFL
	Tous les Objets Contraints	Regrouper tous les Objets Contraints en CFL
Automate	Générer Xautograph	Créer un fichier d'un Objet au format Xautograph
	Générer AGEL	Créer un fichier d'un Objet au format AGEL
	Lire Xautograph	Lire un fichier d'un Objet au format Xautograph
	Lire AGEL	Lire un fichier d'un Objet au format AGEL

FIGURE VI.15 : FONCTIONNALITÉS DE COCE.

Par la suite, il serait intéressant de développer une interface graphique de visualisation des modèles des OC, qui seront gérés sous forme de fenêtres, formant les items du menu "Ecran". La catégorie "Simuler", non développée, représente aussi un axe de développement. Elle permettrait de mettre en évidence sur le modèle graphique, l'état courant, les événements sensibles et l'évolution en fonction des événements sensibilisés. Il est à noter que les items de "Automate", ont été développés à des fins d'interfaçage. Il est ainsi possible d'importer/exporter des automates vers XAUTOGRAPH [ROY, 89] et AGEL [ILOG, 92A], [CHAILLOUX, 91B]. Nous insistons sur l'interface avec AGEL, puisqu'elle permet une compilation des filtres avec les modules de contrôle de commande (cf CHAPITRE VII).

Notons que l'algorithme de validation de l'accessibilité des OC, a été inspiré par l'algorithme de ROY destiné à la fermeture transitive des graphes [ROY, 62].

III.3. ALGORITHME DE REGROUPEMENT.

Nous ne présentons pas en détail les différents algorithmes de l'application. Un rapport technique a été réalisé à cet effet [ELKHATTABI, 93B]. Nous nous intéresserons néanmoins à l'algorithme de regroupement des composants. Considérons deux composants O_1 et O_2. L'algorithme se décompose en trois phases principales :

- création d'un nouveau CFL O_1-O_2 :
  - réaliser le produit des états de O_1 et O_2,
  - réaliser la somme des actions de O_1 et O_2,
  - générer les arcs, en éliminant les cas contraints,
  - supprimer les contraintes liant O_1 et O_2,
  - supprimer les composants O_1 et O_2,
- réduction du modèle :

  Tant qu'il existe un état inaccessible, il sera éliminé. Le traitement s'arrête lorsque tous les états sont accessibles.
- propager les contraintes :

  Pour chaque contrainte C_i

  si le second composant de C_i est O_1 ou O_2, alors remplacer le premier composant de C_i par O_1-O_2,

  sinon, si le premier composant de C_i est O_1 ou O_2, alors générer autant de contraintes qu'il existe d'états composés par l'état de O_1 ou de O_2 intervenant dans C_i, et supprimer C_i.

Un avantage certain de cette méthode, réside dans la taille des modèles générés. Ils sont les plus petits possibles, tout en faisant une description fine des comportements. Prenons le cas du Tour, formé de 8 composants. Chaque composant dispose de 2 états, 2 actions et 2 arcs (sans considérer les boucles). Un regroupement sans contraintes engendre un CFL à 256 états, 16 actions, et 3840 arcs (sans les boucles). Un tel modèle serait illisible et très difficile à valider. Le regroupement avec 64 contraintes, génére un CFL à 16 états, 16 actions et 30 arcs (sans les boucles). Dans tous les cas, le nombre d'actions doit être égal à la somme. Dans le cas contraire, nous avons une perte de fonctionnalité(s). Ce qui constitue une anomalie.

CONCLUSION.

La démarche de conception est systématique, et automatisée. Elle assiste le concepteur dans la génération des modéles des CFL. Le modèle final des CFL, constitué des comportements élémentaires de chaque composant, est minimal, et validé. Il intégre les contraintes de fonctionnement et de sécurité inhérentes au système physique. Les propriétés de validation permettent de garantir un fonctionnement sûr et exempt d'accident. La disponibilité matérielle est accrue.

De plus, le modèle comportemental, qui constitue le filtre de commande, peut être utilisé à différents niveaux du SC. Il peut servir comme modèle de base dans le cadre du pilotage temps réel ou encore du recouvrement d'erreurs.

Nous disposons, ainsi, d'un modèle vérifiant la propriété de commandabilité, réutilisable à plusieurs niveaux. Ce modèle pourrait être enrichi pour intégrer les contraintes de coopération entre CFL. La génération du modèle reste identique (cf §III pour l'algorithme de génération).

Une restriction de l'utilisation de cette approche, réside dans la difficulté à adapter ce modèle à des ressources complexes disposant de plusieurs emplacements de stockage (un convoyeur par exemple). Cette restriction est due à la non représentation de la dynamique par ce modèle.

PARTIE IV.

INTEGRATION DE LA SURVEILLANCE A CASPAIM.

Chapitre VII. Intégration de la surveillance à CASPAIM.

CHAPITRE VII.

INTÉGRATION DE LA SURVEILLANCE À CASPAIM.

INTRODUCTION.

Dans ce chapitre, nous allons nous intéresser à l'intégration de la surveillance à une démarche de conception des systèmes de commandes (CASPAIM). En premier lieu, nous exposerons de manière succinte, la démarche de conception proposée par CASPAIM. Le lecteur trouvera en ANNEXE B, un exposé de cette démarche appliquée à un exemple.

Dans un second temps, notre approche de la surveillance sera abordée ; ainsi que les interactions entre **M**odule de **C**ontrôle de **C**ommande (**MCC**) et **M**odule de **F**iltres de **C**ommande (**MFC**).

Dans un troisième temps, nous présenterons la démarche de la surveillance. L'exemple d'un SAS permet d'illustrer celle-ci. Un accent est mis à ce niveau sur la généricité de la démarche. La détection des défauts de capteurs est également abordée.

Enfin, nous terminerons par une présentation de l'intégration de ces modèles à CASPAIM.

I. LE PROJET CASPAIM.

I.1. HISTORIQUE DES TRAVAUX.

Au début des années 80, l'équipe SED du LAIL, créée à l'initiative du Professeur J.C. GENTINA, visait principalement l'étude des SFPM et notamment l'analyse et la conception d'un SC. De cette volonté de mise au point d'une méthodologie de conception de la partie logicielle des SC des SFPM, est né le projet CASPAIM1. L'objectif de ce projet était de prendre en charge la totalité des étapes du cycle de développement du SC depuis la définition du cahier des charges jusqu'à l'implantation finale sur site en utilisant une approche homogène. Cette méthodologie se voulait conforme à l'approche génie logiciel [BOUREY, 93].

L'équipe s'est très vite orientée vers l'utilisation de l'outil de modélisation Réseaux de Petri (RdP). Dans un premier temps, le modèle RdP Structuré (RdPS) a été adopté [CORBEEL, 79 ET 80], ensuite, les extensions RdP Adaptatifs et Colorés [PETERSON, 80], [JENSEN, 86] (RdPSAC) ont été ajoutées au modèle de base. En parallèle, les techniques classiques d'analyse des propriétés des RdP et les méthodes analytiques d'étude des performances du système ne pouvaient être utilisées qu'aux travers de simplifications des modèles. Cette restriction ne permet plus l'étude fine de certains comportements. La simulation a donc été l'unique méthode retenue pour la validation du système : analyse des performances, étude des blocages et indéterminismes, détermination des régimes transitoires et des modes de marche, ... Un simulateur de RdP a donc été développé à cet effet [CASTELAIN, 87]. A partir d'un graphe fonctionnel appelé *Prégraphe* [KAPUSTA, 88], [HEIZERLING, 88], un modèle structuré simulable de la PC, est développé [BOUREY, 88]. Une implantation du modèle de la commande, sur un réseau d'automates programmables industriels (API), est ensuite envisagée [CRAYE, 89].

I.2. LES LIMITES DE CASPAIM1.

Le projet CASPAIM1 s'est révélé satisfaisant pour des systèmes de production plutôt orientés usinage, peu complexes et peu sujets à évolution ultérieure. Néanmoins, lors d'une étude des limites de CASPAIM1 [CRUETTE, 91A], il s'est avéré que les processus de fabrication pour lesquels l'une des conditions suivantes était vérifiée, posaient des difficultés de modélisation :

- La flexibilité des moyens de transport est importante. Ce qui entraîne une explosion combinatoire du nombre de processus de transfert élémentaire,
- La flexibilité opératoire des pièces à assembler. En effet, CASPAIM1 ne prend en compte que des assemblages avec antériorité fixe de type palettisation,
- Le partage d'une zone opératoire d'un lieu physique par plusieurs ressources de production. Il s'agit de systèmes tels que les robots d'assemblage et de soudage,
- Le NH ne prend en compte que l'aspect pilotage temps réel et n'intègre pas le niveau ordonnancement et planification,
- La modélisation du procédé n'est pas assez fine et ne met pas en évidence tous les comportements élémentaires possibles,
- La non prise en compte des aspects surveillance et de modes de marches. Ce qui a comme conséquence de s'exposer à des risques d'accidents, blocage de la commande, ...
- La modification de l'architecture physique remet en cause toute la modélisation et oblige le concepteur à réétudier tout le système.

Ces limitations sont dues essentiellement à des techniques de modélisation [BOUREY, 93]. On a vu ainsi naître une deuxième version de CASPAIM. L'objectif initial de CASPAIM1 a été reconduit. Néanmoins, il fallait dissocier l'aspect opérationnel de l'aspect fonctionnel du SFPM et intégrer les aspects ordonnancement, planification et surveillance à travers une approche systémique. Cette nouvelle méthodologie, devrait également assurer une couverture plus large du cycle de vie d'un SAP en intégrant la possibilité d'étudier un système de production dont l'architecture matérielle n'ést pas encore définie.

I.3. VERS UNE NOUVELLE APPROCHE DES SFPM.

I.3.1. Décomposition des SFPM.

La décomposition systémique initiale d'un SFPM, étant satisfaisante à un haut niveau d'abstraction, est conservée. Néanmoins, la PC était conçue indépendamment du NH et donc son interfaçage avec celui-ci ne permettait pas de prendre en compte certains type de stratégies de fonctionnement. Pour répondre à ce besoin, un module baptisé *Interface*, a été introduit entre la PC et le procédé [HEBRARD, 92A]. Ce qui a conduit à une structuration de la PC en deux parties (FIGURE II.1). D'une part, la *Partie Commande Produit* (PCP) qui est chargée d'assurer le contrôle et le séquencement des opérations au niveau de chaque

gamme de fabrication considérée de façon isolée. La résolution des conflits posés par les évolutions simultanées de gammes concurrentes, au sens des contraintes d'utilisation des ressources de production, n'est pas traitée à ce niveau. D'autre part, l'*Interface* qui est chargée de satisfaire les différentes commandes émanant de la PCP ou de l'opérateur. L'interface doit alors coordonner les différentes ressources afin d'optimiser le fonctionnement du procédé selon une stratégie de fonctionnement imposée par le NH.

I.3.2. L'Interface

L'interface a pour rôle de coordonner les différentes ressources à partir des commandes transmises par la PCP [HEBRARD, 92A]. Il s'agit notamment de gérer l'allocation des emplacements physiques des ressources du point de vue des produits, et d'élaborer des commandes fines à destination du procédé en respectant une stratégie de fonctionnement définie par le NH.

Pour présenter le gestionnaire de ressources, il est nécessaire d'introduire deux niveaux de complexité concernant les ressources : les *ressources complexes* et les *ressources élémentaires*. Ces deux concepts sont basées sur la définition de *lieu caractéristique* [AMAR, 92]. Un lieu caractéristique est définit comme étant *un lieu physique, mobile ou non, pouvant recevoir un produit. Ce lieu peut être soit un lieu de transformation fonctionnelle, soit un lieu accessible depuis un lieu physique d'un autre moyen de production (accessibilité directe externe).*

La ressource sera dite complexe, si, elle comporte plusieurs lieux caractéristiques ou au moins un lieu non caractéristique. Dans le cas contraire, la ressource est appelée ressource élémentaire. Ainsi, un système de convoyage est considéré comme une ressource complexe : les zones tampons contiennent plus d'un emplacement pouvant accueillir des produits. En revanche, un robot portique qui ne transfère qu'un seul objet à la fois entre plusieurs lieux d'un système, est une ressource élémentaire.

Le gestionnaire des ressources élémentaires de l'Interface peut alors être décomposé en deux modules principaux (FIGURE VII.1) : un *allocateur de ressource* chargé de gérer les conflits d'accès, et les *graphes de commandes* [HEBRARD, 92B].

Pour les ressources complexes, deux modèles supplémentaires sont nécessaires. Il s'agit d'un *modèle comportemental* dont le but est d'assurer la cohérence entre les commandes transmises et l'état global de la ressource (une des utilisations possibles des

filtres de commande), et d'un *résolveur d'indéterminisme locaux* permettant de mettre en œuvre les stratégies de fonctionnement définies par le Niveau Hiérarchique.

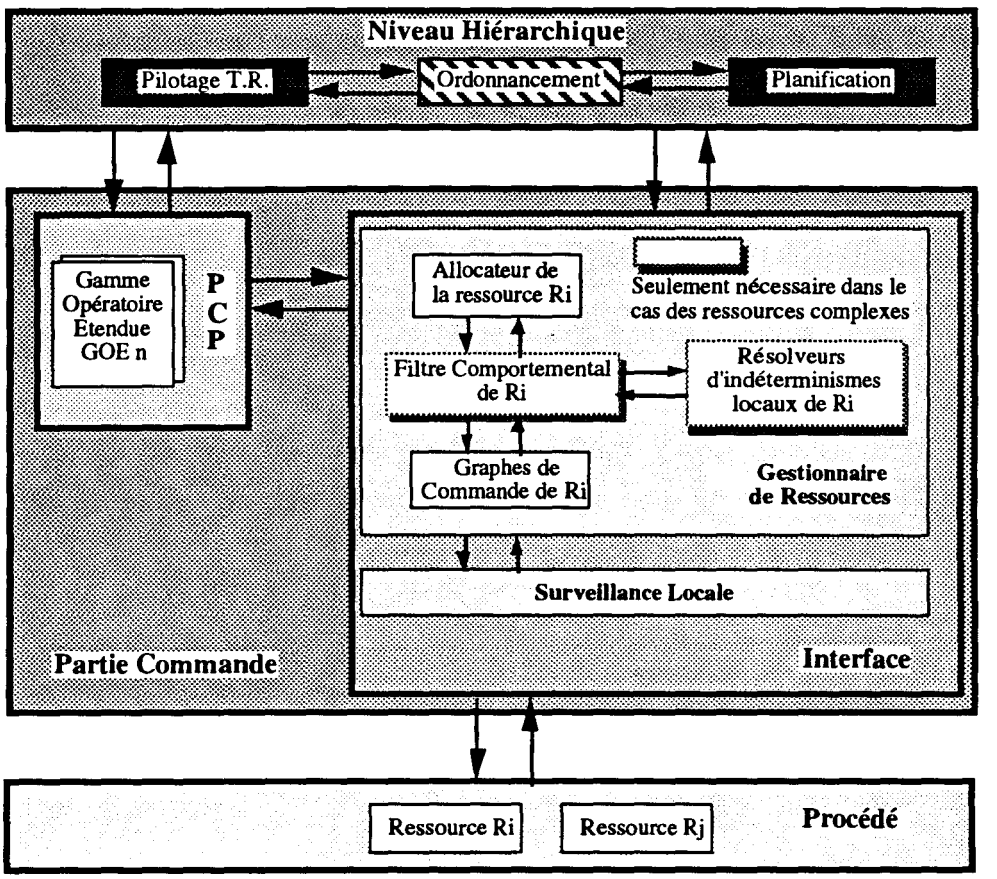


FIGURE VII.1 : DÉCOMPOSITION CONCEPTUELLE DE L'INTERFACE ET DU NH [HUVENOIT, 92].

Le modèle logique proposé dans CASPAIM2 est représenté sur la FIGURE VII.2. Les différentes fonctions représentées du schéma conceptuel sont décomposées et réparties au sein des différentes entités représentées. La décomposition et la répartition des fonctions ont été effectuées selon les critères des méthodes de conception orientée objets [BOOCH, 91], [COAD, 91] , [LAI, 91]. A partir du schéma logique, le niveau opérationnel est déduit en vue d'une implantation effective.

I.3.3. Modèles utilisés.

Le choix d'un autre modèle que les RdPSAC, s'est orienté vers des modèles de description de plus haut niveau. Il s'agit des RdP à Prédicats/Transitions [GENRICH, 87], des RdP à Structures de Données (RdPSD) [SIBERTIN, 85] et des RdP à Objets (RdPO)

[SIBERTIN, 90], [BASTIDE, 92]. Les deux derniers modèles possèdent une meilleure lisibilité inhérente à leur structuration, et permettent de mieux appréhender la complexité des SFPM par abstraction. Ainsi, des décisions de niveau pilotage, ordonnancement lors d'indéterminismes résultant de la flexibilité du système, peuvent être prises en compte.

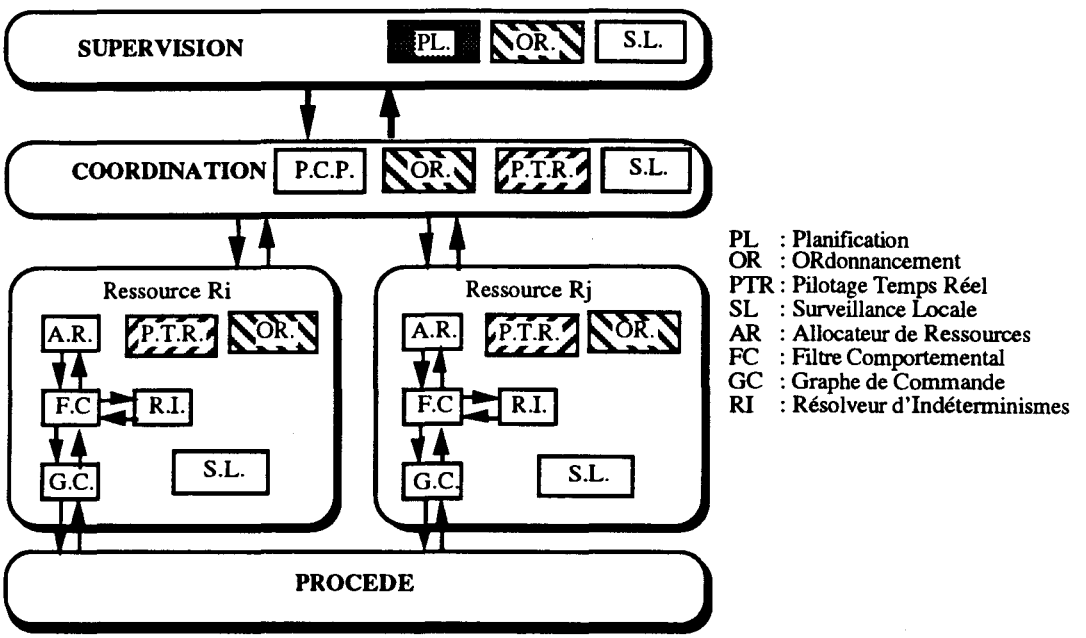


FIGURE VII.2 : SCHÉMA LOGIQUE D'UN SFPM [BOUREY, 93].

I.3.4. Dissociation fonctionnel /opérationnel.

L'une des limites de CASPAIM1, a amené l'équipe à dissocier l'aspect fonctionnel de l'opérationnel. En effet, l'aspect fonctionnel décrivant les opérations de transformations, devient indépendant de toute mise en œuvre : cette description se fait à travers la définition des gammes logiques [CRUETTE, 91B]. Quant à l'aspect opérationnel, il modélise les moyens de production et les relations d'accessibilité des lieux physiques des moyens. Cet aspect, permet de générer une flexibilité de routage des pièces [CRUETTE, 91A], [HEBRARD, 92A].

I.4. CONCLUSION.

L'ANNEXE B présente la démarche CASPAIM2 appliquée à un exemple. La méthodologie CASPAIM proposée par le LAIL, permet :

- une structuration de la démarche de conception d'un SFPM,

- une construction "propre" du SC intégrant des exigences très diverses,
- une intégration maximale de la flexibilité,
- une conception méthodologique, assistée, intégrant la conception, l'évaluation, et l'implantation.

Notons que, cette approche met en évidence une flexibilité de l'architecture de commande, qui est une transposition non restrictive de la flexibilité potentielle du SFPM. Ce qui facilite l'intégration des aspects ordonancement, surveillance et maintenance.

II. APPROCHE DE SURVEILLANCE PROPOSÉE.

L'approche proposée dans ce mémoire concerne les SED. Le comportement est vu de manière discrète. En effet, les états sont discrets, et l'évolution se fait à base d'événements discrets (l'information fournie par les capteurs est binaire). Dans le cadre de la surveillance, un (plusieurs) capteur(s) permet(tent) de générer un compte rendu (comportement normal) ou une erreur (comportement anormal).

Nous supposons que la commande est saine et respecte les contraintes fonctionnelles régissant la coopération des ressources du procédé. Le respect des contraintes fonctionnelles, défini un séquencement strict des commandes. Ces contraintes permettent d'éviter les risques d'accidents humains ou d'équipements et des collisions (sécurité), et d'assurer par là même une disponibilité des ressources. La commande étant saine, la surveillance a été basée sur la réalisation des commandes, respectant les contraintes : surveillance indirecte du procédé.

Suite à cette structuration, la surveillance est organisée autour d'un Composant Fonctionnel Logique (CFL). Les commandes sont adressées au filtre du CFL, représentant les comportements élémentaires des composants contraints, qui juge de leur validité. Ainsi, une commande non compatible avec l'état réel du procédé, conduit à une erreur de conception. Cette situation gèle la commande et nécessite une reconfiguration. Par contre, une commande valide, sera transmise au composant physique et activera son contrôle de manière simultanée. Le contrôle de la réalisation de la commande est basée sur la notion de "chien de garde" [ANDRÉ, 93], [ELKHATTABI, 93A]. La langage utilisé (ESTEREL) rend le contrôle réactif. La détection d'une défaillance, basée sur l'état des capteurs de commande, est ainsi synchrone à son apparition.

Un module de détection des défauts capteurs, complète cette structuration. Ce module posséde deux composantes. La première est dédiée à la détection des incompatibilités des états des capteurs de commande. La deuxième est optionnelle et sert d'interpréteur des capteurs de surveillance. La première peut être decrite de manière systématique puisqu'elle est liée au fonctionnement du composant. Quant à la deuxième, elle dépend de l'objectif visé par les capteurs dédiés à la surveillance. Cependant, ces capteurs peuvent être très utiles dans le cas de la redondance pour confirmer une détection de défauts de capteurs.

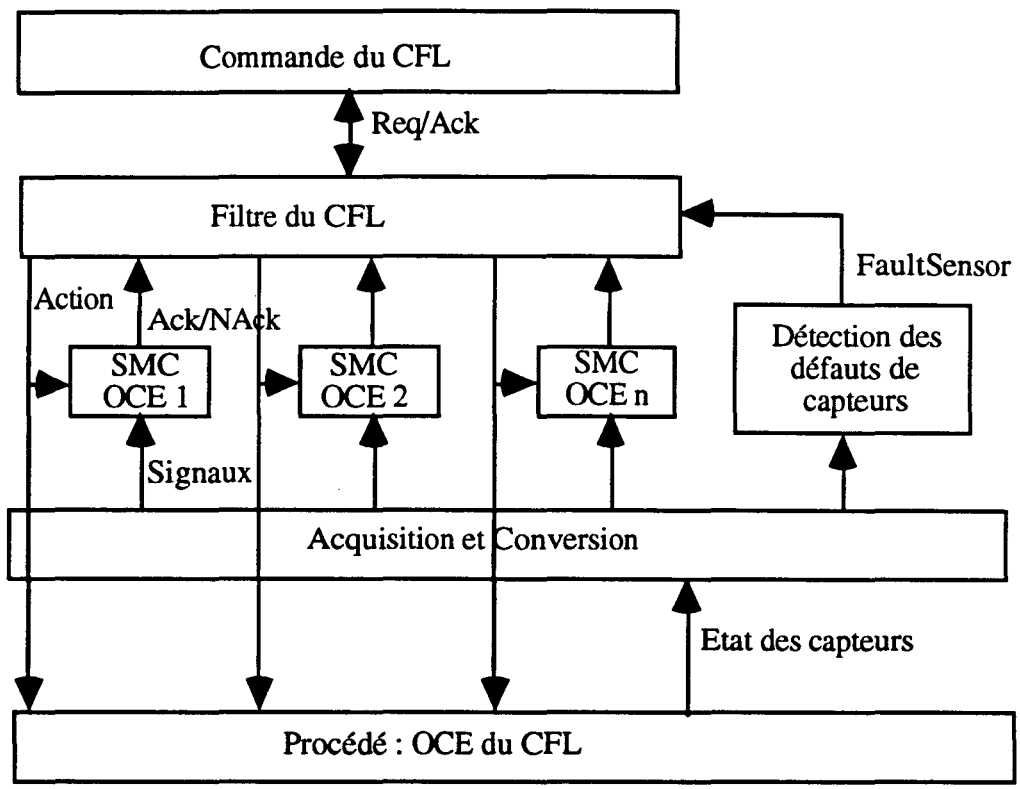


FIGURE VII.3 : ORGANISATION DE LA SURVEILLANCE DE BAS NIVEAU D'UN CFL.

La rapidité de détection et le séquencement strict des commandes permettent d'éviter la propagation des erreurs. En effet, la détection d'une défaillance gèle le système de commande du composant défaillant et le séquencement évite la "contagion" des autres composants. Nous serrons donc amenés à introduire un état "hors service" (HS) dans le modèle comportemental de chaque CFL.

L'organisation de la surveillance en modules (CHAPITRE IV), dédiés au contrôle des différents composants du procédé, rend les étapes de localisation et d'identification des composants défaillants, implicites et immédiates. L'identification des composants défaillants

déclenche un recouvrement d'erreurs qui peut engager éventuellement un processus de maintenance. Cette organisation nous a amené à réaliser, une surveillance de bas niveau, intégrée au SC.

III. CONTRÔLE ET FILTRE DE COMMANDE.

III.1. SPÉCIFICATION DE L'EXEMPLE : SAS D'UN TOUR NUMÉRIQUE.

A titre d'illustration, nous travaillerons sur le SAS d'un tour numérique. Nous avons délibérement choisi un exemple simple, afin de ne pas allourdir la présentation.

Action	Time Out Constant	Ack	Liste des Capteurs à 1	NAck	Liste des Capteurs à 0
Open	DO	isOpened	Hi1, Hi2	isNotOpened	Lo
Close	DC	isClosed	Lo	isNotClosed	Hi1, Hi2

FIGURE VII.4 : SPÉCIFICATION FONCTIONNELLE DE L'EXEMPLE.

Le SAS peut recevoir deux actions : Open et Close. Chacune des actions doit se réaliser dans un délai bien précis. Pour "Open" (resp. "Close"), ce temps est "DO" (resp. "DC") unités. Le MCC associé au SAS, renvoie "isOpened" (resp. "isClosed"), en cas de la réalisation de l'action "Open" (resp. "Close") dans les délais spécifiés. Dans le cas contraire, le MCC génére "isNotOpened" pour l'action "Open" et "isNotClosed" pour "Close". La génération de ces comptes rendus se fait par l'information recueillie auprès des capteurs. Pour chaque action, des capteurs doivent être à 1 et d'autres à 0. Pour l'action "Open", les capteurs à 1 sont "Hi1" et "Hi2". Sur la FIGURE VII.5, on retrouve les différents actionneurs et capteurs spécifiés précédemment, ainsi qu'un capteur suplémentaire, dédié à la surveillance, qui permet de détecter la présence d'un intrus. Cette détection permet l'arrêt du SAS afin d'éviter tout accident.

Le module d'acquisition permet de générer des signaux relatifs à l'état du procédé. Cette génération se fait à partir des capteurs de commande et doit tenir compte de la spécification. Les signaux générés par ce module sont destinés à la détection des défauts des capteurs et au contrôle de la réalisation des actions.

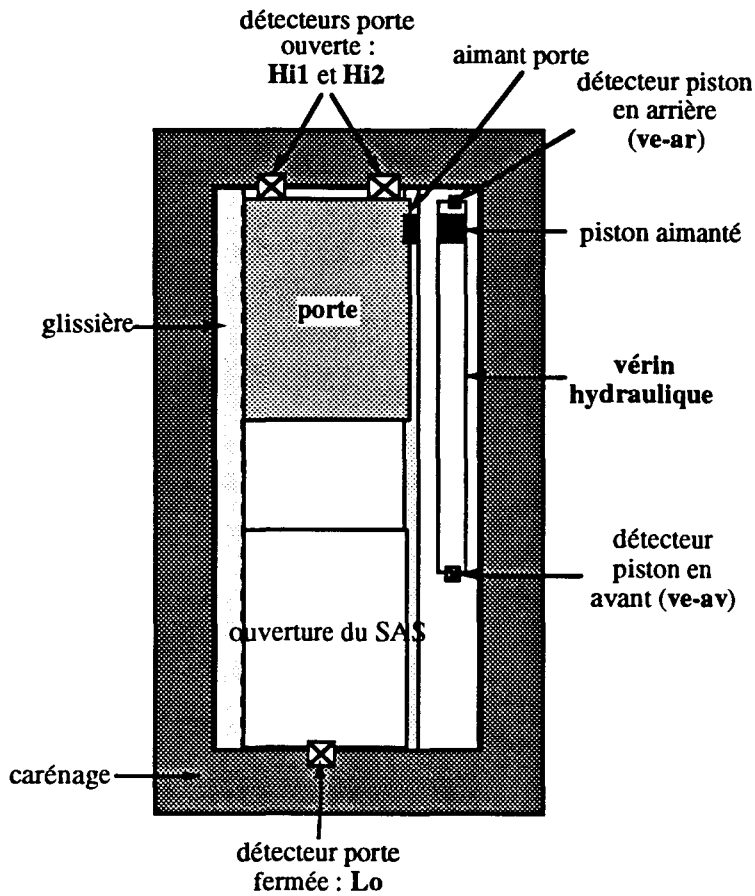


FIGURE VII.5 : EXEMPLE D'APPLICATION.

Le filtre de commande reçoit des ordres depuis le SC (FIGURE VII.6). Le filtrage de ces ordres conduit à leur transmission au module de contrôle ("FClose", "FOpen"). Le contrôle est subdivisé en sous modules. Chaque sous module est dédié au contrôle de la réalisation d'une action ("Closing", "Opening"). Cette subdivision a été introduite par souci de généricité et de modularité. A partir des signaux générés par le module d'acquisition, le contrôle confirme ou infirme la réalisation de la commande par une génération de compte rendu ("isFClosed", "isFOpened") ou d'une erreur ("CloseTo", "OpenTo", "isNotClosed", "isNotOpened"). Nous détaillerons par la suite la signification de chacun des signaux utilisés par les différents modules.

III.2. MODULE DE CONTRÔLE DE COMMANDE.

Le MCC suit la réalisation des actions filtrées par le procédé. Il est implémenté en ESTEREL. Cette implémentation permet d'assurer la réactivité de la détection et de la commande. Le MCC doit en plus du contrôle des actions détecter des erreurs de capteurs.

Etant dans un environnement synchrone, toute évolution doit être signalée de manière instantanée. Le module d'acquisition doit donc fonctionner de manière permanente. Il doit générer des signaux, qui vont servir à formuler (FIGURE VII.7 et VII.8) :

- des comptes rendus (ACK),
- des erreurs de réalisation (NACK),
- des TimeOut de réalisation (TO),
- des erreurs sur capteurs (FaultSensor).

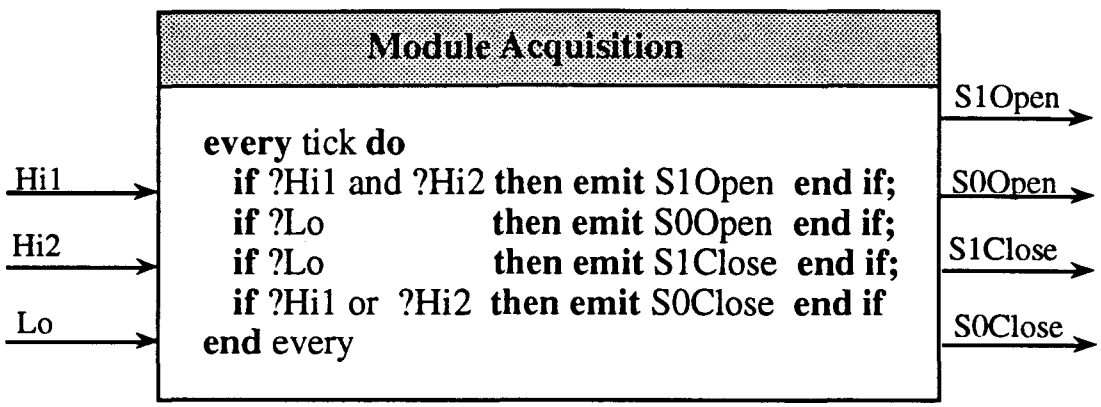


FIGURE VII.8 : MODULE D'ACQUISITION DE L'EXEMPLE.

Le module "acquisition" de l'exemple est représenté par le FIGURE VII.8. En entrée, on retrouve les capteurs de commandes et en sortie les signaux destinés à la génération de comptes rendus et d'erreurs. Il s'agit d'une boucle activée à chaque tick. Le signal tick étant présent en permanence, l'état du système est connu à tout instant.

III.2.2. Le Contrôle d'une commande.

Le contrôle d'une commande peut être représenté de manière générique par la FIGURE VII.9. Ce module se compose de plusieurs sous modules. "InterfaceAction" sert de générateur de comptes rendus et d'erreurs, et d'activateur de l'action "MAction" et du contrôle par chien de garde "ActioningTo". L'algorithme correspondant à ce module est donné FIGURE VII.10.

"MAction" est basé sur la primitive "Exec". Cette primitive permet d'interfacer les environnements synchrone et asynchrone. Il est ainsi possible de lancer une "tâche" asynchrone depuis un environnement synchrone. Ce module active l'exécution de l'action par le procédé.

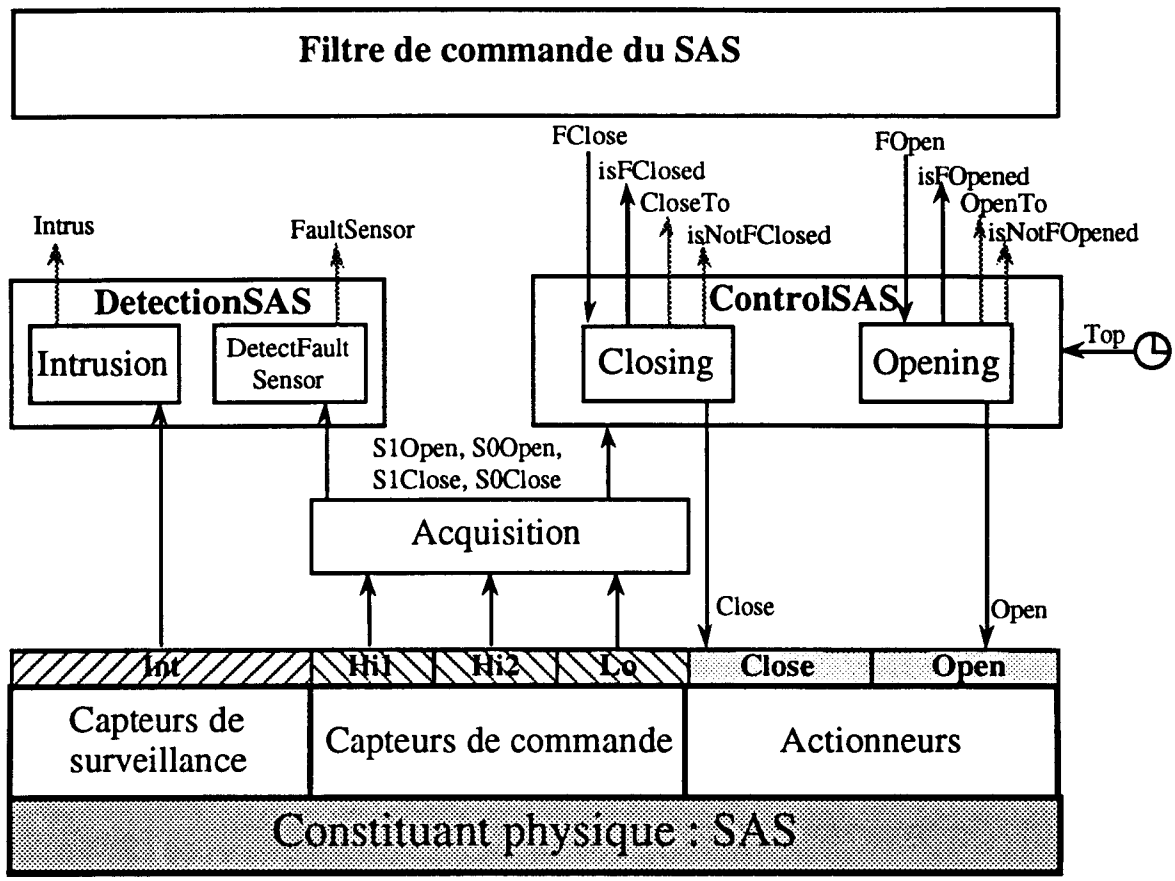


FIGURE VII.6 : SCHÉMA LOGIQUE DE L'EXMPLE.

III.2.1. Acquisition.

L'acquisition des informations auprès des capteurs, permet de traduire l'état du système sous forme de signaux : conversion des capteurs en signaux. Etant donné que la préemption en ESTEREL se fait par l'occurence d'un signal et non par la consultation de l'état d'un capteur, cette conversion est necéssaire. De plus, elle permet de gérer la combinaison des capteurs à 1 et à 0 pour chaque action du composant.

- Déclarer tous les capteurs de commande en entrée.
- Pour chaque action du composant :
  - **Si** le ET de tous les capteurs à 1, est vrai, **alors** générer **S1Action** (S1Close et S1Open).
  - **Si** le OU de tous les capteurs à 0, est vrai, **alors** générer **S0Action** (S0Close et S0Open).

FIGURE VII.7 : ALGORITHME GÉNÉRIQUE DE L'ACQUISITION.

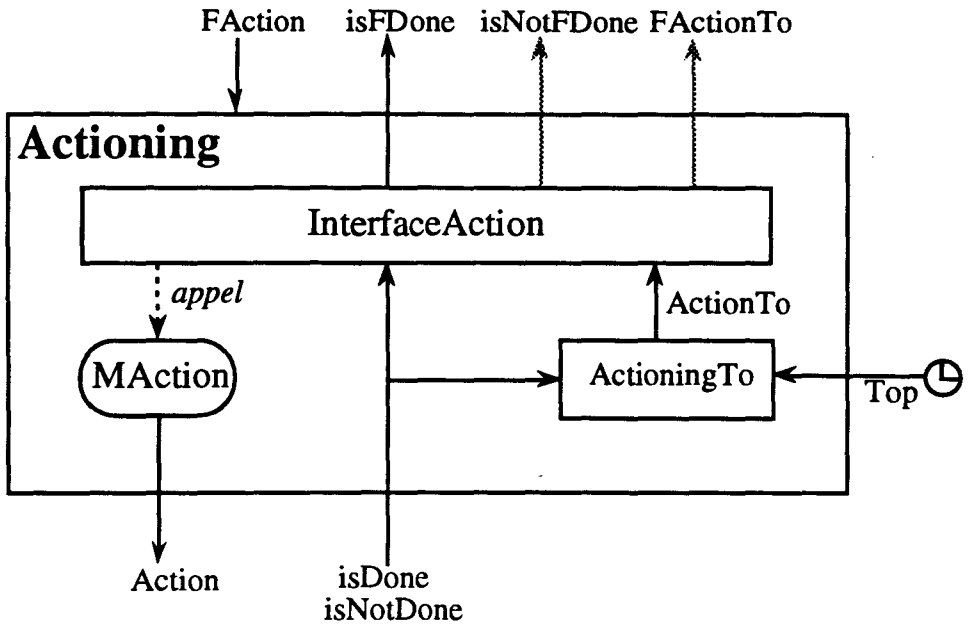


FIGURE VII.9 : SCHÉMA LOGIQUE DE "ACTIONING".

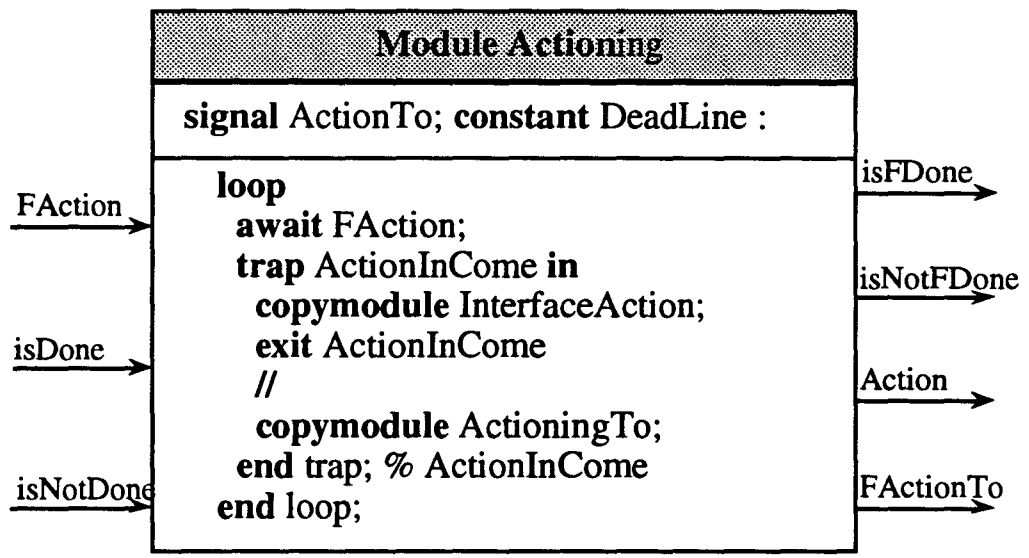


FIGURE VII.10 : ALGORITHME GÉNÉRIQUE DE "ACTIONING".

III.2.2.1. Module "InterfaceAction".

Ce module active en parallèle le module "MAction", l'attente du signal "isDone", l'attente du signal "isNotDone" et l'attente du signal "ActionTo". Le premier signal arrivé parmi ces trois permet de générer un signal correspondant vers le filtre de commande. Il s'agit en l'occurrence de "isFDone" dans le cas de "isDone", de "isFNotDone" dans le cas de "isNotDone", de "FActionTo" dans le cas de "ActionTo". Le premier signal est un

compte rendu. Les deux derniers caractérisent deux erreurs de type différent. Il s'agit soit de la non réalisation de l'action ou de la fin du "timeout". L'algorithme générique de ce module est donné par la FIGURE VII.11.

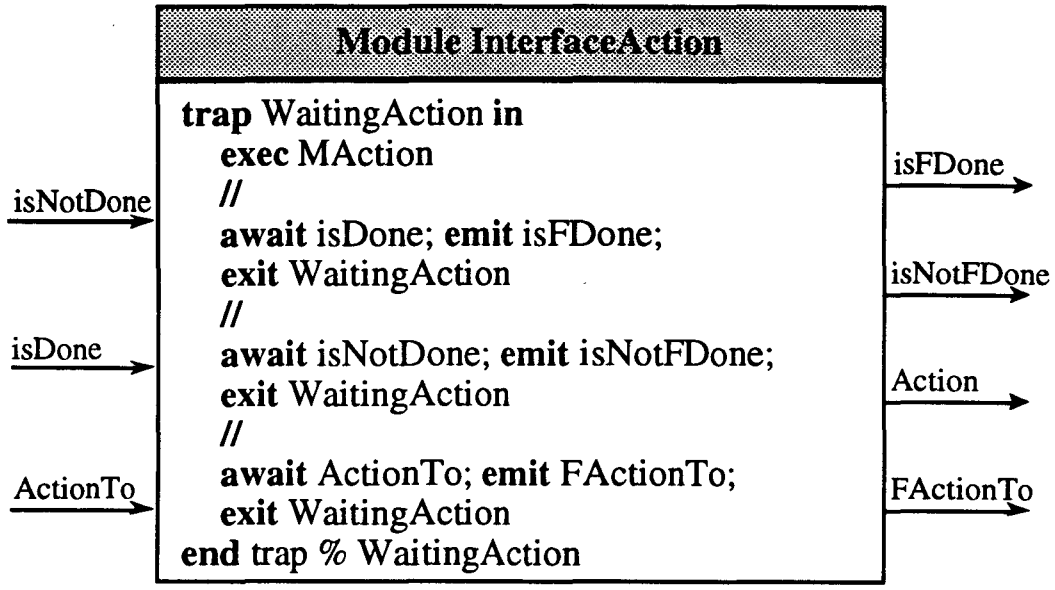


FIGURE VII.11 : ALGORITHME GÉNÉRIQUE DE "INTERFACEACTION".

III.2.2.2. Module "TO".

Le module générique "To", représenté par la FIGURE VII.12, est activé par "InterfaceAction". Ce module est activé pendant une durée maximale de "DeadLine Top" (Top étant une unité temps). Cette durée correspond à la limite supérieure de la fenêtre temporelle pendant laquelle l'action doit se réaliser. Le signal "ActionTo" est émis, si ce temps est écoulé. Notons que si la fin de l'action coincide avec la fin de la temporisation, "ActionTo" n'est pas émis.

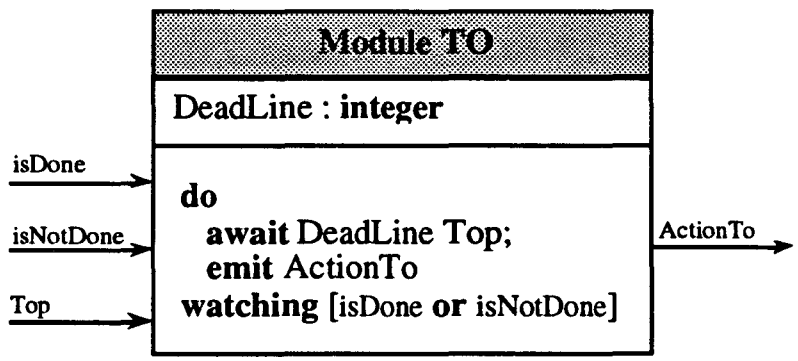


FIGURE VII.12 : MODULE GÉNÉRIQUE DE "TO".

III.2.2.3. Conclusion.

La démarche de contrôle présentée, offre les avantages suivants :

- généricité des modules et algorithmes utilisés,
- génération des ACK,
- génération des NACK et des TO,
- détection des erreurs de bas niveau,
- réactivité de l'approche (programmation synchrone).

En effet, à partir des ordres de commandes filtrées, le contrôle de commande assure une évolution au SC conforme à l'état du procédé. Il permet également au filtre de disposer d'une image reflétant l'état réel du procédé. Ce module est complété par un module de détection des défauts de capteurs, que nous présentons dans le paragraphe suivant.

III.3. DÉTECTION DES DÉFAUTS DE CAPTEURS.

Le module de détection [ANDRÉ, 93] des défauts des capteurs vérifie de manière permanente la compatibilité de l'état des capteurs de commandes. Ce module prend éventuellement en charge l'interprétation des capteurs de surveillance. Nous nous intéressons seulement aux capteurs de commandes.

Pour chaque action, "Acquisition" génére deux signaux ("S0Action" et "S1Action"). "S1Action" est présent si le "et logique" des capteurs qui doivent être à 1, est vrai; et "S0Action" est présent si le "ou logique" des capteurs qui doivent être à 0, est vrai. Ces deux signaux sont compatibles. En effet, ces deux signaux doivent être présents simultanément. Ainsi, si cette condition n'est pas remplie, nous concluerons sur une défaillance de l'un des capteurs concernés dans les combinaisons précédentes.

Une autre situation de défaillance des capteurs de commandes, fait intervenir les signaux de plusieurs actions d'un même composant. Il s'agit de l'exploitation de l'exclusivité des actions. Ainsi, quand "S0Action-i" est présent, alors tous les "S1Action-j" ($i \neq j$) doivent être absents. La FIGURE VII.13 donne toute les combinaisons possibles des capteurs de commandes du composant SAS. Elle met en évidence, la signification et les signaux générés dans chaque situation.

Hi1	Hi2	Lo	S1Open	S0Open	S1Close	S0Close	S1Open et S0Open	S1Close et S0Close	S0Open et non S1Close	S0Close et non S1Open	Commentaires
0	0	0	0	0	0	0	0	0	0	0	*Transitoire*
0	0	1	0	1	1	0	0	0	0	0	*isFClosed*
0	1	0	0	0	0	1	0	0	0	1	***FaultSensor isNotFClosed***
0	1	1	0	1	1	1	0	1	0	1	***FaultSensor isNotFClosed isNotFOpened***
1	0	0	0	0	0	1	0	0	0	1	***FaultSensor isNotFClosed***
1	0	1	0	1	1	1	0	1	0	1	***FaultSensor isNotFClosed isNotFOpened***
1	1	0	1	0	0	1	0	0	0	0	*isFOpened*
1	1	1	1	1	1	1	1	1	0	0	***FaultSensor isNotFClosed isNotFOpened***
			Hi1 et Hi2	Lo	Lo	Hi1 ou Hi2					

FIGURE VII.13 : TABLEAU DES COMBINAISONS DES CAPTEURS DU SAS.

Chaque capteur de surveillance a sa propre signification et un usage particulier auquel il est destiné. Dans le cadre des défaillances des capteurs de commandes, il serait utile d'introduire une redondance de ceux-ci. Cette démarche ne devrait pas être systématique, mais motivée par une étude préalable de la nature des capteurs et de leur taux de défaillance.

III.4. FILTRE DE COMMANDE.

Le filtre de commande d'un CFL représente les comportements élémentaires des différents composants, et les contraintes de fonctionnement entre composants. Pour notre exemple, s'agissant d'un composant élémentaire, cet aspect n'a pas été pris en compte (cf CHAPITRE VI). Il s'agit donc d'un modèle simple à deux états ("Opened" et "Closed"), dont les transitions sont étiquetées par "Open" et "Close".

Ce modèle vérifie les propriétés requises. En effet, il est commandable élémentaire (déterministe complètement spécifié et 1-accessible). La spécification est donc considérée comme valide. Ce composant est évidement contraint avec les autres composants du Tour. Le modèle complet du Tour est donné FIGURE VI.13. Le modèle du SAS, doit être interfacé d'une part avec le module de contrôle de commande et le système de commande, correspondants.

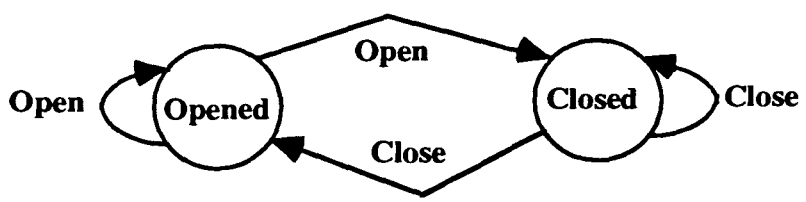


FIGURE VII.14 : MODÈLE COMPORTEMENTAL DU SAS.

IV. INTÉGRATION DES DIFFÉRENTS MODÉLES.

IV.1. INTERFACE DU FILTRE ET DU SYSTÈME DE COMMANDE.

L'intégration au système de commande consiste à étendre le modèle comportemental qui sert d'interface. Cette extension consiste à développer le mode transitoire de chacune des transitions (les boucles ne sont pas considérées). Ainsi, un arc sera représenté par un arc

"début d'action", un état transitoire "action en cours", et un arc "fin d'action" (cf FIGURE. V.7). L'état transitoire, représente un état non stable du système.

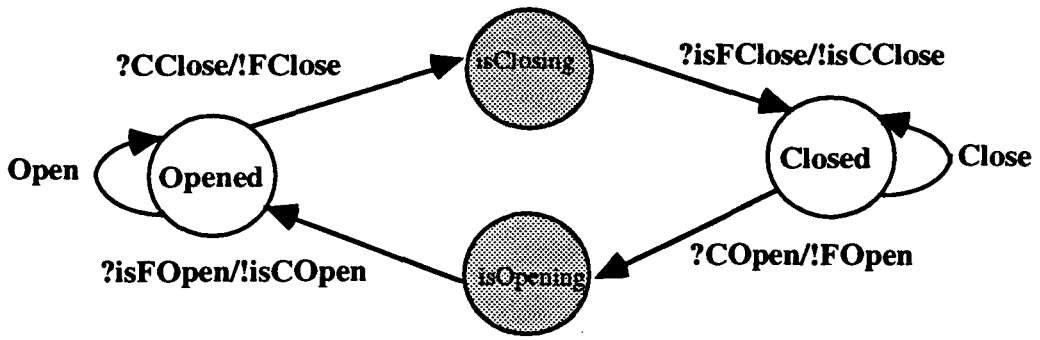


FIGURE VII.15 : INTÉGRATION DES ÉTATS TRANSITOIRES.

L'arc de début de transition est étiquetté par un réception d'ordre "?CAction", qui déclenche une émission d'ordre de réalisation de l'action "!FAction". Inversement, l'arc de fin d'action est étiquetté par la réception d'un compte rendu "?isFAction", qui déclenche une émission de compte rendu de réalisation de l'action "!isCAction". Le "C" devant "Action" indique qu'il s'agit du Système de Commande et le "F" du filtre de commande. Ainsi "?CClose" correspond à une attente d'ordre du SC de fermeture du SAS et "!isFClose" à une émission de compte rendu de fermeture du SAS.

IV.2. INTERFACE DU FILTRE ET DU MODULE DE CONTRÔLE DE COMMANDE.

L'interface a été réalisée de manière implicite (cf FIGURE VII.6). Néanmoins, le filtre doit être étendu, en lui ajoutant un état "Hors Service" noté "HS", afin de pouvoir prendre en compte les cas d'erreurs. Nous ne discriminons pas l'erreur ni par son composant, ni pour sa cause.

Le modèle de la FIGURE VII.16, permet de représenter l'état "HS" qui est accessible depuis les états transitoires. L'apparition d'une erreur n'est possible qu'en cours de réalisation de l'action. Le filtre reste dans cet état tant que le composant défaillant n'a pas été remis en marche. La remise en marche implique la remise en service du composant qui passe obligatoire par une phase de reprise. Le compte rendu est ainsi généré par la reprise (recouvrement). Le filtre peut donc évoluer vers un état stable. L'apparition d'une erreur, mettant le filtre dans un état "HS", le géle de manière implicite et lui interdit toute évolution, puisque le compte rendu de la commande n'a pas été émis. La réparation quant à elle permet

au filtre de générer un accusé et par là même dégeler la commande. Le SC reprend ainsi le contrôle de son évolution.

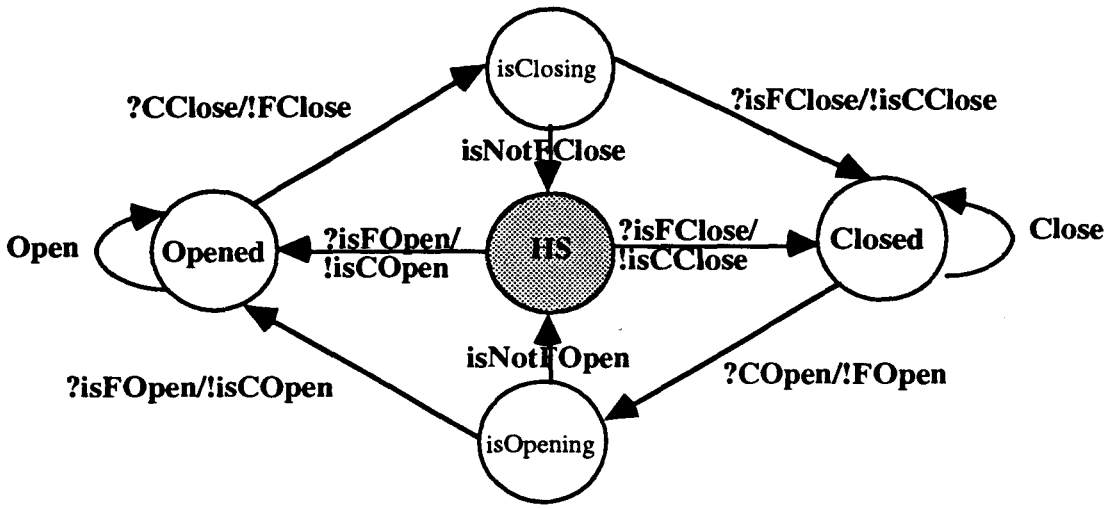


FIGURE VII.16 : INTÉGRATION DEL'ÉTAT "HS".

IV.3. INTÉGRATION INFORMATIQUE DU FILTRE ET DU SMC.

La conception du filtre d'un CFL ou d'un composant se fait de manière systèmatique et assistée. L'outil utilisé à cet effet est COCE [ELKHATTABI, 93B]. La conception des modules de contrôle se fait à base de modules génériques explicités au §III. La programmation de ces modules se fait en ESTEREL, en utilisant l'environnement de développement AGEL [ILOG, 92A]. Cet environnement permet l'écriture de programmes réactifs en ESTEREL ou sous forme graphique représenté par un automate à états finis. ESTEREL ne modifie nullement le modèle automate. Néanmoins, il procéde à une vérification du déterminisme du modèle.

Les modèles générés par COCE sont des automates. Ces automates représentant les filtres, sont transcrits dans le format AGEL. Il suffit par la suite de les intégrer à la description ESTEREL du SMC, en considérant chaque automate comme étant un module ESTEREL. La compilation génére ainsi un automate final, intégrant le contrôle et le filtre.

L'intégration du filtre au SMC, ne pose aucun problème particulier à la validation, si ces deux modèles ont été validés préalablement. Par contre, le nombre d'états de l'automate final s'accroit. Sachant que ces deux modèles évoluent de manière parallèle, le modèle final reprèsente la combinatoire de ceux-ci. AGEL permet de plus une verification des propriétés

des automates et une simulation in vivo du modèle final. La simulation permet ainsi de valider l'ensemble de la conception.

CONCLUSION.

La commande étant saine, la surveillance a été basée sur la réalisation des commandes, respectant les contraintes de sécurité et de fonctionnement par un suivi indirect des commandes. Ce qui a conduit à une structuration de la surveillance, organisée autour d'un Composant Fonctionnel Logique (CFL). Les commandes sont adressées au filtre du CFL, représentant les comportements élémentaires des composants contraints, qui juge de leur validité. La non compatibilité de la commande, gèle le SC et nécessite une reconfiguration. Par contre, une commande valide, sera transmise au composant physique et activera son contrôle de manière simultanée.

Le contrôle de la réalisation de la commande est basée sur la notion de "chien de garde". Le langage utilisé (ESTEREL) rend le contrôle réactif. La détection d'une défaillance, basée sur l'état des capteurs de commande, est ainsi synchrone à son apparition. L'organisation de la détection en modules, dédiés au contrôle des différents composants du procédé, rend les étapes de localisation et d'identification des composants défaillants, implicites et immédiates. L'identification des composants défaillants déclenche un recouvrement d'erreurs qui peut engager éventuellement un processus de maintenance. Cette organisation nous a amené à réaliser, une surveillance de bas niveau, intégrée au SC.

Un module de détection des défauts capteurs, compléte cette structuration. Ce module posséde deux composantes. La première est dédiée à la détection des incompatibilités des états des capteurs de commande. La deuxième est optionnelle et sert d'interpréteur des capteurs de surveillance. La première peut être decrite de manière systématique puisqu'elle est liée au fonctionnement du composant. Quant à la deuxième, elle dépend de l'objectif visé par les capteurs dédiés à la surveillance. Cependant, ces capteurs peuvent être très utiles dans le cas de la redondance pour confirmer une détection de défauts de capteurs.

La conception du filtre d'un CFL ou d'un composant se fait de manière systèmatique et assistée. L'outil utilisé à cet effet est COCE. La conception des modules de contrôle se fait à base de modules génériques. La programmation de ces modules se fait en ESTEREL, en utilisant l'environnement de développement AGEL. La compilation génére ainsi un automate final, intégrant le contrôle et le filtre.

CONCLUSION

GENERALE.

CONCLUSION GENERALE.

Nous avons présenté dans ce mémoire une approche de la surveillance en ligne des SED, concernant particulièrement les ateliers flexibles de production manufacturière. Cette approche a été motivée par trois objectifs principaux : garantir la sécurité humaine et celle des équipements, accroître la disponibilité matérielle et enfin assurer la fiabilité du SC. Pour répondre à ces objectifs, nous avons structuré le système de surveillance en deux modules principaux : le **M**odule de **C**ontrôle de **C**ommande (**MCC**) et le **M**odule des **F**iltres de **C**ommande (**MFC**).

Le système de surveillance proposé, doit être développé dans le cadre d'une politique générale de la maintenance et conçu en même temps que le SC. Nous avons également montré les intérêts apportés par l'élaboration d'une étude prévisionnelle de la sûreté de fonctionnement dans le cadre de la définition ou l'amélioration d'une architecture matérielle.

Notre approche est basée d'une part sur la surveillance indirecte du procédé et d'autre part le filtrage des commandes.

La réactivité du système de détection et l'organisation interne du MCC rendent les étapes de localisation et d'identification des composants défaillants implicites. Cette détection se fait par un traitement synchrone des événements générés par le procédé : une approche événementielle de la détection. La programmation synchrone, nous a donc permis d'assurer un contrôle réactif. La détection instantanée des défaillances, rend le recouvrement des erreurs rapide et évite la contagion d'autres composants du système. Ces deux caractéristiques ont un effet direct sur la disponibilité des composants du système physique.

Parmi les langages synchrones, notre choix s'est porté sur ESTEREL, pour son expression explicite des contraintes temporelles et son orientation "activités". Il est particulièrement apte à programmer les SED qui doivent réagir à des séquences de stimuli .

La décomposition du MCC en **S**ous **M**odule de **C**ontrôle (SMC) associés aux composants, confère au système un certain nombre de propriétés intéressantes :

- modularité et généricité,
- indépendance entre SMC,
- déterminisme de la réaction par rapport aux événements d'entrée,
- lisibilité et facilité de validation,
- intégration aisée au système de commande.

Le filtrage des commandes, intégrant les contraintes de sécurité et de fonctionnement, active la disponibilité du procédé et garantit la sécurité humaine et matérielle.

La démarche de conception des filtres de commande, est systématique, automatisée et assiste le concepteur dans la génération des modèles des **C**omposants **F**onctionnels **L**ogiques (**CFL**). Les objets commandables ont été développés afin de modéliser le comportement du système physique. Le modèle final des CFL, constitué des comportements élémentaires de chaque composant, est minimal, et validé. Il intègre les contraintes de fonctionnement et de sécurité inhérentes au système physique. Les propriétés de validation permettent de garantir un fonctionnement sûr et exempt d'accident. La disponibilité matérielle est accrue. Une application informatique (COCE ou Conception des Objets Commandables Elémentaires) a été conçue afin d'assister le concepteur dans la modélisation des CFL.

De plus, le modèle comportemental, qui constitue le filtre de commande, peut être utilisé à différents niveaux du système de commande. Il peut servir comme modèle de base dans le cadre du pilotage temps réel ou encore du recouvrement d'erreurs.

L'environnement de développement AGEL et l'outil COCE permettent de fournir un modèle final, validé et garantissant les objectifs initiaux.

Enfin, l'intégration de la surveillance à un SC conçu par la méthodologie CASPAIM, a été abordée. Cette intégration est illustrée par un exemple de composant d'atelier flexible.

Plusieurs axes de recherche se dégagent et permettront du point de vue des perspectives de prolonger le travail entrepris. En premier lieu, il faut réaliser le module de recouvrement d'erreurs et le module de traitement d'anomalies, afin de rendre le système de surveillance complet.

Le deuxième axe est de rechercher une méthodologie permettant de rendre générique le système de surveillance préconisé ici, à d'autres niveau de la hiérarchie des systèmes automatisés. Il s'agit des niveaux pilotage temps-réel, coordination et supervision.

Un troisième axe concerne le développement d'une interface graphique pour COCE afin de visualiser et simuler dynamiquement les modèles.

Enfin, il serait intéressant de voir comment modéliser le comportement des ressources complexes disposant de plusieurs emplacements de stockage. Ce problème pourrait trouver une solution dans une extension des objets commandables. Il s'agit de pouvoir exprimer la dynamique du système modélisé.

ANNEXES.

Annexe A. Maquette Prolog et Fichiers COCE du Tour.
Annexe B. Démarche CASPAIM2 appliquée à un exemple.
Annexe C. Programme Esterel d'un SAS.
Annexe D. Glossaire.

ANNEXE A.

MAQUETTE PROLOG ET FICHIERS COCE DU TOUR.

Annexe A1. Maquette Prolog .
Annexe A2. Fichiers COCE du Tour : spécifications.
Annexe A3. Fichiers COCE du Tour : regroupement.

ANNEXE A1.

MAQUETTE PROLOG.

```
objet(porte).
objet(broche).
objet(moteur).

etats(porte,[ouverte,fermee]).
etats(broche,[serree,desserree]).
etats(moteur,[marche,arret]).

etat(O,X) :- etats(O,E), in(X,E).

actions(porte,[ouvrir,fermer]).
actions(broche,[serrer,desserrer]).
actions(moteur,[tourner,arreter]).

action(O,B) :- actions(O,A), in(B,A).

arcs(porte,[    (ouverte,ouvrir,ouverte), (fermee,ouvrir,ouverte),
                (ouverte,fermer,fermee), (fermee,fermer,fermee)]).
arcs(broche,[   (serree,desserrer,desserree), (desserree,desserrer,desserree),
                (serree,serrer,serree), (desserree,serrer,serree)]).
arcs(moteur,[   (arret,arreter,arret), (arret,tourner,marche),
                (marche,arreter,arret),(marche,tourner,marche)]).

arc(O,V) :- arcs(O,U), in(V,U).
arc_in(O,U,(E1,A,E2)) :- arc(O,(E1,A,E2)), in(E1,U).
```

```
contrainte(porte,fermee,broche,desserrer).
contrainte(porte,ouverte,moteur,tourner).
contrainte(broche,desserree,porte,fermer).
contrainte(broche,desserree,moteur,tourner).
contrainte(moteur,marche,porte,ouvrir).
contrainte(moteur,marche,broche,desserrer).

contraints(O1,O2) :- contrainte(O1,_,O2,_).
contraints(O1,O2) :- contrainte(O2,_,O1,_).

arc_croix(O1*O2,(X*Z,B,Y*Z)) :-
        etat(O1*O2,X*Z),
        etat(O1*O2,Y*Z),
        action(O1,B),
        arc(O1,(X,B,Y)),
        not(contrainte(O2,Z,O1,B)).
arc_croix(O1*O2,(Z*X,B,Z*Y)) :-
        etat(O1*O2,Z*X),
        etat(O1*O2,Z*Y),
        action(O2,B),
        arc(O2,(X,B,Y)),
        not(contrainte(O1,Z,O2,B)).

arrive(O,X) :- arc(O,(Y,_,X)), not(X==Y).
etat_cont(O,X) :- objet(O), etat(O,X), not(arrive(O,X)).
etats_cont(O,E) :- ensde(X,etat_cont(O,X),E).

propage_cont11(O1*O2):-not(contrainte(O1,_,_,_)).
propage_cont11(O1*O2):-
        contrainte(O1,E,O3,A),
        retract(contrainte(O1,E,O3,A)),
        assertz(contrainte(O1*O2,E*_,O3,A)),
        propage_cont11(O1*O2).
```

```
propage_cont12(O1*O2):- not(contrainte(_,_,O1,_)).
propage_cont12(O1*O2):-
        contrainte(O3,E,O1,A),
        retract(contrainte(O3,E,O1,A)),
        assertz(contrainte(O3,E,O1*O2,A)),
        propage_cont12(O1*O2).

propage_cont21(O1*O2):-not(contrainte(O2,_,_,_)).
propage_cont21(O1*O2):-
        contrainte(O2,E,O3,A),
        retract(contrainte(O2,E,O3,A)),
        assertz(contrainte(O1*O2,_*E,O3,A)),
        propage_cont21(O1*O2).

propage_cont22(O1*O2):- not(contrainte(_,_,O2,_)).
propage_cont22(O1*O2):-
        contrainte(O3,E,O2,A),
        retract(contrainte(O3,E,O2,A)),
        assertz(contrainte(O3,E,O1*O2,A)),
        propage_cont22(O1*O2).

propage_conts(O1*O2):-
        propage_cont11(O1*O2),
        propage_cont12(O1*O2),
        propage_cont21(O1*O2),
        propage_cont22(O1*O2).

object(O, E, A, U)    :- objet(O), etats(O,E), actions(O,A), arcs(O,U), !.
regroupe(O1*O2,E,A,U)     :-
        contraints(O1,O2), assertz(objet(O1*O2)),
        object(O1,E1,A1,U1), object(O2,E2,A2,U2),
        conc(A1,A2,A), assertz(actions(O1*O2,A)),
        prod(E1,E2,E3), assertz(etats(O1*O2,E3)),
        ensde(V,arc_croix(O1*O2,V),U3), assertz(arcs(O1*O2,U3)),
        etats_cont(O1*O2,E4), moins(E4,E3,E),
        assertz(etats(O1*O2,E)), retract(etats(O1*O2,E3)),
        ensde(V,arc_in(O1*O2,E4,V),U4), moins(U4,U3,U),
```

```
        retract(arcs(O1*O2,U3)), assertz(arcs(O1*O2,U)),
        retract(objet(O1)), retract(etats(O1,E1)),
        retract(actions(O1,A1)), retract(arcs(O1,U1)),
        retract(objet(O2)), retract(etats(O2,E2)),
        retract(actions(O2,A2)), retract(arcs(O2,U2)),
        retract(contrainte(O1,_,O2,_)), retract(contrainte(O2,_,O1,_)),
        propage_conts(O1*O2).

in(X, [X|L]).
in(X, [Y|L]) :- in(X,L).

conc([],L,L).
conc([X|L1],L2,[X|L]) :- conc(L1,L2,L).

prod1(X,[],[]).
prod1(X,[Y|L1], [X*Y|L]) :- prod1(X,L1,L).

moins1(X, [X|L], L) :- ! .
moins1(X, [Y|L1], [Y|L]) :- moins1(X, L1, L).

moins([],L,L).
moins([X|L1], L2, L) :- moins1(X, L2, L3), moins(L1,L3,L).

prod([],L,[]).
prod([X|L1], L2, L) :- prod1(X,L2,L3), prod(L1,L2,L4), conc(L3,L4,L).

ensde(X,P,L) :- setof(X,P,L), !.
ensde(_,_,[]).
```

ANNEXE A2.

FICHIER COCE DU TOUR : SPÉCIFICATIONS.

```
(plist 'env '(nom tour.spe nouveau () sauve t objet b l-objs (b to ct cl sf f ma p)
l-ctes (cte64 cte63 cte62 cte61 cte60 cte59 cte58 cte57 cte56 cte55 cte54 cte53
cte52 cte51 cte50 cte49 cte48 cte47 cte46 cte45 cte44 cte43 cte42 cte41 cte40
cte39 cte38 cte37 cte36 cte35 cte34 cte33 cte32 cte31 cte30 cte29 cte28 cte27
cte26 cte25 cte24 cte23 cte22 cte21 cte20 cte19 cte18 cte17 cte16 cte15 cte14
cte13 cte12 cte11 cte10 cte9 cte8 cte7 cte6 cte5 cte4 cte3 cte2 cte1)
l-rgts () l-auts ()))

(setq nb-ctes 64)

(setq b '#:objet:#[b (ab tb) (ba bt) ((ba ba) (bt bt)) 2 2])
(setq to '#:objet:#[to (rt pt) (tr tp) ((tr tr) (tp tp)) 2 2])
(setq ct '#:objet:#[ct (rct act) (ctr cta) ((ctr ctr) (cta cta)) 2 2])
(setq cl '#:objet:#[cl (rcl acl) (clr cla) ((clr clr) (cla cla)) 2 2])
(setq sf '#:objet:#[sf (rsf asf) (sfa sfr) ((sfr sfr) (sfa sfa)) 2 2])
(setq f '#:objet:#[f (af rf) (fa fr) ((fa fa) (fr fr)) 2 2])
(setq ma '#:objet:#[ma (sma dma) (mas mad) ((mas mas) (mad mad)) 2 2])
(setq p '#:objet:#[p (op fp) (po pf) ((po po) (pf pf)) 2 2])

(setq cte64 '#:contre:#[cte64 sf sfr b tb])
(setq cte63 '#:contre:#[cte63 b bt cl rcl])
(setq cte62 '#:contre:#[cte62 b bt ct rct])
(setq cte61 '#:contre:#[cte61 b bt to pt])
(setq cte60 '#:contre:#[cte60 b bt cl acl])
(setq cte59 '#:contre:#[cte59 b bt ct act])
(setq cte58 '#:contre:#[cte58 b bt sf asf])
(setq cte57 '#:contre:#[cte57 b bt sf rsf])
(setq cte56 '#:contre:#[cte56 b bt f rf])
```

```
(setq cte55 '#:contre:#[cte55 b bt f af])
(setq cte54 '#:contre:#[cte54 b bt ma dma])
(setq cte53 '#:contre:#[cte53 b bt p op])
(setq cte52 '#:contre:#[cte52 to tp cl acl])
(setq cte51 '#:contre:#[cte51 to tp ct act])
(setq cte50 '#:contre:#[cte50 to tp sf asf])
(setq cte49 '#:contre:#[cte49 to tp f af])
(setq cte48 '#:contre:#[cte48 to tp ma dma])
(setq cte47 '#:contre:#[cte47 to tp p op])
(setq cte46 '#:contre:#[cte46 to tp ct rct])
(setq cte45 '#:contre:#[cte45 to tr b tb])
(setq cte44 '#:contre:#[cte44 cl clr to pt])
(setq cte43 '#:contre:#[cte43 cl clr ct act])
(setq cte42 '#:contre:#[cte42 cl clr b tb])
(setq cte41 '#:contre:#[cte41 cl cla sf asf])
(setq cte40 '#:contre:#[cte40 cl cla f af])
(setq cte39 '#:contre:#[cte39 cl cla ma dma])
(setq cte38 '#:contre:#[cte38 cl cla p op])
(setq cte37 '#:contre:#[cte37 ct cta cl rcl])
(setq cte36 '#:contre:#[cte36 ct cta cl acl])
(setq cte35 '#:contre:#[cte35 ct cta sf asf])
(setq cte34 '#:contre:#[cte34 ct cta f af])
(setq cte33 '#:contre:#[cte33 ct cta ma dma])
(setq cte32 '#:contre:#[cte32 ct cta p op])
(setq cte31 '#:contre:#[cte31 ct ctr to pt])
(setq cte30 '#:contre:#[cte30 ct ctr b tb])
(setq cte29 '#:contre:#[cte29 sf sfr to pt])
(setq cte28 '#:contre:#[cte28 sf sfr cl acl])
(setq cte27 '#:contre:#[cte27 sf sfr ct act])
(setq cte26 '#:contre:#[cte26 sf sfr f af])
(setq cte25 '#:contre:#[cte25 sf sfa ma dma])
(setq cte24 '#:contre:#[cte24 sf sfa p op])
(setq cte23 '#:contre:#[cte23 f fr to pt])
(setq cte22 '#:contre:#[cte22 f fr cl acl])
(setq cte21 '#:contre:#[cte21 f fr ct act])
(setq cte20 '#:contre:#[cte20 f fr b tb])
(setq cte19 '#:contre:#[cte19 f fa sf rsf])
```

```
(setq cte18 '#:contre:#[cte18 f fa sf asf])
(setq cte17 '#:contre:#[cte17 f fa ma dma])
(setq cte16 '#:contre:#[cte16 f fa p op])
(setq cte15 '#:contre:#[cte15 ma mad to pt])
(setq cte14 '#:contre:#[cte14 ma mad cl acl])
(setq cte13 '#:contre:#[cte13 ma mad ct act])
(setq cte12 '#:contre:#[cte12 ma mad sf asf])
(setq cte11 '#:contre:#[cte11 ma mad f af])
(setq cte10 '#:contre:#[cte10 ma mad b tb])
(setq cte9 '#:contre:#[cte9 ma mad p fp])
(setq cte8 '#:contre:#[cte8 p pf ma dma])
(setq cte7 '#:contre:#[cte7 p po to pt])
(setq cte6 '#:contre:#[cte6 p po cl acl])
(setq cte5 '#:contre:#[cte5 p po ct act])
(setq cte4 '#:contre:#[cte4 p po sf asf])
(setq cte3 '#:contre:#[cte3 p po f af])
(setq cte2 '#:contre:#[cte2 p po b tb])
(setq cte1 '#:contre:#[cte1 p po b ab])
```

ANNEXE A3.

FICHIER COCE DU TOUR : REGROUPEMENT.

```
(SetQ e1  'sfa-ba-pf-fa-mas-ctr-tr-clr)
(SetQ e2  'sfa-ba-pf-fa-mas-ctr-tr-cla)
(SetQ e3  'sfa-ba-pf-fa-mas-cta-tr-cla)
(SetQ e4  'sfa-ba-pf-fa-mas-cta-tp-cla)
(SetQ e5  'sfa-ba-pf-fr-mas-ctr-tr-clr)
(SetQ e6  'sfa-ba-pf-fr-mas-ctr-tr-cla)
(SetQ e7  'sfa-ba-pf-fr-mas-cta-tr-cla)
(SetQ e8  'sfa-ba-pf-fr-mas-cta-tp-cla)
(SetQ e9  'sfa-bt-pf-fa-mas-cta-tr-cla)
(SetQ e10 'sfa-bt-pf-fa-mas-cta-tp-cla)
(SetQ e11 'sfr-ba-po-fr-mas-ctr-tr-clr)
(SetQ e12 'sfr-ba-po-fr-mad-ctr-tr-clr)
(SetQ e13 'sfr-ba-pf-fr-mas-ctr-tr-clr)
(SetQ e14 'sfr-ba-pf-fr-mas-ctr-tr-cla)
(SetQ e15 'sfr-ba-pf-fr-mas-cta-tr-cla)
(SetQ e16 'sfr-ba-pf-fr-mas-cta-tp-cla)

(plist 'env '(nom tour.reg nouveau () sauve t objet tour-groupe
l-objs (tour-groupe) l-ctes () l-rgts () l-auts ()))

(setq nb-ctes 256)

(setq tour-groupe '#:objet:#[tour-groupe (rsf asf ab tb op fp af rf sma dma rct act
rt pt rcl acl) (e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 e13 e14 e15 e16)
(((() () () ()  e13 e14 e15 e16 () ()  e11 e12 e13 e14 e15 e16)
 (() () () ()  e5  ()  ()  ()  () ()  ()  ()  e5  ()  ()  () )
 (e1 e2 e3 e4  e5  e6  e7  e8  e3 e4  ()  ()  e13 e14 e15 e16)
 (() () () e10 ()  ()  ()  ()  () e10 ()  ()  ()  ()  ()  () )
```

```
((() () () () () () () () () () e11 e12 e11 () () () )
(e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e13 () e13 e14 e15 e16)
(e1 () () () e1 () () () () () () () () () () () () )
(e5 e6 e7 e8 e5 e6 e7 e8 () () e11 e12 e13 e14 e15 e16)
(e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e11 e13 e14 e15 e16)
((() () () () () () () () () () e12 e12 () () () () )
(e1 e2 e2 () e5 e6 e6 () () () e11 e12 e13 e14 e14 () )
((() e3 e3 () () () () () () () () () () () () () )
(e1 e2 e3 e3 e5 e6 e7 e7 e9 e9 e11 e12 e13 e14 e15 e15)
((() () e4 e4 () () () () () () () () () () () () )
(e1 e1 () () e5 e5 () () () () e11 e12 e13 e13 () () )
(e2 e2 () () () () () () () () () () () () () () ))
16 16])
```

ANNEXE B.

DÉMARCHE CASPAIM2 APPLIQUÉE À UN EXEMPLE.

I. PRÉSENTATION DE L'EXEMPLE.

A titre d'illustration, considérons l'exemple de la cellule présentée FIGURE B.1. Elle se compose de :

- un tampon d'entrée E,
- un tampon de sortie S,
- un robot manipulateur R1, chargé de l'alimentation du tour T,
- un robot manipulateur R2, assurant les entrées/sorties de pièces ainsi que le chargement/déchargement du centre d'usinage F,
- un convoyeur, sur lequel sept butées définissent trois postes de chargement et déchargement de pièces (P5, P6, P7), et quatre files d'attente (P1, P2, P3, P4),
- un tour T,
- un centre d'usinage F, muni de deux tables équivalentes permettant d'effectuer les entrées/sorties de pièces.

Plusieurs types de pièces peuvent être usinées :pièces à tourner, pièces à fraiser, pièces à tourner et fraiser, pièces à fraiser et à tourner.

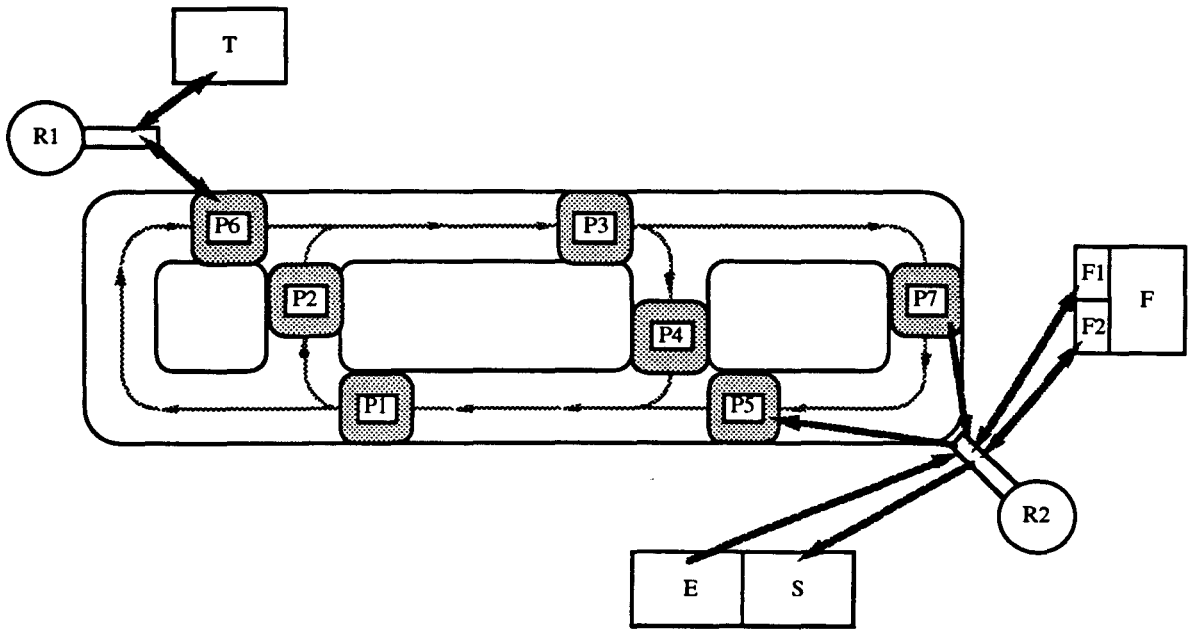


FIGURE B.1. : EXEMPLE D'ATELIER.

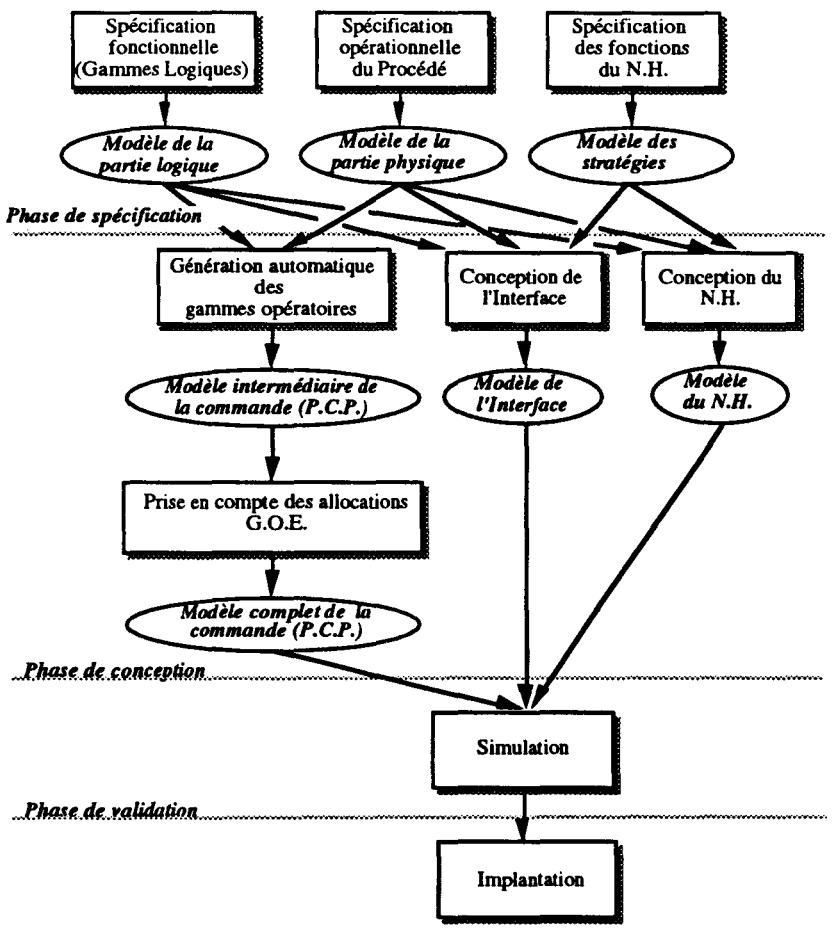


FIGURE B.2 : DÉMARCHE CASPAIM2.

II. LA DÉMARCHE CASPAIM2.

La démarche, qui est synthétisée sur la FIGURE B.2, repose sur une décomposition en neuf phases. Nous allons nous intéresser au fonctionnement de ces différentes phases.

II.1. SPÉCIFICATION FONCTIONNELLE DES OPÉRATIONS.

A ce niveau, nous définissons les *opérations caractéristiques* et les *gammes logiques*. Une opération est dite caractéristique si elle n'a pas comme objectif un transfert et/ou un stockage uniquement. Une gamme logique d'un produit décrit le séquencement des opérations caractéristiques définissant ainsi le processus de fabrication qui permet d'obtenir un produit fini à partir de son état initial.

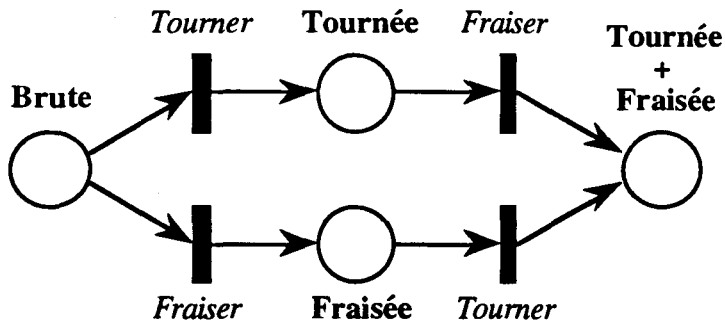


FIGURE B.3 : GAMME LOGIQUE T-F-/F-T-.

Selon la complexité du système, le formalisme retenu pour la description des gammes logiques, sera un RdPC, ou un RdPSD, ou un RdPO. La flexibilité des opérations transformationnelles, est prise en compte à ce niveau de la description [CRUETTE, 91B]. La FIGURE B.3 donne la gamme logique d'une pièce qui doit être tournée et fraisée (t-f-) ou fraisée et tournée (f-t-) : flexibilité des opérations.

II.2. SPÉCIFICATION OPÉRATIONNELLE DES MOYENS DE PRODUCTION.

Cette modélisation s'effectue en deux temps. D'une part, le procédé est décrit par une arborescence d'objets (FIGURE B.4), reliés par une *relation de composition*. Sur cette arborescence, seuls les *lieux caractéristiques* sont représentés. D'autre part, cette représentation est complétée par une description des *relations d'accessibilité* (FIGURE B.5) [CRUETTE, 91B].

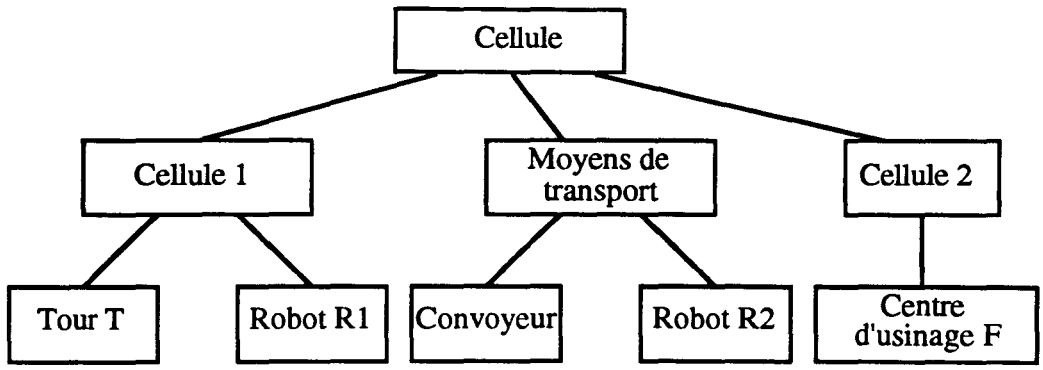


FIGURE B.4 : DESCRIPTION ARBORESCENTE DU PROCÉDÉ.

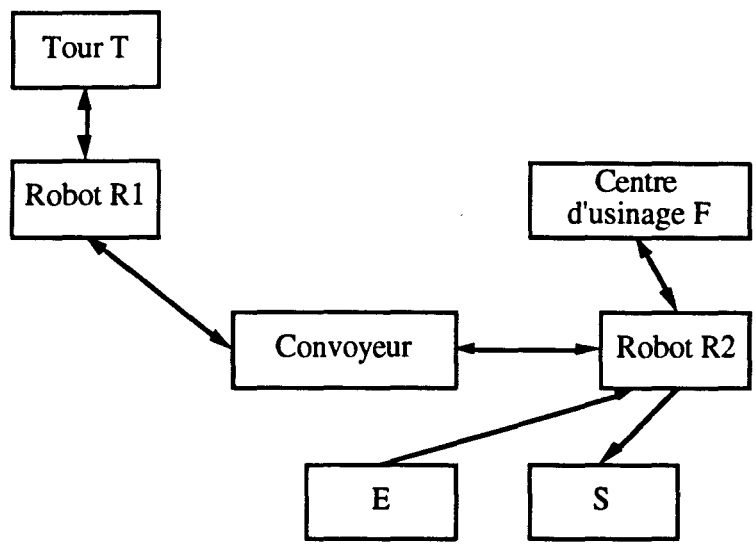


FIGURE B.5 : DESCRIPTION DES RELATIONS D'ACCESSIBILITÉ.

II.3. SPÉCIFICATION DU NIVEAU HIÉRARCHIQUE.

Une spécification des niveaux de décisions (pilotage temps réel, ordonnancement, planification) qui devront être pris en compte, est élaborée lors de cette phase. Cette dernière est en cours de développement. Quelques travaux, ont été réalisés dans ce sens [HAMMADI, 91].

II.4. GÉNÉRATION DES GAMMES OPÉRATOIRES.

Le modèle des *gammes opératoires* est obtenu par fusion des modèles logiques (gammes logiques) et physique (modèle du procédé) [AMAR, 92]. Ce modèle décrit ainsi le

séquencement des différentes transformations fonctionnelles ou positionnelles d'un produit. La *flexibilité d'affectation* apparaît à ce niveau de la modélisation. Elle met en évidence la flexibilité opératoire [AUSFELDER, 94].

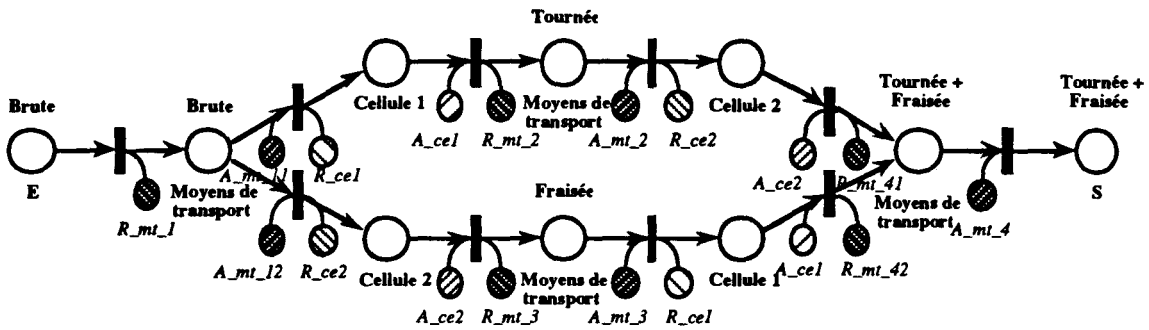


FIGURE B.6 : GAMME OPÉRATOIRE T-F-/F-T-.

La FIGURE B.6 représente la gamme opératoire des pièces t-f-/f-t-. Sur cette gamme, les lieux opératoires et les requêtes/accusés de réception (communication avec les modèles des ressource), sont mis en évidence.

II.5. EXTENSION DES GAMMES OPÉRATOIRES

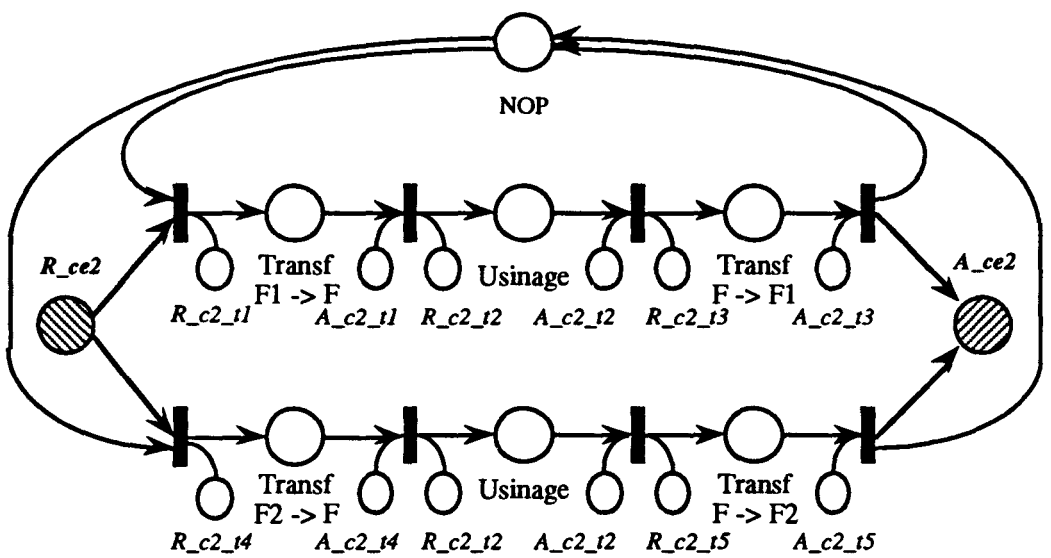


FIGURE B.7 : GAMME OPÉRATOIRE ÉTENDUE DE LA PLACE CELLULE2.

Toutes les ressources de production possèdent nécessairement une capacité limitée. L'accès à un lieu, suite à une requête, nécessite une allocation préalable de ce lieu (limitation

de capacité). Il est ainsi possible de prendre en compte la disponibilité des ressources nécessaire à la réalisation d'une opération.

La prise en compte des protocoles d'allocation et des requêtes d'accès, par les GL, aboutit au modèle gammes opératoires étendues (GOE) (FIGURE B.7).

II.6. CONCEPTION DE L'INTERFACE.

Dans cette phase, les différents constituants de l'interface sont élaborés. Il s'agit notamment de construire les modèles de comportements, les graphes de commande et le niveau hiérarchique local des ressources [HEBRARD, 91A] (FIGURE B.8). C'est à ce niveau du cycle, qu'apparaît la *flexibilité de routage* des objets, par la sélection d'un chemin optimal.

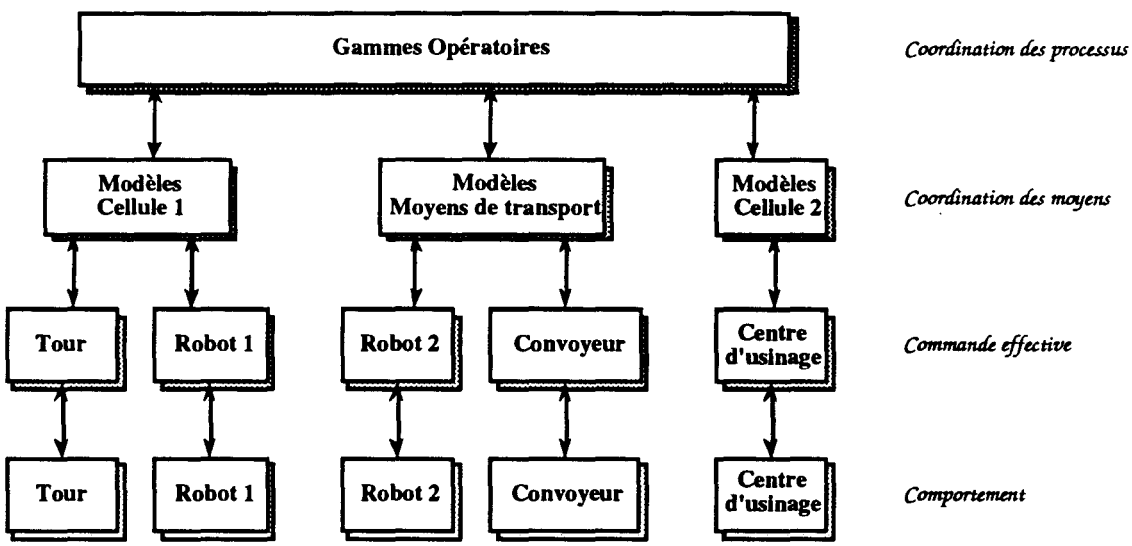


FIGURE B.8 : CONNEXIONS DES DIFFÉRENTS MODÈLES.

II.7. CONCEPTION DU NIVEAU HIÉRARCHIQUE

Il s'agit d'élaborer les différents modèles traduisant les objectifs stratégiques au niveau de la planification, de l'ordonnancement et du pilotage temps réel [HAMMADI, 91], [TAWEGOUM, 92].

L'introduction d'un système de gestion de mode de marche a été introduit par S. BOIS [92]. Ces travaux sont en cours et ont été repris dans [KERMAD, 93A ET 93B].

II.8. SIMULATION.

Lors de cette phase, une validation (quantitative et qualitative) globale du système, est élaborée [AUSFELDER, 92]. A ce titre, des travaux sont actuellement menés sur l'évaluation de performances, par méthodes analytiques approchées basées sur les RdP Stochastiques ou les Réseaux à Files d'Attente [OHL, 93]. Cette approche rend possible une évaluation parallèle à la phase de conception.

II.9. IMPLANTATION.

Il s'agit d'implanter les différents modèles sur une architecture de contrôle/commande (API, mini-micro, ...), choisie ou existante. Cette phase, en cours de développement, doit conserver les caractères modulaire, structurés et hiérarchisés des modèles de conception.

ANNEXE C.

PROGRAMME ESTEREL DU SAS.

```
%
% This is an AGEL Reactive Component
%
% Author: samir
% Created on system: sun4
% Creation date : Wed Mar  3 93 11:38:07
% AGEL version : 1.0
%
% Description:
%
% module generique d'action : debut
%
module Actioning :
input FAction, isDone, isNotDone, Top;
output isFDone, isNotFDone, FActionTo, Action;
constant DeadLine : integer;

 signal ActionTo in
  loop
   await FAction;
   trap ActionInCome in
```

```
      [copymodule InterfaceAction; exit ActionInCome]
      ||
      copymodule TO;
     end trap % ActionInCome
   end loop
 end signal
end module
% module generique d'action : fin
%
% module generique d'interfacage de l'action avec le niveau superieur : debut
%
module InterfaceAction :
input isDone, isNotDone, ActionTo;
output isFDone, isNotFDone, FActionTo, Action;

 emit Action;
 trap WaitingAction in
  [await isDone; emit isFDone; exit WaitingAction]
  ||
  [await isNotDone; emit isNotFDone; exit WaitingAction]
  ||
  [await ActionTo; emit FActionTo; exit WaitingAction]
 end trap % WaitingAction
end module
% module generique d'interfacage de l'action avec le niveau superieur : fin
%
% module generique de Time Out : debut
%
module TO :
input isDone, isNotDone, Top;
output ActionTo;
constant DeadLine : integer;

 do
  await DeadLine Top;
  emit ActionTo;
 watching [isDone or isNotDone];
```

```
end module
% module generique de Time Out : fin
%
% module principal : debut
%
module SAS :
input  FClose,FOpen, Top;
output isFOpen, isNotFOpen, FOpenTo,
       isFClose, isNotFClose, FCloseTo,
       FaultSensor, Intrus,
       Close, Open;
sensor Lo: boolean, Hi1: boolean, Hi2: boolean, Int: boolean;

 signal SLo, TLo, SHi, THi, SInt in
  [copymodule ControlSAS [constant 3/DC, 3/DO]
   ||
   copymodule DetectFaultSensor
   ||
   copymodule Intrusion
   ||
   copymodule Acquisition]
 end signal
end module
% module principal : fin
%
% module de commande du SAS : debut
%
module ControlSAS :
input FClose,FOpen, SLo,TLo, SHi, THi, Top;
output isFOpen, isNotFOpen, FOpenTo,
       isFClose, isNotFClose, FCloseTo,
       Close, Open;
constant DC : integer, DO : integer;

 copymodule Actioning   [signal FClose/FAction,
                         SLo/isDone,
                         THi/isNotDone,
```

```
                    FCloseTo/FActionTo,
                    isFClose/isFDone,
                    isNotFClose/isNotFDone,
                    Close/Action;
              constant DC/DeadLine];
||
copymodule Actioning   [signal FOpen/FAction,
                    SHi/isDone,
                    TLo/isNotDone,
                    FOpenTo/FActionTo,
                    isFOpen/isFDone,
                    isNotFOpen/isNotFDone,
                    Open/Action;
              constant DO/DeadLine];
end module
% module de commande du SAS : fin
%
% module de conversion des capteurs en signaux : debut
%
module Acquisition :
sensor Hi1: boolean, Hi2: boolean, Lo: boolean;
output SHi, THi, SLo, TLo;

 var B : boolean in
  every tick do
   B:= ?Hi1 and ?Hi2;
   if B then emit SHi; end if;
   B:= ?Hi1 or  ?Hi2;
   if B then emit THi; end if;
   B:= ?Lo;
   if B then emit SLo; end if;
   if B then emit TLo; end if;
  end every
 end var
end module
% module de conversion des capteurs en signaux : fin
%
```

```
% module de detection d'incompatibilite des capteurs de commande : debut
%
module DetectFaultSensor :
input SHi, THi, SLo, TLo;
output FaultSensor;

 every tick do
  present [THi and SLo or
           TLo and SHi or
           THi and not SHi or
           TLo and not SLo]
  then emit FaultSensor; end present
 end every
end module
% module de detection d'incompatibilite des capteurs de commande : fin
%
% module de detection d'intrus : debut
%
module Intrusion :
sensor Int: boolean;
output Intrus;

 every tick do
  if ?Int then emit Intrus end if
 end every
end module
% module de detection d'intrus : fin
```

ANNEXE D.

GLOSSAIRE.

Actionneur :

est un composant matériel qui permet de *transformer les ordres* de l'opérateur en actions physiques.
Terme anglais : Actuator.

Capteur :

est un composant matériel qui permet *d'évaluer* par mesure une grandeur physique.
Terme anglais : Sensor.

Contrôle de commande : [ELKHATTABI, 92B]

sa fonction est de *contrôler l'execution des actions* par le procédé et de *détecter* les défaillances de manière *instantanée*. La *propagation des erreurs* est évitée grâce à l'association modulaire par composant indépendant commande/contrôle de commande.
Terme anglais : Control Command.

Défaillance : [CHAPOUILLE, 68]

est la fin (cessation) de l'*aptitude* d'un dispositif à accomplir sa *fonction requise* (nominale). Elle peut être de plusieurs natures :

- La **défaillance progressive**, est une défaillance *prévisible* par un examen antérieur des caractéristiques de l'élément défaillant,

- La **défaillance soudaine**, est une défaillance *imprévisible* par un examen antérieur des caractéristiques de l'élément défaillant,
- La **défaillance partielle**, est une défaillance résultant de déviations d'une ou des caractéristiques au-delà des limites spécifiées mais telles qu'elles n'entraînent *pas une disparition complète* de la fonction requise,
- La **défaillance complète**, est une défaillance résultant de déviations d'une ou des caractéristiques, telles qu'elles entraînent une *disparition* de la fonction requise,
- La **défaillance catalectique** qui est à la fois *soudaine* et *complète*,
- La **défaillance par dégradation** qui est à la fois *progressive* et *partielle*.
- La **défaillance d'exploitation** est causée par le *non-respect* des consignes d'exploitation ou le *dépassement* des limites d'exploitation du composant considéré.

Terme anglais : Failure.

Disponibilité : [VILLEMEUR, 88]

Aptitude d'une entité à être en état d'accomplir une *fonction requise* dans des *conditions données* et à un *instant donné*.

Terme anglais : Availability.

Disponibilité (mesure de la) : [VILLEMEUR, 88]

Probabilité qu'une entité soit en état d'accomplir une fonction requise dans des conditions données et à un instant donné.

Elle est notée A(t).

Terme anglais : Availability.

Erreur :

C'est la *manifestation* d'une faute. Une erreur peut présenter deux états :

- *latente* : elle existe mais elle n'est pas manifeste,
- *effective* : l'erreur est activée.

Terme anglais : Error.

Evénement :

c'est la *cause* d'un changement d'état dans un système : possibilité d'évolution des SED.

Terme anglais : Event.

Faute : [AVIZIENIS, 78]

défaut, anomalie. Deux types de fautes :

- *physiques* : perturbations internes ou externes au système,
- *humaines* : de conception ou d'exploitation (interaction).

Terme anglais : Fault.

Faute->Erreur->Défaillance : [LAPRIE, 85]

Une *erreur* est la partie de l'état d'un système qui est *susceptible* de provoquer une *défaillance*. La *cause phénoménologique* d'une *erreur* est une *faute*. Lorsqu'une *faute* survient ou et commise, il y a *création* d'une *erreur latente* qui deviendra *effective* lorsqu'elle sera activée; lorsque l'erreur affecte le *service délivré*, une *défaillance* survient.

Fiabilité :[CEI, 85; AFNOR, 77]

est l'*aptitude* d'un dispositif à accomplir une *fonction requise*, dans des *conditions données*, pendant une *durée donnée*.

Terme anglais : Reliability.

Fiabilité (mesure de la) : [VILLEMEUR, 88]

Probabilité qu'une entité accomplisse une fonction requise dans des conditions données, pendant une durée donnée.

On suppose que l'entité est en état d'accomplir la fonction requise au début de l'intervalle de temps donnée. La fiabilité est noté ***R(t)***.

Terme anglais : Reliability.

Filtres de commande : [ELKHATTABI, 92A]

est un *modèle fonctionnel* associé à la commande du procédé et vise à assurer une meilleure disponibilité du système et de sa commande. Il permet à la fin :

- d'assurer la *compatibilité* des commandes avec l'état réel du procédé,
- et de prédire le *comportement* du procédé au niveau des composants élémentaires indépendants.

Terme anglais : Command Filters.

Gamme logique : [AMAR, 92], [CRUETTE, 91A]

la gamme logique d'un produit, décrit les *séquencements* possibles(contraintes d'ordre) des *opérations élémentaires* et "*caractéristiques*" définissant ainsi le processus

de fabrication qui permet d'obtenir le produit fini à partir de son état brut. Une opération est dite caractéristique si elle n'a pas comme objectif un transfert et/ou un stockage uniquement.

Terme anglais : Logical sequence.

Gamme opératoire : [CRUETTE, 91A]

la gamme opératoire d'un produit décrit les *séquencements* possibles des différentes *transformations, fonctionnelles* et *positionnelles*, faisant ainsi apparaître la *succession* des lieux physiques sur lesquels il transite.

Terme anglais : Operative sequence.

Maintenabilité : [VILLEMEUR, 88]

Aptitude d'une entité à être *maintenue ou rétablie* dans un état dans lequel elle peut accomplir une *fonction requise* lorsque la maintenance est accomplie dans des *conditions données* avec des procédures et des moyens *prescrits*. (notion duale de la fiabilité).

Terme anglais : Maintainability (performance).

Maintenabilité (mesure de la) : [VILLEMEUR, 88]

Pour une entité donnée, *probabilité* qu'une maintenance accomplie dans des conditions données, avec des procédures et des moyens prescrits, soit achevée au temps t sachant que l'entité est défaillante au temps $t=0$. Elle est notée $M(t)$.

Terme anglais : Maintainability.

Mode de fonctionnement normal :

fonctionnement non-transitoire du système correspondant à la *satisfaction* des spécifications fixées dans les conditions pévues de production.

Terme anglais : Normal mode.

Mode de fonctionnement dégradé :

fonctionnement non-transitoire du système issu de *la dégradation* des conditions de production qui ne répond pas aux exigences souhaitées.

Terme anglais : degraded mode.

Partie commande :

Son rôle est d'assurer la *coordination* et le *séquencement* des commandes applicables à la partie opérative, en tenant compte des décisions issues des différents niveaux de la

gestion de production.
Terme anglais : Control Unit.

Partie opérative :

comporte toutes les *ressources physiques* de production commandables.
Terme anglais : Operative Unit.

Procédé :

est constitué de deux parties :

- Opérative (consommable) : un ensemble d'*éléments physiques* : formé des machines et moyens de l'unité de production,
- Passive (non-consommable) : un ensemble d'*éléments consommables* : formé des outils et produits manufacturés.

Terme anglais : Process.

Redondance explicite : [LARDNER]

c'est la réalisation de la même opération par *deux méthodes différentes* et de préférence par *deux organes différents.*
Terme anglais : Explicit redundance.

Redondance implicite : [LAPRIE, 85]

est due à une inconsistence de la conception. Elle doit être éliminée.
Terme anglais : Implicit redundance.

Redondance matérielle : [LAPRIE, 85]

consiste à multiplier les mêmes organes physiques dans le but d'*accroître* la disponibilité matérielle en cas de défaillance.
Terme anglais : Material redundance.

Ressource de production: [AMAR, 92]

est un objet matériel qui *réalise les ordres élémentaires* de production établis par la partie commande afin de transformer, transporter ou stocker un produit. Une ressource est dite complexe si elle comporte soit, plusieurs (>1) lieux caractéristiques ou soit, au moins, un lieu non caractéristique. Elle est dite élémentaire, si elle comporte un et un seul lieu caractéristique.
Terme anglais : production resource.

Sécurité : [VILLEMEUR, 88]

Aptitude d'une entité à *éviter* de faire apparaître, dans des *conditions données*, des *événements critiques* ou *catastrophiques*.

Terme anglais : Safety.

Sécurité (mesure de la) : [VILLEMEUR, 88]

Probabilité qu'une entité évite de faire apparaître, dans des conditions données, des événements critiques ou catastrophiques.

Terme anglais : Safety.

Signal :

c'est un *événement élémentaire* fugitif (non mémorisé). Un signal peut porter une valeur.

Terme anglais : Signal.

Sûreté de fonctionnement : [LAPRIE, 85; CARTER, 82]

est la *crédibilité* et la *qualité* du service que le système délivre, de telle sorte que les utilisateurs puissent lui accorder (au système) une *confiance justifiée*.

Au sens large, la Sûreté de fonctionnement est considérée comme la *Science des défaillances* et des pannes [VILLEMEUR, 88]

Terme anglais : Dependability.

Système de supervision : [MILLOT, 88]

sa fonction est d'optimiser la production, le fonctionnement et la sécurité des systèmes automatisés, dans un contexte de production spécifique. En effet, il intégre plusieurs sous systèmes :

- la *surveillance*,
- la *gestion des modes de marches*,
- l'interface homme-machine.

Terme anglais : Supervision system.

Système de surveillance :

permet de rendre le système automatisé *sûr de fonctionnement* par la détection des défaillances, le diagnostic de leur origine et leur traitement.

Terme anglais : Monitoring system.

Reprise ou Recouvrement :

c'est l'étape qui permet la *mise en œuvre* les décisions afin de remettre un système défaillant en fonctionnement normal.

Terme anglais : Recovery.

Détection: [BRUNET, 90]

c'est l'opération qui permet de *décider* si le système est ou n'est pas en état structurel normal. C'est une opération logique dont la réponse doit être binaire : oui ou non.

Terme anglais : Detection.

Etat :

permet de caractériser la "situation" d'un système. Les états doivent être observables et différentiables [COMBACAU, 91].

Etat interdit : à partir duquel certaines lois de fonctionnement correct du composant risquent d'être violées [COMBACAU, 91].

Etat utilisable : est un état observable non interdit. Les états utilisables définissent la zone de fonctionnement normal du procédé [COMBACAU, 91].

Etat absolu : caractéristique d'un système, abstraction faite de tout contexte de production ou de fonctionnement.

Etat relatif : état d'un système dans un contexte de production ou de fonctionnement particulier.

Localisation-Diagnostic: [BRUNET, 90]

La localisation *attribue* le défaut détecté à l'organe défaillant (capteur, ressource, commande). Le diagnostic est une opération de classification qui caractérise le défaut par type et degré de sévérité. Elle est adaptée aux défauts et pannes d'usure. Elle implique une estimation de la cause, la quantification du degré de confiance du diagnostic émis.

Le diagnostic est le bloc servant à l'*identification* et la *localisation* d'anomalies détectées [SAHRAOUI, 87].

Terme anglais : Diagnosis.

RÉFÉRENCES BIBLIOGRAPHIQUES.

[AFNOR, 77] NORME AFNOR.
"Statistique et Qualité, Introduction à la fiabilité", X NF, 06-501, Novembre, 1977.

[ALANCHE, 86] P. ALANCHE, P. LHOSTE, G. MOREL, M. ROESCH, M. SALIM ET P. SALVI.
"Application de la modélisation de la Partie Opérative à la structuration de la commande", Journées AFCET 86, 1-14, Montpellier, 1986.

[AMAR, 90] S. AMAR, E. CASTELAIN ET J.C. GENTINA.
"Modélisation des moyens de production par langages orientés objet en vue de la conception de la commande d'un système de production flexible", Actes du colloque CIM'90, Bordeaux, 1990.

[AMAR, 92] S. AMAR, E. CRAYE ET J.C. GENTINA.
"A method of hierarchical specification and prototyping of FMS", IEEE International Workshop on Emerging Technologies and Factory Automation, pp. 44-49, Melbourne, Août 1992.

[ANDRÉ, 91] C. ANDRÉ ET L. FANCELLI.
"Etude d'une réalisation mixte (Asynchrone/Synchrone) d'un système temps-réel.", APII, Vol. 25, N°4, pp 109-140, 1991.

[ANDRÉ, 92] C. ANDRÉ.
"Les approches Synchrones et les Systèmes Automatisés de Production.", I3S, Rapport de Recherche N° 92-43, Nice, 1992.

[ANDRÉ, 93] C. ANDRÉ, S. ELKHATTABI, M.A. PERALDI ET J.C. GENTINA.
"Esterel programming in FMS.", IEEE/SMC 1993, International Conference on Systems Man and Cybernitics, Le Touquet, Octobre 1993 (article accepté).

[ARLAT, 88] J. ARLAT.
"Méthodes et outils pour l'évaluation de la sûreté de fonctionnement des systèmes informatiques tolérant les fautes", TSI, Vol 7, N° 4, 1988.

[AUBRY, 91] J.F. AUBRY ET C. ZANNE.
"Intégration de la sûreté de fonctionnement dans la conception des systèmes de commande numérique des processus électromagnétiques.", APII, Vol. 25, N°4, 1991.

[AUSFELDER, 92] C. AUSFELDER, E. CASTELAIN ET J.C. GENTINA.
"An Object Oriented Simulation Tool to Validate the Dynamic Behaviour of FMS", European Simulation Multiconference, York, UK, Juin 1992.

[AUSFELDER, 94] C. AUSFELDER, E. CASTELAIN ET J.C. GENTINA.
"A Method for Hierarchical Modeling of the Command of FMS", IEEE, Transactions SMC, Vol. 24, N°2, Mars 1994 (article accepté).

[AVIZIENIS, 78] A. AVIZIENIS.
"Fault-tolerance, the Survival Attribute of Digital Systems", Proceedings of the IEEE, 66(10), 1109-1125, Octobre 1978.

[AVIZIENIS, 79] A. AVIZIENIS.
"Towards a discipline of reliable computing", IFIP working conf. on reliable computing and fault-tol-lerance in the 1980's, Londres, 1979.

[BAKO, 90] B. BAKO, R. VALETTE ET M. COURVOISIER.
"A controlled rule-based interpreter : an application to F.M.S. simulation", A.I., Simulation an planning autonome in high autonomy systems, Tucson, Arizona, 26-26 Mars 1990.

[BASTIDE, 92] R. BASTIDE.
"Objets coopératifs : un formalisme pour la modélisation des systèmes concurrents", Thèse de Doctorat de l'Université de Paul Sabatier, Toulouse, 6 Février 1992.

[BAYART, 91] M. BAYART ET M. STAROSWIECKI.
"Functional analysis of smart actuators", 3rd International Congress COMADEM'91, Southampton, UK, 1991.

[BERRY, 85] G. BERRY, P. COURONNE ET G. GONTHIER.
"ESTEREL v2.2 system manuals", Collection de rapports techniques ENSMP/INRIA, 1985.

[BERRY, 87] G. BERRY, P. COURONNE ET G. GONTHIER.
"Programmation synchrone des systèmes réactifs : le langage ESTEREL", TSI, Vol. 6, n° 4, 305-316, 1987.

[BOIS, 92] S. BOIS.
"Intégration de la gestion des modes de marche dans le pilotage d'un système automatisé de production", Thèse de Docteur de l'Université de Lille I, Nov. 1992.

[BONI, 84] BUREAU D'ORIENTATION DE LA NORMALISATION EN INFORMATIQUE
"SCEPTRE : proposition de noyau normalisé pour les exécutifs temps-réel", TSI, Vol. 3, 1984.

[BOOCH, 88] G. BOOCH.
"Ingénierie du logiciel en ADA", Inter Editions, IIA, 1988.

[BOOCH, 91] G. BOOCH.
"Object Oriented design with applications", Benjamin/cumming Publishing Company Inc, 1991.

[BOUCHOUCH, 92] A. BOUCHOUCH.
"Evaluation de performances d'un réseau de files d'attente à capacité limitée", Thèse de Doctorat d'Université, Institut National Polytechnique de Grenoble, Grenoble, 6 Février 1992.

[BOUDOL, 85] G. BOUDOL.
"Notes on algebric calculi of process", Rapport INRIA N° 395, 1985.

[BOUREY, 88] J.P. BOUREY.
"Structuration de la partie procédurale du système commande des cellules flexibles dans l'industrie manufacturière", Thèse de Docteur de l'Université de Lille I, Mars 1988.

[BOUREY, 89] J.P. BOUREY, E. CASTELAIN, J.C. GENTINA ET M. KAPUSTA.
"CASPAIM : A computer aided design of the control system of FMS", IMACS Annals on Computing and Applied Mathematics, 131-135, Paris, 1989.

[BOUREY, 91] J.P. BOUREY ET J.C. GENTINA.
"Conception structurée et modulaire d'architectures de contrôle réparti en production flexible manufacturière.", APII, Vol. 25, N°4, 1991.

[BOUREY, 93] J.P. BOUREY.
"Méthode de conception de la commande des systèmes flexibles de production manufacturièrs", Habilitation à diriger des recherches de l'Université de Lille I, Février 1993.

[BOUSSINOT, 91] F. BOUSSINOT ET R. DE SIMONE.
"The Esterel language", Another Look at Real-Time Programming, Proceedings of the IEEE, Special Issue, Septembre 1991.

[BRAMS, 83A] G.W. BRAMS.
"Réseaux de Petri : Théorie et Pratique. Tomel : Théorie et Analyse", Masson, 1983.

[BRAMS, 83B] G.W. BRAMS.
"Réseaux de Petri : Théorie et Pratique. Tome 2 : Modélisation et Applications", Masson, 1983.

[BRUNET, 90] J. BRUNET, D. JAUNE, M. LABARRÈRE, A. RAULT ET M. VERGÉ;
"Détection et diagnostic de pannes : approche par la modélisation", Ed. Hermès, Paris, 1990.

[CARDELLI, 85] L. CARDELLI.
"SQUEAK, a language for communicating with mice", In AT&T Bell Laboratories Report, Bell Laboratories, Murray Hill, N.J., 1985.

[CARTER, 82] W. C. CARTER.
"A Time for Reflection", Proceedings of the 12th Int. Symp. on Fault-Tolerant Computing, Los Angeles, 41, Juin 1982.

[CASPI, 87] P. CASPI, D. PILAUD, P. HALBWACHS ET J. PLAICE.
"LUSTRE, a declarative language for real-time programming", In Proc. conf. on principles of programming languages, Munich, 1987.

[CASTELAIN, 87] E. CASTELAIN.
"Modélisation et simulation interactive de cellules flexibles dans l'industrie manufacturière", Thèse de Doctorat de l'Université de Lille I, 26 Février 1987.

[CARDOSO, 90] J. CARDOSO.
"Sur les réseaux de Petri avec marquages flous", Thése de Doctorat de l'Université de Paul Sabatier, Toulouse, Octobre 1990.

[CEI, 74] COMMISSION ELECTROTECHNIQUE INTERNATIONALE.
"Liste des termes de base, définitions et mathématiques applicables à la fiabilité", Publication 271, 1974.

[CEI, 85] COMMISSION ELECTROTECHNIQUE INTERNATIONALE.
"Liste des termes de base, définitions et mathématiques applicables à la fiabilité", Publication 271C, 1985.

[CHAILLOUX, 91A] J. CHAILLOUX.
"Programmation synchrone et génie logiciel", La lettre d'Esterel, CISI Ingenierie et Ilog, N° 1, Juillet 1991.

[CHAILLOUX, 91B] J. CHAILLOUX ET P. COURONNÉ.
"AGEL : un atelier de développement de systèmes réactifs synchrones", Génie logiciel et Systèmes experts, N° 25, 1991.

[CHANDRASEKARAN, 89] B. CHANDRASEKARAN, J.W. SMITH ET J. STICKLEN.
"Deep models and their relation to diagnosis", Artificial Intelligence in Medecine, Vol 1, 29-40, 1989.

[CHAPOUILLE, 68] P. CHAPOUILLE ET R. DE PAZZIS.
"Fiabilité des systèmes", Ed. Masson, Paris, 1968.

[COAD, 91] P. COAD ET E. YOURDON.
"Object Oriented Analysis", Yourdon Press Computing Series, Prentice Hall, 1991.

[COAD, 92] P. COAD ET E. YOURDON.
"Object Oriented Design", Yourdon Press Computing Series, Prentice Hall, 1992.

[CODD, 70] E.F. CODD.
"A relational model for Large Shared Data Banks", CACM, 13, n°6, 377-387, 1970.

[COMBACAU, 90] M. COMBACAU ET M. COURVOISIER.
"A hierarchical and modular structure for FMS control and monitoring", First International Conference on AI Simulation and Planning in high autonomy systems, Tucson, Arizona, Mars 1990.

[COMBACAU, 91] M. COMBACAU.
"Commande et surveillance des systèmes a événements discrets complexes : application aux ateliers flexibles", Thèse de Doctorat de l'Université de Paul Sabatier, Toulouse, Décembre 1991.

[CORBEEL, 79] D. CORBEEL.
"Schéma de cablâge et schéma de contrôle. Application à la simulation et à la gestion des processus industriels", Thèse de Doctorat de Spécialité de Lille I, 1979.

[CORBEEL, 80] D. CORBEEL.
"Formal description of processes systems and execption handling", Mini&Micro, Proc pp. 335-339, Budapest, Septembre 1980.

[COSTES, 80] A. COSTES.
"A relational model for Large Shared Data Banks", CACM, 13, n°6, 377-387, 1970.

[CRAYE, 89] E. CRAYE.
"De la modélisation à l'implantation automatisée de la commande hiérarchisée de cellules de production flexibles dans l'industrie manufacturière", Thèse de Docteur de l'Université de Lille I, Janvier 1989.

[CREUSOT, 89] D. CREUSOT, J. PERRAUD ET O. ROUX.
"Le système ELECTRE", Rapport interne LAN-ENSM, N° 89.01, 1989.

[CRUETTE, 91A] D. CRUETTE.
"Méthodologie de conception des systèmes complexes à événements discrets : application à la conception et la validation de la commande des cellules flexibles de production dans l'industrie manufacturière", Thèse de Docteur de l'Université de Lille I, Fév. 1991.

[CRUETTE, 91B] D. CRUETTE, J.P. BOUREY ET J.C. GENTINA.
"Description and validation of logical operationg sequences in the flexible manufacturing system context using object Petri Nets", IMACS-IFAC Symposium, Vol. 1, 567-572, Lille, Mai 1991.

[DEVAPRIYA, 92] D. S. DEVAPRIYA, B. DESCOTES-GENON ET P. LADET.
"Petri net based node structures for distributed problem solving in F.M.S. control", CIMS, Vol 5, N°3, pp. 229-237, Août 1992.

[ELKHATTABI, 91] S. ELKHATTABI.
"Un système de contrôle des commandes des ateliers flexibles en industrie manufacturière", Journées SED, GT2, Paris, Fevrier 1991.

[ELKHATTABI, 92A] S. ELKHATTABI.
"Intégration d'un système surveillance dans la conception des systèmes de commande d'ateliers flexibles", Rapport interne du LAIL, NI/92/1, Lille, Janvier 1992.

[ELKHATTABI, 92B] S. ELKHATTABI, D. CORBEEL ET J.C. GENTINA.
"Integration of dependability in FMS", 7th IFAC Symposium on Information Control Problems in Manufacturing Technology, 249-255, Toronto, Mai 1992.

[ELKHATTABI, 93A] S. ELKHATTABI.
"Esterel programming in FMS : low level monitoring", Journées SED, GT2, Paris, Mai 1993.

[ELKHATTABI, 93B] S. ELKHATTABI.
"COCE : un outil pour la Conception des Objets Commandables Elémentaires", Rapport interne du LAIL, NI/93/1, Lille, Juin 1993.

[FANCELLI, 91] L. FANCELLI.
"Approche mixte (Asynchrone/synchrone) d'un système temps-réel", Thèse de Doctorat, Université de Nice, 1991.

[FARAH, 91] A. FARAH.
"Détection et diagnostic de pannes dans les ateliers de production flexibles de l'industrie manufacturière", Rapport de DEA, Université de Lille 1, 1991.

[FRACHET, 87] J.P. FRACHET.
"Une introduction au génie automatique : faisabilité d'une chaine d'outils de CAO pour la conception et l'exploitation des machines automatiques industrielles", Thèse d'Etat, Université de Nancy I, 1987.

[GENRICH, 87] H.J. GENRICH.
"Predicate/Transition Nets", Lecture notes in computer science, Vol 254, pp. 207-247, Springer verlag, 1987.

[GLORY, 90] A.C. GLORY ET J.L. BERGERAND.
"SAGA : conception de logiciels & preuves de propriétés de l'approche synchrone", Génie Logiciel & Systèmes Experts, N° 18, Mars 1990.

[GONDRAN, 79] M. GONDRAN ET M. MINOUX.
"Graphes et algorithmes", Eyrolles, 1979.

[GONTHIER, 88] G. GONTHIER.
"Sémantique et modèles d'exécution des langages réactifs synchrones; application à Esterel" Thèse d'Informatique de l'Université d'Orsay, 1988.

[HAMMADI, 91] S. HAMMADI.
"Une méthode d'ordonnancement minimisant les temps d'attente et de transit dans les systèmes de production flexibles de type job-shop", Thèse de Docteur de l'Université de Lille I, Décembre 1991.

[HAREL, 85] D. HAREL ET A. PNUELI.
"On the development of reactive systems", Logics and Models of Concurrent Systems, NATO ASI Series, Vol. 13, Springer-Verlag, pp. 477-489, 1985.

[HAREL, 87] D. HAREL.
"Statecharts : a visual formalism for complex systems", Science of Computer Programming, 8, 1987.

[HEBRARD, 92A] A. HEBRARD.
"Etude et réalisation d'un module de pilotage et d'un outil de représentation graphique appliqués à la conception d'un système de production flexible", Thèse de Docteur de l'université de Lille I, Fév. 1992.

[HEBRARD, 92B] A. HEBRARD, J.P. BOUREY ET J.C. GENTINA.
"A tool of Petri net graphs representation in the conception of FMS", 7th IFAC Symposium on Information Control Problems in Manufacturing Technology, 634-639, Toronto, Mai 1992.

[HEIZERLING, 88] T. HEIZERLING.
"Adaptation of the Petri-net of the pregraph designed in CASPAIM project in order to make the prototypage and a preleminary evaluation of

the parallel architecture of the control in the field of CIM", Diplomarbeit TU München/IDN, Lille, 1988.

[HOARE, 78] C.A.R. HOARE.
"Communicating Sequential Process", Com. ACM 21, 8, 1978.

[HUBER, 90] P. HUBER, K. JENSEN ET R.M. SHAPIRO.
"Hierarchies in coloured Petri nets", Lectures Notes in Computer Science, Vol. 483, Springer, pp. 313-341, 1990.

[HUVENOIT, 92] B. HUVENOIT, E. CRAYE ET J.C. GENTINA.
"Elaboration de la commande des cellules de production flexibles en industrie manufacturière", Actes de la Conférence Automatisation Industrielle, Vol I, pp. 6.21-6.25, Montréal, Juin 1992.

[ILOG, 89] ILOG S.A.
"LeLisp de l'INRIA V. 15.22, Le Manuel de Référence ", ILOG, Paris, 1989.

[ILOG, 92A] ILOG S.A.
"AGEL V. 1.0, Workshop Manual", ILOG, Paris, Juin 1992.

[ILOG, 92B] ILOG S.A.
"AIDA V. 1.65, Reference Manual", ILOG, Paris, Novembre 1992.

[JENSEN, 86] K. JENSEN.
"Coloured Petri Nets", Lectures Notes in Computer Science, Springer Verlag, 1986.

[KAPUSTA, 88] M. KAPUSTA.
"Une première étape de conception assistée du modèle de la partie commande de cellules flexibles de production dans l'industrie manufacturière", Thèse de Docteur de l'Université de Lille I, Décembre 1988.

[KAUFMANN, 68] A. KAUFMANN.
"Introduction à la combinatorique", Dunod, 1968.

[KERMAD, 93A] L. KERMAD, C. AUSFELDER, J.P. MAIK, J.C. GENTINA, D. DELFIEU, R. MOISAN ET A.E.K. SAHRAOUI.
"Integration of Operative Modes in the Control of FMS Combining Synchronous and Asynchronous Approach", IEEE/SMC 1993, International Conference on Systems Man and Cybernitics, Le Touquet, Octobre 1993 (article accepté).

[KERMAD, 93B] L. KERMAD, C. AUSFELDER, J.P. BOUREY ET E. CASTELAIN.
"An interactive approach for a functional specification of FMS control", CIMS, Novembre 1993 (article accepté).

[LAI, 91] M. LAI.
"Conception Orientée objet, la méthode HOOD", Dunod informatique, 1991.

[LAPRIE, 85] J.C. LAPRIE.
"Sûreté de fonctionnement des systèmes informatiques et tolérance aux fautes : concepts de base", TSI, 4, n°5, 419-429, 1985.

[LEGUERNIC, 86] P. LE GUERNIC, A. BENVENISTE, P. BOURNAI ET T. GAUTHIER.
"SIGNAL : a data flow oriented language for SIGNAL processing", IEEE-ASSP, 34, n° 2, 362-374, 1986.

[LHOSTE, 91] P. LHOSTE.
"Surveillance des M.S.A.P. : les Atouts de la modélisation de Comportement", journée surveillance du Pôle SED (GT2) du GR Automatique, 7 février 91, Paris.

[LONG, 92] J. LONG, B. DESCOTES-GENON ET P. LADET.
"Hierarchical and intelligent control of flexible manufacturing systems", INCOM'92, 7th IFAC/IFIP/IFOP/IMACS/IPSE symposium on information control problems in manufacturing technology, pp. 243-248, Toronto, May 1992.

[MCDERMOTT, 82] D. MC DERMOTT.
"A temporal logic for reasoning about processes and plans", Cognitive Science, Vol. 6, 101-155, 1982.

[MILLOT, 88] P. MILLOT.
"Supervision des procédés automatisés et ergonomie", Ed. Hermès, Paris, 1988.

[MOREL, 92] G. MOREL.
"Contribution à l'automatisation et à l'ingénierie des systèmes intégrés de production", Mémoire d'habilitation à diriger des recherches, Université de Nancy I, 28 Janvier 1992.

[NIEL, 92] E. NIEL.
"Aspects of working conditions and safety in the use of industrial robots : an overview", in Computer Integrated Manufacturing Systems, Butterorth-Heinemann Ltd Ed., Vol. 5, N°1, 66-77, Fevrier 1992.

[OGUS, 90] A. OGUS.
"Le zéro maintenance, est-ce une contrainte supplémentaire", Achats et Entretien, AFIM, N° 429, Mai 1990.

[OHL, 93] H. OHL, E. CASTELAIN ET J.C. GENTINA.
"State dependant resource allocation in FMS.", IEEE/SMC 1993, International Conference on Systems Man and Cybernitics, Le Touquet, Octobre 1993 (article accepté).

[O'KEEFE, 90] R.A. O'KEEFE.
"The Craft of Prolog", MIT Press, 1990.

[PAGÈS, 80] A. PAGÈS ET M. GONDRAN.
"Fiabilité des systèmes", Ed. Eyrolles, Paris, 1980.

[PALUDETTO, 91] M. PALUDETTO.
"Sur la commande de procédés industriels : une méthodologie basée objets et réseaux de Petri", Thèse de Doctorat de l'Université de Paul Sabatier, Toulouse, Décembre 1991.

[PANETTO, 91] H. PANETTO, P. LHOSTE, G. MOREL ET M. ROESCH.
"SPEX : du génie logiciel pour le génie automatique", Génie logiciel et Systèmes experts, N° 25, 1991.

[PERALDI, 93] M.A. PERALDI.
"Conception et Réalisation de Systèmes Temps-Réel par une Approche Synchrone", Thèse de Doctorat de l'Université de Nice-Sophia Antipolis, Juillet 1993.

[PETERSON, 80] J.L. PETERSON.
"A note on coloured Petri-Nets", Information processing letters, Vol. 11, N°1, pp. 40-43, 1980.

[PHAN, 92] S. PHAN.
"Les langages synchrones et la programmation des systèmes réactifs", Rapport interne de la DER de EDF, Clamart, Avril 1992.

[RIAT, 92] J.C. RIAT.
"Validation par simulation d'architecture de commande d'atelier dans l'industrie de production manufacturière", Thèse de Doctorat de l'Université des Sciences et Technologies de Lille, 12 Novembre 1992.

[ROLLAND, 82] C. ROLLAND ET C. RICHARD.
"REMORA - a methodology for information system design and management", In Proc. of IFIP-CRIS conf., North Holland, 1982.

[ROLLAND, 86] C. ROLLAND.
"REMORA : une methode de conception des systèmes d'information", Genie Logiciel, 4, 36-43, 1986.

[ROY, 62] B. ROY.
"Cheminement et connexité dans les graphes", Thèse de l'Université de Paris, 1962.

[ROY, 89] V. ROY.
"AUTOGRAPH : un outil d'analyse visuelle de Systèmes Concurrents Communicants", Thèse de troisième cycle de l'Université de Nice, 1989.

[SACKS, 86] P.A. SACKS ET AL.
"ESCORT, an expert system for complex operation in real-time", Expert system, Vol. 3, N° 1, Janvier 1986.

[SAHRAOUI, 87] A.E.K. SAHRAOUI.
"Contribution à la surveillance et à la commande d'atelier", Thèse de Doctorat de l'Université de Paul Sabatier, Toulouse, 1987.

[SAHRAOUI, 91] A.E.K. SAHRAOUI ET L. GILHODES.
"Statecharts, temporal logic and Petri nets to specify discrete events controllers: a comparative study on descriptive power", ECC-91, Grenoble, 1991.

[SAHRAOUI, 92] A.E.K. SAHRAOUI.
"Vers une approche globale de surveillance des systèmes à événements discrets", APII, Vol. 26, N° 2, 1992.

[SAKAROVITCH, 84] M. SAKAROVITCH.
"Optimisation combinatoire : Méthodes mathématiques et algorithmiques", HERMANN, 1984.

[SCHOENAUER, 90] G. SCHOENAUER.
"Analyse des huiles de service, maintenance préventive des machines", Achats et Entretien, AFIM, N° 430, Juin 1990.

[SEMBUGAMOORTHY, 86] V. SEMBUGAMOORTHY ET V. CHANDRASEKARAN.
"Functional representation of devices and compilation of diagnosis problem solving systems", in J.L. Kolodner and C.K. Riesbeck, Experience, Memory and Reasonning Ed., Lauwrence Erlbaum Associates, 1986.

[SFALCIN, 89] A. SFALCIN, M. ROESCH, P. LHOSTE ET G. MOREL.
"Fiabilité, Disponibilté et Sécurité des installations intégrant des Filtres de comportements", 2° Colloque annuel du Club FIABEX, I.N. des Télécommunications, Every, Novembre 1989.

[SIBERTIN, 85] C. SIBERTIN-BLANC.
"High level Petri-nets with data structures", 6th European workshop on theory and application of Petri nets", Espoo, Finlande, 1985.

[SIBERTIN, 90] C. SIBERTIN-BLANC.
"L'approche objet pour la surveillance et la mise en œuvre des Ateliers Flexibles", Réunion SED, GT2, Paris, 11 Janvier 1990.

[STAROSWIECKI, 91] M. STAROSWIECKI ET M. BAYART .
"Les fonctionnalités des actionneurs intelligents", MESUCORA'91, Paris, Novembre 1991.

[STERLING, 86] L. STERLING ET E. SHAPIRO.
"The Art of Prolog : Advanced Programming Techniques", MIT Press, 1986.

[TAWEGOUM, 92] R. TAWEGOUM, E. CASTELAIN ET J.C. GENTINA.
"Real time piloting of flexible manufacturing systems", Third International Workshop on PROJECT MANAGMENT AND SCHEDULING, Como, Italy, Juillet 1992.

[TIXADOR, 89] J.M. TIXADOR.
"Une contribution au génie automatique : la spécification exécutable des machines et systèmes automatisés de production", Thèse de Docteur de l'Université de Nancy I, Juillet 1989.

[TOGUYENI, 90] A.K.A TOGUYENI, E. CRAYE ET J.C. GENTINA.
"A method for temporal analysis to perform on-line diagnosis in the context of Flexible Manufacturing System", IECON'90, Vol. 1, 445-450, Pacific Grove-California, Novembre 1990.

[TOGUYENI, 91] A.K.A TOGUYENI, E. CASTELAIN ET E. CRAYE.
"From the treatment of failures to the management of working modes", IMACS-IFAC Symposium, Vol. 1, 595-601, Lille, Mai 1991.

[TOGUYENI, 92] A.K.A TOGUYENI.
"Surveillance et diagnostic en ligne dans les systèmes flexibles de l'industrie manufacturière", Thèse de Docteur de l'Université de Lille I, Novembre 1992.

[VALLOTON, 91] J.J. VALLOTTON.
"Statemate : un outil pour la spécification des systèmes réactifs", Génie logiciel et Systèmes experts, N° 25, 1991.

[VERGAMINI, 87] D. VERGAMINI.
"Vérification de réseaux d'automates finis par équivalence observationnelle : le système AUTO", Thèse de Doctorat de l'Université de Nice, 1987.

[VILLEMEUR, 88] A. VILLEMEUR.
"Surêté de fonctionnement des systèmes industriels", Ed. Eyrolles, Paris, 1988.

[VORGRIG, 86] R. VORGRIG, G. MOREL, D. DEI-SVALDI ET J.P. VAUTRIN.
"Contribution à l'analyse de la sécurité d'un système automatisé par SADT", Electronique Industrielle, N° 114/15, Octobre 1988.

[YOURDON, 89] E. YOURDON.
"Modern structured analysis", Prentice Hall/YOURDON Press, Eng. Cliffs, N.J., 1989.

TABLE DES MATIERES.

CHAPITRE I. POSITIONNEMENT DE LA SURVEILLANCE DES SED.

CHAPITRE II. VERS UNE APPROCHE DE LA SURVEILLANCE.

CHAPITRE III. UN OUTIL POUR LE CONTRÔLE DE COMMANDE.

CHAPITRE IV. LA CONCEPTION DU CONTRÔLE DE COMMANDE.

CHAPITRE V. UN OUTIL POUR LES FILTRES DE COMMANDE

CHAPITRE VI. LA CONCEPTION DES FILTRES DE COMMANDES.

CHAPITRE VII. INTÉGRATION DE LA SURVEILLANCE À CASPAIM.

ANNEXES.

LISTE DES FIGURES

CHAPITRE I. POSITIONNEMENT DE LA SURVEILLANCE DES SED.

CHAPITRE II. VERS UNE APPROCHE DE LA SURVEILLANCE.

CHAPITRE III. UN OUTIL POUR LE CONTRÔLE DE COMMANDE.

CHAPITRE IV. LA CONCEPTION DU CONTRÔLE DE COMMANDE.

CHAPITRE V. UN OUTIL POUR LES FILTRES DE COMMANDE.

CHAPITRE VI. LA CONCEPTION DES FILTRES DE COMMANDE.

CHAPITRE VII. INTÉGRATION DE LA SURVEILLANCE À CASPAIM.

ANNEXES.