Conception de la commande d'un système automatisé de production : apport des graphes et de l'ordonnancement cyclique

Olivier Fournier

UNIVERSITE DE LA REUNION

FACULTE des SCIENCES et TECHNOLOGIE

Année :2002 numéro d'ordre |_|_|_|_|_|

Thèse présentée par

Olivier Fournier

Pour l'obtention du titre de Docteur de l'Université de La REUNION

Discipline : Productique, Automatique et Informatique Industrielle

Conception de la commande d'un système automatisé de production : apport des graphes et de l'ordonnancement cyclique.

Soutenue le mercredi 20 mars 2002

Au LAAS-CNRS - 7, Avenue du Colonel ROCHE - 31077 TOULOUSE.

Composition du jury

Jean-Pierre CHABRIAT	Président du jury,	Professeur à l'Université de La REUNION.
Jean-Louis FERRIER	Examinateur,	Professeur à l'Université d'ANGERS.
Jean-Claude GENTINA	Rapporteur,	Professeur à l'Ecole Centrale de LILLE.
Jean-Daniel LAN SUN LUK	Directeur de thèse,	Professeur à l'Université de La REUNION.
Pierre LOPEZ	Examinateur,	Chargé de Recherche 1 CNRS-LAAS, TOULOUSE.
François ROUBELLAT	Rapporteur,	Directeur de Recherche CNRS-LAAS, TOULOUSE.

Remerciements,

Les travaux présentés dans ce mémoire ont été effectués au sein de l'équipe ACTES (Analyse et Contrôle des Transferts Energétiques et des Systèmes) du Laboratoire de Génie Industriel de l'Université de La REUNION, laboratoire dirigé par le Professeur Patrick HERVE.

Je remercie l'Université de La REUNION et ses Conseils pour les décharges d'enseignement qu'ils ont bien voulues m'accorder pendant ces trois années et sans lesquelles ce travail n'aurait pu être mené à bien. Je remercie également le LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes) de nous avoir accueilli dans ses locaux pour présenter ces travaux.

Toute ma gratitude va au Professeur Jean-Daniel LAN SUN LUK, coordonnateur de l'équipe ACTES, pour avoir dirigé cette Thèse, pour son soutien et pour la confiance qu'il m'a témoignée durant cette période.

Je tiens à remercier vivement le Professeur Jean-Claude GENTINA, Directeur de l'Ecole Centrale de Lille et Monsieur François ROUBELLAT, Directeur de Recherche CNRS–LAAS de Toulouse, pour leurs critiques constructives et pour l'intérêt qu'ils ont porté à mon travail en acceptant d'en être les rapporteurs.

Je suis très honoré que le Professeur Jean-Louis FERRIER, Directeur du Laboratoire d'Ingénierie des Systèmes Automatisés d'ANGERS et que Monsieur Pierre LOPEZ, Chargé de Recherche au CNRS-LAAS de Toulouse, aient accepté d'étudier ce travail et de participer au jury. Je leurs en suis très reconnaissant.

J'exprime mes profonds remerciements au Professeur Jean-Pierre CHABRIAT de l'Université de La REUNION pour son soutien constant et pour avoir accepté la présidence du jury.

Mes remerciements vont également à Monsieur Ouajdi KORBAA, Maître de Conférence à l'Ecole Centrale de Lille et à Pierre LOPEZ pour les nombreux conseils scientifiques qu'ils n'ont cessés de me donner pendant cette dernière année.

Je voudrais enfin remercier tous les membres de l'équipe ACTES, mes enfants et mon épouse, pour leurs différentes contributions à cette expérience enrichissante : que chacun trouve ici, le sentiment qui lui revient.

Thank you, あなたに感謝しなさい , *Gracias,* 너를 감사하십시오 , *Danke,* 谢谢 , *Agradeça-o,* شكراً

Sommaire.

Partie 2 : Optimisation de la productivité des SAP.

Partie 3 : Etude de cas - Impact sur la flexibilité.

Partie 4 : Conclusions et perspectives.

Annexe 1 : Procédures spécifiques à l'analyse des graphes.

Annexe 2 : Procédures spécifiques à l'ordonnancement cyclique.

Introduction générale.

Notre champ d'étude concerne la commande des systèmes à événements discrets réactifs et synchrones. Notre positionnement au sein de la classification CIM (*Computer Integrated Manufacturing*) se situe aux niveaux *0* et *1*, c'est-à-dire aux niveaux des machines automatisées et des ateliers de production. L'objectif que nous nous sommes fixés s'appuie sur la problématique générale du domaine, à savoir, la productivité, la flexibilité et l'agilité. Cette dernière qualité récemment mise à jour requiert des méthodes basées sur la capitalisation et la ré-utilisation du savoir-faire de l'entreprise. Sans négliger l'aspect agile de l'entreprise, nous nous focalisons davantage sur le double objectif de productivité et de flexibilité : *Un système automatisé de production peut-il être à la fois productif et flexible ?* Ces deux notions sont-elles complémentaires ou antinomiques ? C'est La Question[1] à laquelle nous tentons de répondre dans le cadre de ce travail.

A l'issue de l'étude bibliographique du domaine des systèmes automatisés de production (SAP), nous dressons le bilan des préoccupations de la communauté des automaticiens étendue à celle des informaticiens, des mathématiciens et des logiciens. Nous examinons les nombreuses méthodes de spécification et de vérification/validation du modèle de commande des systèmes automatisés, dans des formalismes appropriés à chacune de ces méthodes. Les performances de ces formalismes et de ces méthodes autorisent l'étude approfondie d'un grand nombre de propriétés, mais elles ne sont pas totalement adaptées à notre préoccupation qui impose davantage une représentation macroscopique principalement basée sur le *temps*. Ces formalismes et ces méthodes restent cependant complémentaires et indispensables à notre approche dès lors que nous souhaitons spécifier, vérifier et/ou valider les systèmes étudiés avec plus de précision.

[1] De façon inductive, se pose le problème de l'évaluation de la productivité et la flexibilité.

Il nous faut donc définir un formalisme qui réponde à la fois à notre préoccupation et à une cohabitation possible avec les formalismes existants, en particulier avec le formalisme GRAFCET. Ce formalisme, que nous appelons formalisme *GMT* pour Graphe Marqué et Temporisé, s'inspire en partie du formalisme *Marked Directed Graph* exposé dans [COMMONER F., 71]. Nous accédons ainsi à une plate-forme multi-modèle admettant le formalisme GRAFCET comme formalisme 'pivot'. Une passerelle GMT-GRAFCET autorise des allers et retours entre les deux formalismes et permet de passer d'un modèle GMT à un modèle Grafcet, tout en conservant au second modèle les propriétés de célérité et de vivacité du premier. L'architecture du modèle GMT et les propriétés évoquées ci-dessus sont analysées à l'aide des outils développés dans le cadre de la théorie des graphes et ces outils sont adaptés localement à notre contexte.

Disposant du formalisme et des outils s'y rapportant, nous proposons alors d'optimiser la productivité et la flexibilité d'un SAP. Un système est flexible parce qu'il autorise la fabrication d'un ensemble de produits possédant chacun sa propre gamme de fabrication où se trouve décrit l'enchaînement des tâches qui conduisent à son élaboration. Mais cette flexibilité entraîne généralement une commande à choix multiples qui nous éloigne de l'optimalité. La stratégie que nous retenons consiste à définir pour chaque produit, une commande spécifique de productivité optimale du système pour ce produit : comme l'exprimait en 1957 le mathématicien américain Richard BELLMAN « *toute politique optimale ne peut être formée que de sous-politiques optimales*[2] ». Cette stratégie implique que nous soyons capables de changer la commande du SAP à volonté et dans la limite de sa flexibilité, mais aussi, que nous soyons capables d'optimiser la productivité des commandes dédiées à chaque produit.

La première implication repose techniquement sur l'emploi d'une commande distribuée, autrement dit, sur une implantation géographique de la commande localisée au SAP ou à la partie du SAP étudiée. La seconde implication revient à définir un agencement des tâches qui minimise le nombre d'encours par cycle et le temps de cycle du SAP, tout en respectant les contraintes de précédence et d'affectation des ressources spécifiées par la gamme du produit. L'agencement des tâches d'un SAP est une problématique très proche de celle de l'ordonnancement cyclique des ateliers de production, mais ces méthodes d'ordonnancement sont actuellement limitées aux traitements de gammes de fabrication exclusivement linéaires. Des travaux de recherche sur le traitement des gammes non-linéaires sont en cours au LAIL (Laboratoire d'Automatique et d'Informatique industrielle de Lille).

[2] Ceci est d'autant plus vrai que la partition en sous-politiques est déterministe.

Nous développons par conséquent une méthode, capable d'ordonnancer cycliquement des gammes non linéaires, prenant par exemple en compte des opérations d'assemblage ou de désassemblage. Cette méthode d'ordonnancement cyclique s'appuie sur les gammes de fabrication décrites dans le formalisme GMT. Un simulateur exécute le plan de production en arbitrant les inéluctables conflits d'affectation, à partir des règles standard de l'ordonnancement ou d'une façon plus originale, en calquant le comportement social des colonies de fourmis. Le résultat de l'ordonnancement cyclique est représenté par un diagramme de GANTT duquel on extrait un modèle GMT de commande : le passage au modèle Grafcet correspondant est automatique. La productivité du modèle Grafcet obtenu est optimale au sens où son temps de cycle est minimal. L'ordonnancement cyclique conduit en effet à la saturation de la ressource goulet du SAP (ressource engagée à 100%). Par contre, nous ne pouvons pas formellement garantir l'optimalité de la productivité sur le plan des encours, car dans notre méthode, le nombre des encours est une conséquence de l'ordonnancement et non un critère.
Nous étudions par la suite la complémentarité du formalisme, des outils d'analyse et de la méthode d'ordonnancement proposée. Nous en présentons également les limites de son champ d'application.

Nous nous appuyons finalement sur un cas d'étude à MPS variable (*Minimal Part Set*) pour monter l'impact des gains de productivité sur la flexibilité de notre système automatisé. Enfin, nous mettons en évidence la limite du système et de sa commande, au-delà de laquelle sa re-conception s'avère indispensable.

Ce mémoire comprend quatre parties où sont successivement exposés l'étude bibliographique et la définition de notre problématique (partie *1*), le formalisme, les outils d'analyse et la méthode d'ordonnancement cyclique (partie *2*), une étude de cas (partie *3*) et finalement les conclusions et perspectives de développement envisageables (partie *4*).

Première partie : Problématique des Systèmes Automatisés de Production.

Table des matières.

1. Les systèmes automatisés de production.

1.1. Aspect historique.

1.1.1. L'évolution des systèmes de production et de la technologie.

L'évolution des moyens de production est étroitement liée à l'évolution du marché. Cette évolution, hormis des secteurs particuliers comme l'alimentaire ou l'habillement (liés respectivement à la péremption des produits et au phénomène de mode), est habituellement décrite sur le dernier demi-siècle par trois volets.

Vers 1950, l'offre est inférieure à la demande, les marges sont confortables, la croissance forte et les marchés porteurs. Les problèmes de l'entreprise sont d'ordre technique : la fabrication est sérielle, les moyens de production dédiés et le calcul des quantités économiques fixe la taille des stocks existants entre les machines. Il s'agit de «PRODUIRE, puis de VENDRE ».

Dans les années 1975, l'offre équilibre la demande et le client choisit son fournisseur. L'entreprise s'attache à faire des prévisions commerciales et à maîtriser son activité de production : organisation des approvisionnements, régulation des stocks et respect des échéances. Il s'agit cette fois de « PRODUIRE ce qui SERA VENDU ».

Au début de ce siècle, l'offre est supérieure à la demande et le fournisseur court après le client. Ceci implique une concurrence sévère face à un client de plus en plus exigeant. L'entreprise se doit de maîtriser les coûts, la qualité et les délais. En outre, les produits souvent personnalisés, ont une durée de vie de plus en plus courte, induisant une adaptabilité réciproque des moyens de conception aux techniques de production. Il s'agit à présent de « PRODUIRE ce qui EST DEJA VENDU ».

Ces changements successifs d'objectifs cohabitent avec la rationalisation des techniques de gestion de production. La mise au point du système KANBAN dure plus de quinze années (1950-1965) et sa diffusion externe à l'entreprise TOYOTA, dix années de plus [SHINGO S., 86]. D'autres techniques de gestion, JIT (Juste à Temps – 1965 - JAPON), MRP (Materiel Requirement Planning – 1965 - USA) ou encore OPT (Optimized Production Technology – 1985–USA) viennent compléter la palette des gestionnaires. La lourdeur et l'aspect « surveillant » des programmes de gestion de production font que ceux-ci reçoivent un premier accueil plutôt hostile. Ce n'est que depuis une quinzaine d'années, que l'outil informatique apporte une réelle contribution dans l'acheminement et le traitement des informations, donnant ainsi une image représentative et fidèle du système à gérer. L'outil fait à présent partie de notre culture.

Les systèmes opératifs ont logiquement suivi l'évolution du marché et les impératifs de la gestion de production. A titre d'exemple, la RNUR (Régie Nationale des Usines Renault) remplace vers 1975, les lignes transfert d'usinage et les lignes de montage des années 1950, par les îlots de production et d'assemblage. Cette décentralisation améliore significativement la gestion des stocks et des encours, tout en permettant une meilleure maîtrise de la qualité. Les avancées techniques, dont font partie de façon non exhaustive – les machines modulables, les changeurs d'outils, la palettisation, les machines à commande numérique, les robots de manutention ou d'assemblage – confèrent par la suite à l'outil de production les caractéristiques attendues de flexibilité, de fiabilité et de sûreté.

L'évolution de ces systèmes opératifs n'aurait pas eu lieu sans l'évolution conjointe des organes de commande. En 1950, le relais électromécanique est LE COMPOSANT des organes de commande. De fait, l'automaticien est avant tout un électromécanicien. Le coût de ces composants (environ 100F l'unité en 1950 [BOYER H. & al., 76]) et le coût prohibitif du câblage des « armoires » de commande entraînent les constructeurs vers d'autres technologies, telles que la fluidique, la pneumatique et l'électronique. La généralisation des circuits imprimés date de 1960, celle des circuits intégrés de 1967. Cette année là, les 4 portes « nand » coûtent 1$US. L'aspect statique de ces réalisations encourage les constructeurs vers des solutions plus flexibles : programmateurs à came, à diodes, à fiches. Mais il faut attendre l'avènement des microprocesseurs pour disposer des premiers automates programmables industriels (API). En 1969, ALLEN-BRADLEY et MODICON répondent à la demande du marché automobile américain. Ils seront suivis en France par MERLIN-GERIN et ALSPA en 1971 [MICHEL G., 88].

Au fil des années, le coût des API évolue mais paradoxalement, pas forcément à la baisse. Ce constat reflète les services de plus en plus élaborés rendus par les API, en particulier sur la nature des entrées/sorties (numériques, analogiques, à seuil) et sur leurs aptitudes à communiquer. Les architectures de commande se décentralisent.

Cependant et malgré les nouvelles possibilités offertes par les API, les solutions technologiques de commande ne passent pas systématiquement par le « tout programmé ». A la fin des années 1970, des considérations d'ordre économique (coût relativement élevé des interfaces avec les API) et d'ordre humain (qualification des hommes du métier à la programmation) entraînent aussi chercheurs et constructeurs vers des solutions câblées et modulaires. Les avancées méthodologiques et technologiques donnent naissance aux séquenceurs et aux CUSA (Cellules Universelles pour Séquences Asynchrones).

1.1.2. L'évolution théorique et les actions de normalisation.

Jusqu'au début des années 1950, les travaux théoriques dans le domaine de l'automatique séquentielle sont restés là où les avait laissés Georges BOOLE en 1847. Les premières méthodologies portent sur la représentation et la simplification des équations booléennes de KARNAUGH, QUINE et McCLUSKEY, puis en 1954, sur la synthèse des circuits séquentiels d'HUFFMAN [NASLIN P., 70]. Dans ce processus de synthèse, le dénombrement des phases au moyen de diagrammes des transitions constitue une approche méthodologique et graphique primordiale. Par la suite, GIRARD et NASLIN introduisent la notion d'états et de transitions et, parallèlement mais indépendamment, PETRI définit une modélisation mathématique de ces systèmes à l'aide de réseaux Etapes-Transitions : les réseaux de PETRI (RdP). Il publiera ces travaux dans [PETRI C.A., 62].

Dans les années 1970, la complexité des systèmes étudiés s'accroît et la nécessité de distinguer le **besoin** de la **réalisation** devient de plus en plus pressante. M. BLANCHARD [BLANCHARD M., 79], expose la situation du moment : « *...la voie nouvelle dans laquelle s'engageait la conception des systèmes logiques se précisait alors. Deux mots clefs la définissent : cahier des charges et modélisation. Partout en France... d'importants travaux de recherche...aboutissaient à des propositions de modèles aptes à représenter un cahier des charges...* » En 1975, sur l'initiative de l'AFCET (Association Française pour la Cybernétique Economique et Technique), une commission composée à parts égales d'industriels et de chercheurs entreprend la rédaction d'un rapport dont l'objectif est d'homogénéiser les différentes approches afin de dégager un outil unique de représentation d'un cahier des charges d'un automatisme logique. Deux années plus tard, cette commission présente un rapport sur la définition du contenu d'un cahier des charges et sur la définition de l'outil de représentation [AFCET, 77] : le GRAFCET est né, et il a immédiatement un vif succès, comme en témoigne son introduction dans les bureaux d'études et dans les programmes d'enseignements (1980). Parallèlement aux travaux des automaticiens, les informaticiens se tournent eux aussi vers la conduite de processus industriels (et militaires). Ils introduisent le concept « temps réel » (Noyau SCEPTRE - 1984), développent les langages orientés « temps réel » (ADA, MODULA2), les langages réactifs (ESTEREL-82) et définissent des protocoles de communications (OSI, protocole à 7 couches...).
Depuis, la commission de l'AFCET et la communauté scientifique qu'elle a engendrée n'ont cessé d'améliorer le modèle GRAFCET, en augmentant d'une part son pouvoir d'expression et d'autre part son indépendance vis à vis de son implémentation technologique.

Déposé en 1979 à l'UTE[1], le rapport de l'AFCET devient la norme française NFC03-190 en novembre 1981, puis en 1988, la norme internationale CEI-848 (renommée IEC-60848 en 1997). En 1993, l'UTE apporte au modèle GRAFCET une série de compléments. La bibliothèque des symboles se trouve augmentée par les notions de source, de puits et de macro-étape. Les structures de contrôle sont consolidées par l'utilisation rendue possible de la hiérarchisation et des ordres de forçage, et deux postulats temporels viennent préciser le comportement du modèle. C'est la norme française NFC03-191. Enfin, en septembre 1995, l'UTE change les statuts de la première norme NFC03-190 en y adjoignant quelques définitions (réutilisation, partage de ressource, couplage) et en autorisant l'utilisation du parallélisme interprété... avec prudence (dixit la norme). Conjointement, la CEI[2] dépose en novembre 1993 le rapport IEC-61131 dont la partie 3 décrit *les langages de programmation devant être utilisés pour les automates programmables*, confortant ainsi le GRAFCET dans son rôle de *formalisme de spécification normalisé.*

Cependant, les modèles GRAFCET établis sont difficilement vérifiables, dés lors que s'accroissent leurs degrés de parallélisme. C'est pourquoi, sur cette même période, les réseaux de PETRI font également l'objet de nombreux travaux théoriques poursuivant un double objectif. VIDAL-NAQUET présente les réseaux de PETRI comme un bon compromis, capable dans certaines limites de représenter des systèmes complexes et de les analyser [VIDAL-NAQUET G. & al., 92]. Le modèle mathématique sous-jacent permet en premier lieu de vérifier de nombreuses propriétés comportementales ou structurelles du modèle [COMMONER F., 71], [BRAMS GW. & al., 83]. En second lieu, la non-normalisation du formalisme des réseaux de PETRI autorise la déclinaison du modèle de base vers d'autres modèles dotés d'un pouvoir de spécification accru [DAVID R. & al., 89] : RdP non autonomes, continus, hybrides, colorés, à prédicats... Forts de leur expérience industrielle, PROTH et XIE recommandent l'utilisation des réseaux de PETRI élémentaires dans la phase de conception préliminaire des systèmes de production et le champ d'utilisation des RdP s'élargit fréquemment aux problèmes d'ordonnancement [PROTH J.-M. & al., 95]. AYGALINC propose *'en prenant quelques précautions'* de transformer le Grafcet initial en RdP afin de valider ses propriétés [AYGALINC P. & al., 92].
Outre les RdP, la communauté des automaticiens a hérité des communautés connexes des mathématiciens, informaticiens et logiciens d'autres outils de spécification capables de

[1] Union Technique de l'Electricité.

[2] Commission Electrotechnique Internationale – *International Electrotecnical Commission.*

preuves : les algèbres, les automates finis, les langages réactifs, la logique du premier ordre et les logiques temporelles. Ces outils sont également utilisés par les spécialistes de la communauté pour vérifier, après traduction, des modèles initiaux généralement spécifiés dans le formalisme GRAFCET ou celui des RdP.

1.2. Situation présente.

Les systèmes automatisés de production sont le lieu de rencontre de nombreux acteurs industriels et universitaires, où chacun partage ses préoccupations et ses méthodes.
Sur le plan industriel, les méthodes de développement et les langages d'implémentation cherchent à favoriser la ré-utilisation et la portabilité de composants dans un environnement dont la tendance est distribuée (ALSTOM[3][NOURY P., 99], PSA[4]...). Regroupés dans le consortium[5] PLCopen, constructeurs et utilisateurs définissent actuellement une méthode de certification des ateliers logiciels et un format d'échange neutre dans le but d'assurer un niveau de portabilité[6] minimal et d'améliorer la maintenabilité. Ces travaux s'appuient sur la norme IEC-61131-3 et ne concernent donc que l'aspect implémentation. L'intérêt industriel des normes n'est plus à démontrer.
Sur le plan universitaire, la communauté participe activement à l'élaboration des normes et des labels. Elle travaille à leur exploitation et à leur intégration dans la chaîne de développement des applications, mais son activité ne se cantonne pas au strict contexte des normes. Comme nous l'évoquons au paragraphe *1.1.2*, la communauté développe hors normes les méthodes de spécification et de validation. Une norme est adoptée parce qu'elle représente la bonne solution à un moment donné. Mais l'évolution des besoins, les développements technologiques et les avancées méthodologiques entraînent sa mise à jour. Nous pensons que le manque apparent de réactivité de la norme vis à vis de son environnement permet en fait d'explorer plus finement l'espace des solutions et d'opérer des choix plus judicieux. C'est un frein nécessaire.

Nous allons à présent définir la classe des systèmes faisant l'objet de notre étude. Nous rappelons également les définitions des termes ultérieurement utilisés.

---

[3] Projet de la norme IEC-61499 [NOURY P., 99]

[4] Méthode SPIRALE développée par PSA [BERGER Ch., 99]

[5]Consortium auxquels adhèrent de par le monde environ 70 industriels et 30 universités. www.plcopen.org

[6] X-Reusability Level – (Cross-Reusability Level). Deux ateliers logiciels peuvent se réclamer du même niveau X-Reusability s'ils utilisent les mêmes formats de données.

1.3. Définitions préliminaires.

1.3.1. Systèmes automatisés de production, systèmes de conduite.

Un système de production est un ensemble de moyens humains et matériels qui interagissent sur des produits manufacturés ou non, dans le but de leur apporter une valeur ajoutée. En réduisant l'intervention humaine, le système de production devient un système automatisé de production (SAP) et le degré d'automatisation traduit la faculté du système à gérer seul le plus grand nombre de situations et/ou à décider de façon optimale du comportement à adopter dans ces situations.

L'accroissement de la productivité et de la flexibilité, l'amélioration de la qualité et de la sécurité et enfin son adaptabilité aux contextes évolutifs de l'entreprise, constituent les principaux objectifs d'un Système Automatisé de Production. Dans [GREPA, 91], l'automatisation est vue comme le transfert partiel ou total des tâches de coordination, auparavant exécutées par des opérateurs humains, dans un ensemble d'objets techniques, logiciels ou matériels, appelé « partie commande ». La partie opérative quant à elle, est constituée des organes physiques qui procèdent au traitement des matières d'œuvre. Les constituants d'un SAP et leurs interactions sont schématisés *Figure 1.* Nous notons la localisation variable de la frontière d'isolement entre la partie commande et la partie opérative.

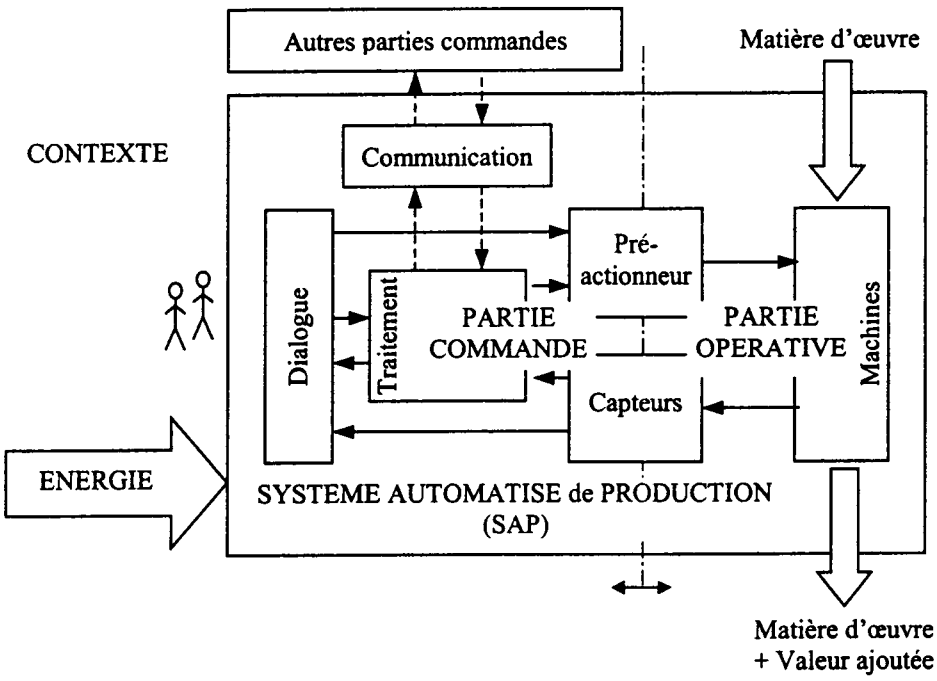


Figure 1 : Schématisation d'un SAP d'après [GREPA, 91]

L'opération d'isolement d'un système décrite dans [DUMERY JJ., 99] fait apparaître les limites de ce système, la vue portée sur ce système et les relations existant entre l'intérieur et l'extérieur de ce système. Cette frontière d'isolement peut-être placée à différents niveaux [GREPA, 91] intégrant dans la commande tout ou partie des éléments d'interfaçage. Dans [DENIS B., 94] l'auteur présente les deux modèles de LE MOIGNE et MELESE dans lesquels apparaissent, avec des degrés de précision variables, trois sous-systèmes : un sous-système opérant ou technologique, un sous-système informationnel ou d'information et un sous-système décisionnel ou de pilotage. DENIS nomme l'ensemble des deux derniers sous-systèmes « système de conduite », définition que nous adoptons.

1.3.2. Systèmes réactifs, systèmes à événements discrets.

Le système de conduite est un sous-système des systèmes automatisés de production qui appartient à la classe des systèmes réactifs. Un système est réactif s'il est réceptif à l'évolution des informations issues d'un processus et de son environnement, et s'il réagit en temps réel, c'est à dire selon des délais cohérents avec la dynamique du processus concerné [PEREZ JP., 90]. La terminologie « réactif » est empruntée aux informaticiens qui classent les programmes informatiques en trois catégories [LE PARC Ph., 94] :

- Les programmes *transformationnels* sont définis comme des fonctions calculant un résultat à partir d'un ensemble de données dans un temps qui leur est propre (compilateur, programmes scientifiques),

- Les programmes *interactifs* qui interagissent avec leur environnement dans un temps qui leur est également propre (système d'exploitation, par exemple),

- les programmes *réactifs* qui sont eux aussi en relation avec l'environnement mais qui évoluent avec une vitesse imposée par celui-ci. Les systèmes de contrôle-commande ou de traitement de signal font partie de cette classe.

En fonction de la nature des informations nécessaires au contrôle du processus, les systèmes réactifs peuvent à leur tour être classés en systèmes continus ou discrets. D'un point de vue formel [CHARBONNIER F., 96], [DUMERY JJ., 99], l'interface d'un système dynamique avec son environnement est modélisée par deux ensembles finis et disjoints. Les entrées $\{e_1,e_2,\ldots e_n\}$ et les sorties $\{s_1,s_2,\ldots s_p\}$. Ces deux vecteurs prennent leurs valeurs sur un corps K : si K=R (ensemble des réels) alors le système est dit continu, si K={0,1} alors le système est dit discret ou séquentiel. Selon [RAMADGE J. & al., 89], les Systèmes à événements discrets (SED) sont des systèmes dynamiques qui évoluent suite à la brusque occurrence d'évènements physiques, se produisant de façon imprévisible et à intervalles irréguliers.

CHARBONNIER, [CHARBONNIER F. & al., 99] reprend cette définition en précisant qu'il s'agit de systèmes *basiquement* asynchrones, *(not clock driven)*. Effectivement, ayant fixé une frontière à un système de conduite, on ne peut pas agir sur ce qui se passe au-delà de cette frontière, c'est-à-dire influer sur la naissance d'événements extérieurs à ce système. Nous ne pouvons que réagir, qu'envoyer des ordres au système extérieur, susceptible en marche normale de générer à son tour le (ou les) événement(s) attendu(s), dans un délai que nous ne maîtrisons pas.

1.3.3. Synchronisme, systèmes asynchrones.

Nous distinguons avant tout, le *synchronisme d'une action* d'acquisition ou d'émission et le *synchronisme d'un système.* Dans le cadre des systèmes de conduite,

- Le synchronisme d'une action d'acquisition signifie que toutes les entrées sont acquises simultanément et celui d'une émission des sorties qu'elles sont toutes émises simultanément. Cette vision du synchronisme provient de l'héritage « top d'horloge » des électroniciens.

- Le synchronisme d'un système signifie que les sorties de ce système sont produites simultanément sur l'occurrence des entrées, autrement dit, qu'elles sont calculées et mises à jour au même instant que l'évolution de n'importe laquelle des entrées dont elles dépendent. Le système a alors une réponse synchrone. Cette interprétation du synchronisme est davantage l'héritage « langages synchrones » des informaticiens.

Dans l'interprétation « système », l'environnement extérieur au système de conduite est *basiquement* asynchrone et les sorties de ce monde extérieur au système de conduite sont émises de façon asynchrone, dans la mesure où de l'intérieur, leur occurrence est imprévisible. L'acquisition synchrone par le système de conduite des événements issus du monde extérieur est une photographie de l'état de ce monde à cet instant, photographie sur laquelle nous pouvons ne pas tout voir.

Toujours avec l'interprétation « système », la réalisation pratique d'un système de conduite est un système physique asynchrone. Le cas contraire impliquerait un ensemble « acquisition, traitement, émission » *infiniment* rapide (Cf. *Figure 2*). Pour que, au sens « système », le modèle d'un tel système soit synchrone, il faut que l'émission au sens « action » soit synchrone (ce qui est possible), que le traitement au sens « système » soit de durée nulle et que l'acquisition au sens « action » soit également synchrone (ce qui est possible, mais avec les risques évoqués plus haut *de ne pas tout voir*).

Enfin, les modèles disponibles (Cf. *§ 2.1*) de ces systèmes de conduite sont généralement synchrones.

Pour résumer, le système de conduite dispose d'entrées asynchrones et le système physique est une réalisation asynchrone. Nous ne pourrons représenter cet ensemble à l'aide d'un modèle synchrone qu'avec le respect de deux postulats :

- Toute évolution du système se réalise avec les entrées figées (atomicité des réactions),
- La causalité est considérée à temps nul (temps de traitement nul).

En pratique, il faut vérifier que toutes les réactions du système de conduite soient terminées avant l'occurrence de l'événement suivant [GAFFE D., 96]. On retrouve la notion « temps réel » sur la cohérence qu'il doit exister entre les délais de traitement et la dynamique du système, notion exposée par PEREZ (Cf. *1.3.2.*).

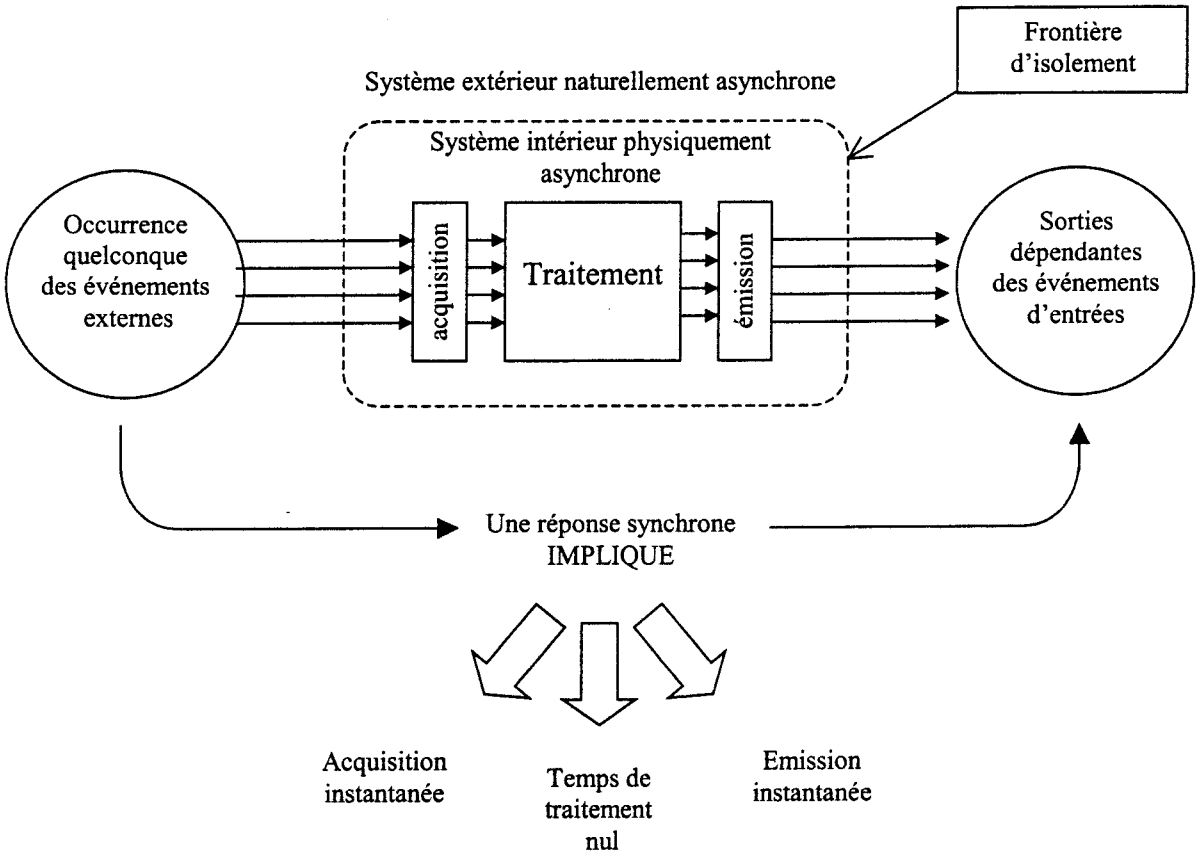


Figure 2 : Synchronisme, asynchronisme, isolement d'un système de conduite

Le cadre de notre étude se précise : nous nous intéresserons au système de conduite des systèmes automatisés de production, systèmes réactifs à des événements discrets. La limite exacte d'isolement de ces systèmes de conduite sera donnée en temps voulu et nous admettons les postulats qui permettent d'adopter un modèle des systèmes de conduite synchrone. Nous définissons à présent le cycle de vie d'un SAP ainsi que les notions de modèles, modélisation et méthodes.

1.3.4. Cycle de vie d'un SAP.

La conception et la construction d'un système automatisé de production comporte d'importants enjeux techniques et économiques dont il convient de limiter les risques. En effet, la recherche et le développement représentent des investissements conséquents et globalement, la productivité du système développé dépend de la productivité de son système de développement (à terme, ventilation des charges fixes d'étude sur les coûts de production). Le développement d'un système comporte les phases de spécification, de conception, de construction, d'intégration et de validation [PEREZ JP., 90]. L'effort de productivité doit porter sur chacune de ces phases.

Les automaticiens ont adopté le schéma, dit « cycle en *V* », que proposait PEREZ dans le cadre des systèmes temps réel. Nous retrouvons fréquemment son adaptation aux domaines des SAP (*Figure 3*).

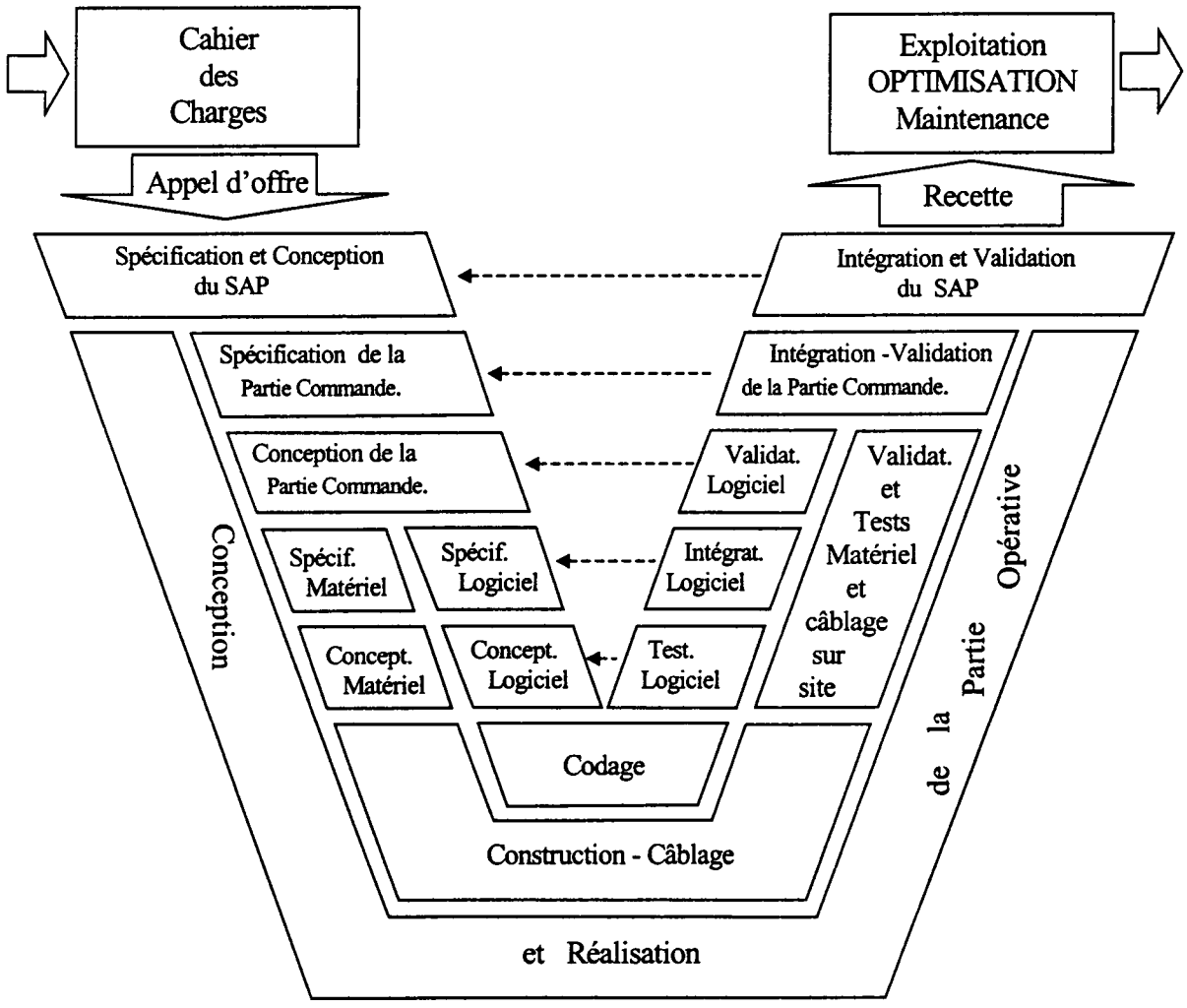


Figure 3 : Cycle de vie d'un SAP

Pourvu d'un axe horizontal temporel et d'un axe vertical de profondeur technique, ce schéma constitue, pour le concepteur-constructeur un guide. Il recense et ordonne les différentes étapes qui conduisent du cahier des charges d'un SAP à son exploitation. A chaque stade du développement d'un système, le concepteur-constructeur spécifie, conçoit et vérifie le respect de son action vis à vis d'un contrat local. Les spécifications situées aux différents niveaux de la branche descendante représentent le contrat local des actions de conception, mais aussi le contrat local des actions d'intégration de la branche montante, *horizontalement* situées au même niveau. Cette correspondance horizontale est *'un plus'* spécifique au SAP mais la démarche n'en reste pas moins descendante. Dans tous les cas, l'invalidation d'un contrat local provoque un retour en arrière et une remise en cause, soit des actions de conception du même niveau, soit des actions d'intégration antérieures.

A l'instar de toutes les méthodes descendantes [MEYER B. & al., 84], le cycle en *V* présente trois inconvénients :

a) Les choix initiaux : ils peuvent se révéler au niveau inférieur impossibles ou trop coûteux à mettre en œuvre physiquement.

b) La non capitalisation du savoir faire : une démarche purement descendante peut conduire le développeur à réécrire l'existant ou une solution semblable à l'existant.

c) L'incertitude : excepté la technique d'échafaudage, a priori très difficile d'emploi dans notre domaine, le test ne peut se faire qu'en fin d'étude: *« il faut attendre d'avoir tout spécifié, pour tout réaliser, pour tout tester... pour enfin savoir si ça marche »* [BERGER Ch., 99] , et d'ajouter « ... *si la demande initialement formulée par le client correspond bien à ses besoins* » .

Le cycle de vie SPIRALE proposé par BERGER [BERGER Ch., 99] pallie en partie ces inconvénients : « *l'objectif poursuivi est de procéder par étapes en partant des spécifications générales vers les spécifications détaillées, comme dans le cycle en V, mais en traversant le V à chaque étape, afin d'obtenir la certitude que chaque fonction est bien comprise et bien réalisée avant de passer à la suivante.* » SPIRALE capitalise le savoir-faire en manipulant chaque composant sous forme de fiche qui recueille l'ensemble de ses caractéristiques. La notion de fiche est en outre en adéquation avec la volonté de concevoir des applications de façon modulaire.

Quoiqu'il en soit, les cycles de vie en *V* ou en spirale mettent en évidence les deux préoccupations de la communauté dans la phase de conception d'un système :

SPECIFIER & CONCEVOIR - INTEGRER & VALIDER.

Ces deux préoccupations sont aussi les axes principaux de la recherche dans le domaine.

La spécification d'un système consiste en la modélisation de ses aspects fonctionnels, informationnels et événementiels [PEREZ JP., 90]. L'intégration–validation d'un système consiste à assembler et à vérifier les éléments constituant ce système. Selon [PATANKAR A.K. & al., 95], la vérification est l'établissement de la preuve de la cohérence intrinsèque d'un modèle, indépendamment de son contexte d'utilisation, alors que la validation est la preuve de l'adéquation du modèle par rapport au cahier des charges qu'il est censé représenter. Précisons les notions de modèles et de modélisation.

1.3.5. Modéles et modélisation.

Dans [DENIS B. & al., 95], DENIS différencie le modèle « résultat » du modèle «formalisme». Par exemple, «Le modèle réseau de PETRI » peut désigner un *formalisme* de type graphe orienté ou le *résultat de l'action* de modélisation. DENIS propose de lever l'ambiguïté en désignant toute vue partielle d'un système par *modèle. Formalisme et langage* sont utilisés pour désigner le moyen symbolique d'exprimer le *modèle* d'un système. Nous adoptons son interprétation.

La modélisation, c'est à dire l'*utilisation* d'un formalisme, est un processus d'abstraction qui isole un ou plusieurs aspects essentiels du système pour ensuite les détailler.

Un formalisme rigoureux permet une lecture non ambiguë du message porté par le modèle mais il peut ne pas couvrir tous les aspects d'un système un tant soit peu complexe et la spécification de tels systèmes peut nécessiter plusieurs *formalismes*. Le ou les *modèles* obtenus doivent être suffisamment complets, structurés et modulaires, de façon à pouvoir prendre en compte une modification de l'environnement et/ou l'ajout d'une fonctionnalité, sans être globalement remis en cause.

1.3.6. Méthodes.

Une méthode est une démarche raisonnée permettant d'atteindre un but fixé. On peut distinguer, en fonction du but visé, les méthodes de spécification-conception et les méthodes de vérification-validation. Le formalisme utilisé dans la phase de spécification et conception d'un système ne permet pas forcément la vérification ou la validation de ce système.

Une méthode de spécification-conception se compose [PEREZ JP., 90]:

- d'outils de modélisation et de règles, qui constituent le langage de représentation,
- d'une démarche de modélisation, appelée aussi stratégie, qui permet de construire, de façon suffisamment formalisée, un modèle d'un système. Il existe a priori deux stratégies : soit le concepteur s'appuie sur un modèle de référence, soit il s'appuie sur un guide permettant une construction progressive du modèle [DENIS B. & al., 95].

Une méthode de vérification-validation comprend [ZAYTOON J. & al., 97]:

- un cadre formel d'expression des propriétés à vérifier et à valider s'appuyant sur un modèle
- un outil de simulation et/ou un système de preuve permettant de vérifier le comportement intrinsèque du modèle et de valider l'adaptation du modèle à son contexte.

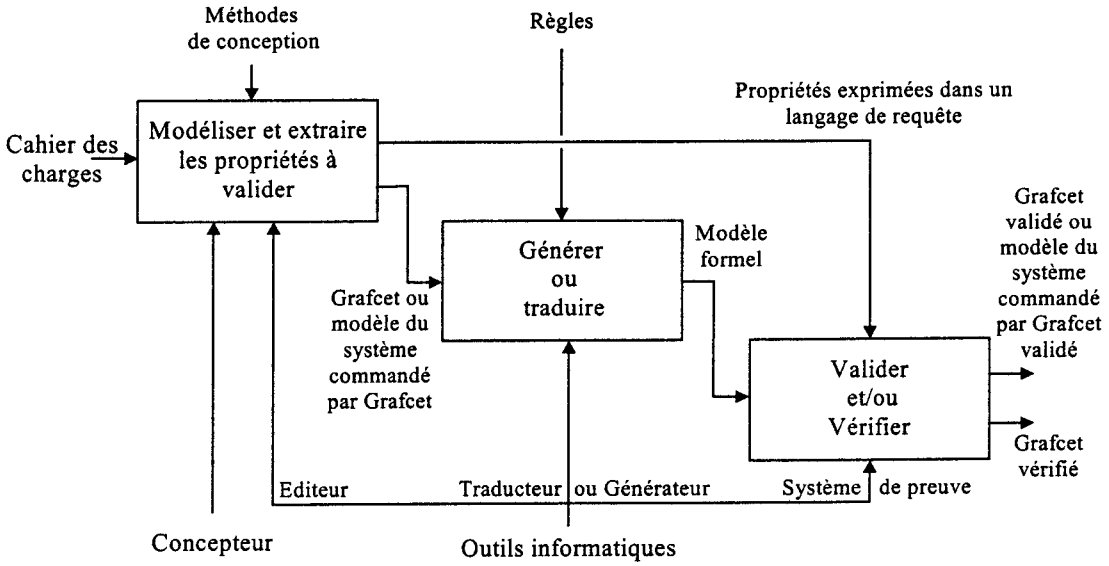
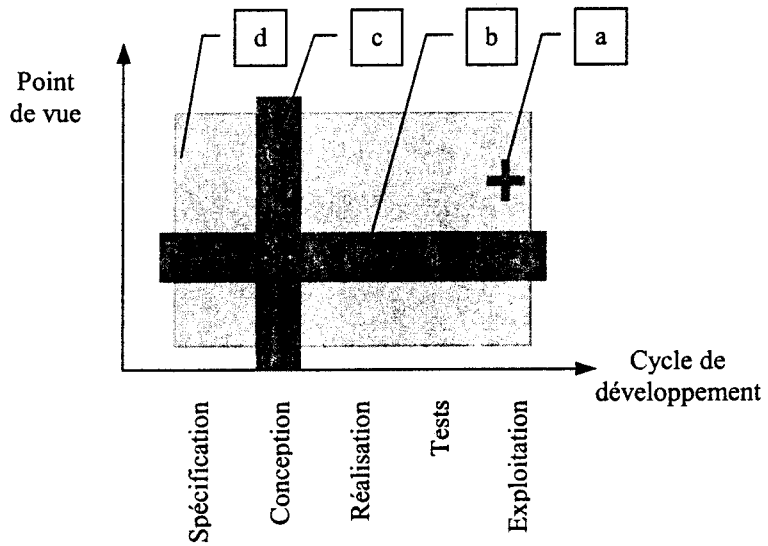


Figure 4 : Exemple d'actigramme représentant une méthode de vérification/validation de Grafcet, d'après [ZAYTOON J. & al., 97]

En général, un formalisme conduit à un modèle qui ne représente qu'un aspect du système à un stade de son développement *Figure 5.cas a*, [DENIS B. & al., 95]



Cas a: Méthode couvrant un point de vue, à un moment du cycle de développement

Cas b: Méthode couvrant un point de vue, tout au long du cycle de développement

Cas c: Méthode couvrant tous les points de vue, à un moment du cycle de développement

Cas d: Méthode couvrant tous les points de vue, tout au long du cycle de développement

Figure 5 : Couverture des méthodes d'après [DENIS B. & al., 95]

Les méthodes « multi-modèles » utilisent conjointement plusieurs formalismes, soit pour couvrir un point de vue sur plusieurs stades de développement *Figure 5.cas b*, soit pour couvrir plusieurs points de vue à un stade donné du développement *Figure 5.cas c*. La méthode universelle *Figure 5.cas d* n'existe pas. La contrepartie de la généralité d'une méthode (capacité à traiter des systèmes variés) est la quasi-impossibilité de couvrir la totalité du développement des systèmes cibles sous tous leurs aspects.

1.4. Définition du champ de l'étude.

Nous venons de rappeler les définitions qui nous permettent de préciser le champ de notre étude. Nous nous intéressons à la conception des systèmes de conduite synchrones, réactifs et à événements discrets des Systèmes Automatisés de Production. Nous souhaitons limiter ce champ d'étude à la partie supérieure du cycle en V des SAP, nous rapprochant dans l'esprit de la première boucle du cycle de vie SPIRALE.

Nous considérons les SAP au sens large, étendant la notion de Systèmes Automatisés de Production aux Ateliers Automatisés de Production. Les machines d'un atelier automatisé seront alors considérées comme les sous-machine et/ou les actionneurs d'un SAP.

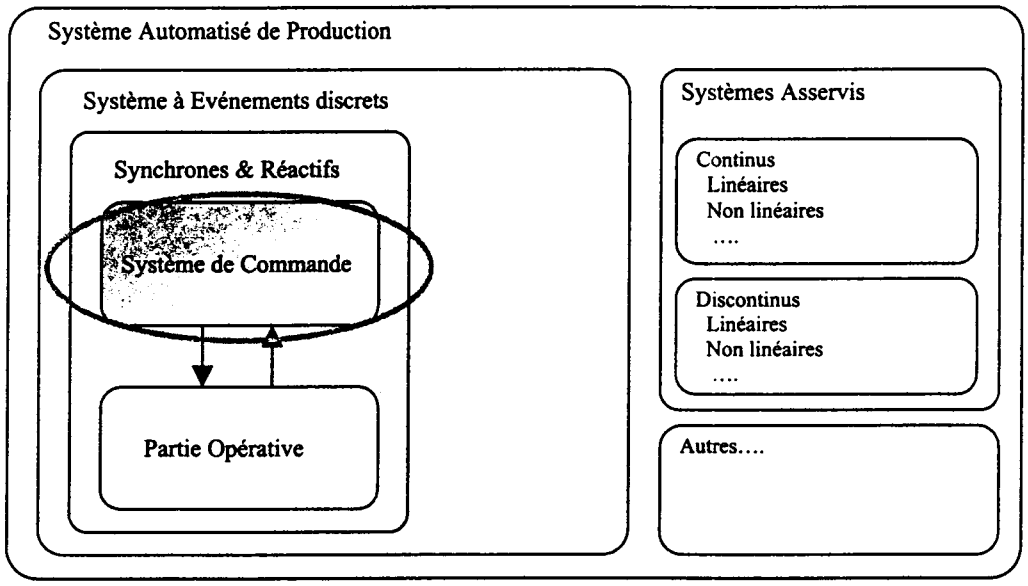


Figure 6 : Champ d'étude.

Nous allons à présent analyser le travail effectué par la communauté (formalismes, méthodes de vérification, de validation...etc). Nous donnerons ensuite notre interprétation de la problématique et nous présenterons les réponses que nous souhaitons y apporter.

2. Formalismes et méthodes, état de l'art.

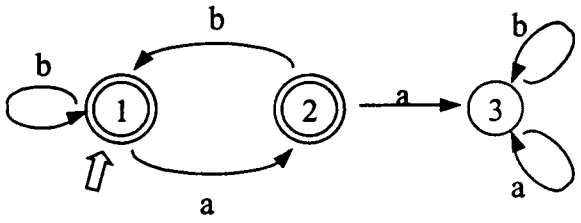
Nous avons évoqué (*§ 1.3.4*) la nécessité de disposer d'une technique de développement productive et sûre. Il est donc légitime de pouvoir contrôler, avant son implémentation et à travers son modèle, que le système étudié est *valide*, c'est à dire que son modèle est en adéquation avec son cahier des charges. Cette activité très importante en génie automatique passe par l'extraction des propriétés de *sécurité* (ce qu'il ne faut pas faire), de *vivacité* (ce qu'il doit faire) et de *célérité* (temps de réponse) [ZAYTOON J. & al., 97]. Elles englobent généralement d'autres concepts comme la fiabilité, la disponibilité ou la durabilité.
Les formalismes de spécification doivent permettre directement ou non d'exprimer ces propriétés. Ils sont classés sur un critère de validation par [DENIS B. & al., 95] en deux catégories : la première catégorie où les formalismes ont été conçus pour que les modèles élaborés soient partiellement ou totalement validables (exemple de formalisme: les langages réactifs), et la seconde catégorie où le formalisme est muni d'outils de validation (exemple de formalisme: le Grafcet). Nous adoptons cependant la démarche d'autres auteurs, [L'HER D., 97] [LAMPERIERE S., 98] ...., qui classent ces mêmes formalismes sur un critère de spécification, suivant qu'ils sont basés sur un système de transitions (concept état/transition) ou sur la mise en œuvre d'une logique temporelle (systèmes axiomatiques).
La vérification, qui consiste à prouver la cohérence intrinsèque du modèle indépendamment de son contexte d'utilisation, ainsi que les propriétés vérifiées, seront étudiées après les formalismes, puisqu'elles sont par définition, liées au choix du formalisme.

2.1. Formalismes de spécification.

2.1.1. Les formalismes du type « système de transitions ».

Les systèmes de transitions [ARNOLD A., 92] sont assez proches dans leur définition des automates finis utilisés dans le domaine des mathématiques. La notion d'automate est associée à celle de langages réguliers, décrits au moyen d'expressions régulières [VELU J., 99]. VELU donne la définition d'un automate fini comme celle du quintuplet suivant :

- un ensemble fini non vide E, représentant les états de l'automate (cf. *Figure 7*),
- un état particulier $I \in E$, représentant l'état initial de l'automate (repéré par une flèche),
- une partie A de E appelée l'ensemble des états acceptants (cerclage double),
- une application δ de $E \times \Sigma \rightarrow E$ qui est la fonction de transition de l'automate,
- un ensemble fini non vide Σ représentant l'alphabet de l'automate.



E = { 1 , 2 , 3 }
I = 1
A = { 1 , 2 }
δ de E x Σ → E
Σ = { a , b }
δ est représentée par sa matrice de transition

δ	a	b
1	2	1
2	3	1
3	3	3

Figure 7 : Exemple d'automate fini – d'après [VELU J., 99]

Les automates tels qu'ils sont définis ci-dessus permettent de définir le fonctionnement entrées/sorties d'un système à événements discrets et en particulier la supervision de ce système. Cette théorie initiée par [RAMADGE J. & al., 89] a été poursuivie par [CHARBONNIER F., 96], les langages réguliers étant utilisés dans l'établissement des preuves respectivement de la supervision et de la commande supervisée.

Les systèmes de transitions sont destinés plus spécifiquement à la modélisation des systèmes parallèles. Formellement, d'après [ARNOLD A., 92], un système de transition est un quadruplet $A = \{ S, T, \alpha, \beta \}$ tel que :

- S représente l'ensemble des états dans lequel peut se trouver le système
- $T \subset S \times S$ est l'ensemble des transitions (ou changement d'état) que le système peut réaliser
- $\alpha, \beta : T \rightarrow S$ sont des applications qui à chaque transition t de T associent les états $\alpha(t)$ et $\beta(t)$ de S, respectivement état origine et état but de la transition t.

Un système de transition étiqueté sur un alphabet σ est un quintuplet $A = \{S, T, \alpha, \beta, \lambda\}$ où λ est une application de T dans σ qui à toute transition t associe son étiquette $\lambda(t)$. L'étiquette $\lambda(t)$ représente l'action ou l'événement qui provoque le changement d'état qualifié de *stimulus* s'il est la conséquence d'événements extérieurs au système ou de *réaction* s'il est la conséquence d'actions effectuées par le système.

Le comportement d'un processus constitué de plusieurs processus en parallèle s'exprime à l'aide de ce formalisme en effectuant l'opération de produit libre. Soient n systèmes de transitions, associés à n processus en parallèle et représentés par n quintuplets A_i avec $A_i = \{ S_i, T_i, \alpha_i, \beta_i, \lambda_i \}$, alors $A = \{ S, T, \alpha, \beta, \lambda \}$ est le produit libre représentant le processus global où $S = \Pi_n S_i$, $T = \Pi_n T_i$, $\alpha = \Pi_n \alpha_i$, $\beta = \Pi_n \beta_i$ *et* $\lambda = \Pi_n \lambda_i$, '$\Pi_n$' représente ici le produit des ensembles pour i de 1 à n . Nous donnons un exemple de produit libre sur la *Figure 8*.

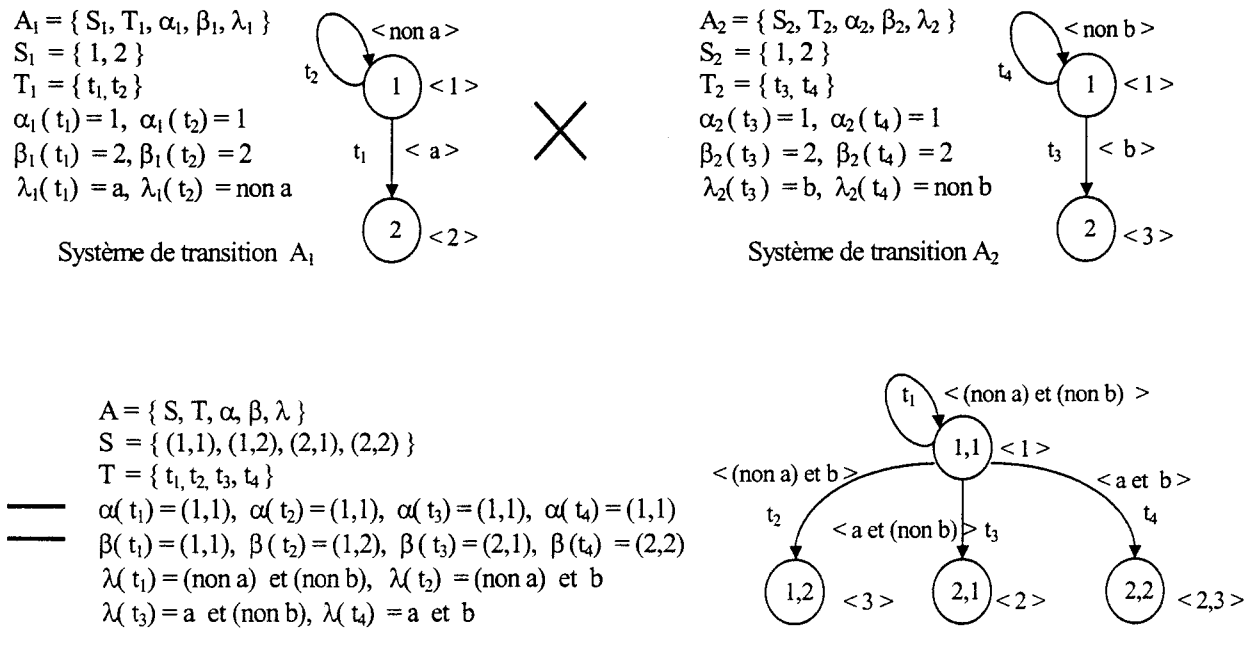


Figure 8 : Produit libre de 2 systèmes de transitions - d'après [LE PARC Ph., 94]

2.1.1.1. Le formalisme « langage réactif synchrone ».

Nous avons donné la définition des langages réactifs (§ *1.3.2*). Les langages synchrones sont des langages réactifs capables de générer du code efficace sous forme *d'automate fini* ou de système d'équations. La rigueur de leurs sémantiques assure au niveau implémentation, que le code généré sur la machine cible a le même comportement que le modèle de départ et les sémantiques formelles de ces langages permettent la construction d'outils de preuve [GAFFE D., 96]. Nous présentons brièvement trois langages.

SIGNAL est un langage déclaratif synchrone de type flot de données, développé à l'IRISA de RENNES [LE GUERNIC P. & al., 91]. Un flot est caractérisé par une suite de valeurs, d'un type donné et d'une horloge qui définit les instants d'occurrence. Il est synchrone au sens où toutes les réactions sont instantanées (cf. § *1.3.3.*), mais si les horloges sont indépendantes, les flots ne sont pas synchronisés entre eux . Il est enfin déclaratif, car le modèle obtenu par ce formalisme « déclare » un ensemble *d'équations dynamiques* reliant entre eux les signaux locaux, d'entrée et de sortie. Les équations « SIGNAL » sont écrites en utilisant quatre opérateurs :

- le retard, qui permet de donner accès à la valeur antérieur d'un signal : *a := b$1 init c*
- le filtre, qui permet de réaliser un sous-échantillonnage : *a := b when c*
- la fusion, qui permet un sur-échantillonnage : *a := b default c*
- la synchro, qui indique que plusieurs signaux ont la même horloge : *synchro{d,e,..,n}*

LUSTRE est aussi un langage déclaratif synchrone de type flot de données, développé dans les années 1985 à l'IMAG de GRENOBLE sur une idée de CASPI [VERILOG, 97]. Il dispose également des opérateurs de retard, de filtre et de fusion (respectivement, pre, when, current) et d'un opérateur « suivi de » permettant d'initialiser une suite. Un programme LUSTRE ou un sous programme est appelé un nœud. A titre d'exemple, la *Figure 9.* représente le « nœud LUSTRE » qui modélise le fonctionnement d'un interrupteur.

```
node SWITCH(set,reset, initial :bool) returns (level :bool) ;
let
        level = initial  →  if set and not pre(level) then true
                            else if reset then false
                            else pre(level)
tel
```

Figure 9 : Exemple d'un nœud LUSTRE, d'après [VERILOG, 97]

Nous donnons *Figure 10* la modélisation du même interrupteur écrite avec SIGNAL.

```
process SWITCH =
{ ? event set, reset
  ! logical level  }
( | level := ( set default ( false when reset ) ) default  zlevel
  | zlevel := level $ 1
  | )
where
logical zlevel init false
end
```

Commentaires :

' | ' signifie en parallèle

zlevel correspond à
l'ancienne valeur de level

Figure 10 : Exemple de programme SIGNAL, d'après [L'HER D., 97]

ESTEREL est le troisième principal langage synchrone utilisé actuellement par la communauté. Il a été initié en 1982 par MARMORAT et RIGAULT au CMA (Centre de Mathématiques Appliquées, Ecole des Mines de Paris, SOPHIA ANTIPOLIS), puis conforté par le travail de Thèse de GONTHIER [GONTHIER G., 88] en développant la sémantique d'ESTEREL. C'est un langage synchrone de type impératif caractérisé par la réactivité, l'atomicité des réactions (entrées figées), la diffusion instantanée des signaux et le déterminisme. La diffusion instantanée signifie que toutes les composantes d'un programme disposent en même temps de la même information (OnToAll). Un système est dit déterministe si la même séquence des entrées produit toujours la même séquence des sorties [BERRY G.,

99]. ESTEREL est un langage impératif, opposé aux langages déclaratifs. ESTEREL n' a pas la vision « flot de données ». Le langage est aussi dit impératif à cause de la syntaxe textuelle à l'aide de laquelle il exprimera la séquentialité et le parallélisme (noté ||). Il est doté de primitives temporelles qui rendent la manipulation du temps intuitive et efficace. Un programme ESTEREL est constitué de plusieurs modules en parallèle qui coopèrent et se synchronisent au moyen de signaux locaux. Sans décrire le langage dans son ensemble, nous donnons ci-dessous les principales familles d'instructions qui le composent:

- les opérateurs classiques (:=, +, *, ... , and, or, not ... >, <, ...),
- les structures de contrôle classiques (if...then...else... end if, loop...end loop...), les fonctions et procédures et des instructions de communication de signaux à tous (emit),
- les primitives temporelles de préemption (do... watching ...timeout, trap...in... endtrap, suspend... when, ...etc.),
- les opérateurs de synchronisation exprimant la séquentialité «;» et le parallélisme «||». Le parallélisme est du type « *fork-join* », les différentes branches sont lancées et s'exécutent en même temps et le parallélisme ne se termine que lorsque toutes les branches sont terminées. Enfin, afin de garantir le déterminisme, il est interdit de partager des variables entre plusieurs branches parallèles[GAFFE D., 96].

La *Figure 11* représente deux « modules » ESTEREL mettant en œuvre le parallélisme (à gauche) et la préemption (à droite).

```
module SYSTEM_1:                    Commentaires:

input BUTTON_A;                     déclaration des entrées
input BUTTON_B;

output LONG_BEEP;                   déclaration des entrées
output SHORT_BEEP;

every BUTTON_A do                   Un appui sur BUTTON-A
   emit LONG_BEEP                   Provoque un bip long
end every
||                                  Et
every BUTTON_B do                   Un appui sur BUTTON_B
   emit SHORT_BEEP                  Provoque un bip court
end every

end module
```

```
module SYSTEM_2:                    Commentaires:

input BUTTON;                       déclaration des entrées
input SECOND;

output LIGHT_ON;                    déclaration des entrées
output ALARM;

abort
   await BUTTON;                    Attendre un appui BUTTON
   emit LIGHT_ON                    Allumer une lumière
when 5 SECOND                       Si pas d'appui dans les 5
do                                  secondes
   emit ALARM                       Alors émettre une alarme
end abort

end module
```

Figure 11 : Exemple de programmes ESTEREL d'après [HAINQUE O., 98]

2.1.1.2. Le formalisme « langage asynchrone ».

ELECTRE est un langage réactif déclaratif asynchrone permettant de spécifier et de programmer des applications orientées « temps réel ». Un programme ELECTRE est composé de trois types d'objets [SUTRE G., 99]:

- les modules, qui représentent les tâches du système. A tout moment d'une exécution d'un programme ELECTRE, ces tâches peuvent être prêtes, interrompues ou terminées,
- les événements, qui sont les signaux de type matériel ou logiciel. Ils sont traités dés leur apparition et éventuellement *mémorisés*,
- les opérateurs, qui permettent de combiner les objets précédents (par exemple, le parallélisme, le séquencement, la répétition, la préemption, le lancement d'un module sur un événement...).

Le langage est asynchrone dans les deux sens du terme (Cf. *2.1.3.*) : Les occurrences d'événements sont asynchrones par rapport à l'évolution des modules et asynchrones entre elles. Dans sa thèse, ROUSSEL expose les difficultés rencontrées lors de la spécification de systèmes aux comportements synchrones dans un langage asynchrone (pour le Grafcet par exemple, les transitions simultanément franchissables doivent être simultanément franchies...) [ROUSSEL JM., 94]. Par la suite, nous ne considérerons plus ce formalisme.

2.1.1.3. Le formalisme « réseau de PETRI ».

Les réseaux de PETRI (RdP)[PETRI C.A., 62] permettent de *modéliser* et de *visualiser* des systèmes présentant du parallélisme, de la synchronisation et/ou du partage de ressources [DAVID R. & al., 89]. Un RdP est composé de deux types de nœuds en nombre fini et non nul: les places et les transitions. Une place est représentée par un cercle, une transition par un trait. Les places et les transitions sont connectées par des arcs orientés. Un arc orienté relie soit une place à une transition, soit une transition à une place. L'ensemble forme un graphe biparti.

Chaque place contient un nombre évolutif entier (positif ou nul) de *marques* ou *jetons* et à un instant donné, le marquage représente l'état du système.

Aux transitions peuvent correspondre des événements dont l'occurrence provoque éventuellement l'évolution de l'état du système. Une transition est franchissable ou validée si chacune des places en amont contient au moins une marque. Une transition franchissable n'est pas systématiquement franchie et il ne peut y avoir qu'un *seul franchissement à la fois.* Le franchissement ou le tir d'une transition consiste à retirer une marque dans chacune des places en amont de la transition et simultanément d'en ajouter une à chacune des places situées en aval.

D'un point de vue formel, un réseau de PETRI ordinaire marqué est représenté par un quintuplet $Q = < P, T, Pré, Pos, M_o >$ où,

$P = \{ P_1, P_2, P_3, ...P_n \}$ représente l'ensemble fini non vide des places,

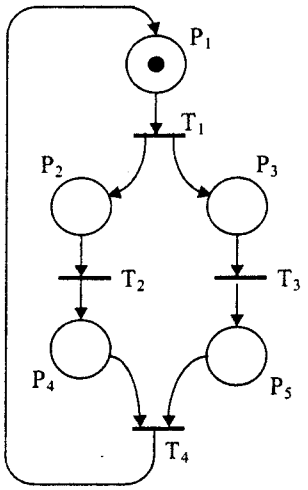
$T = \{ T_1, T_2, T_3, ...T_m \}$ représente l'ensemble fini non vide des transitions,

$Pré : PxT \rightarrow \{0,1\}$ représente l'application d'incidence avant (ou dans N en généralisant),

$Pos : PxT \rightarrow \{0,1\}$ représente l'application d'incidence arrière (ou dans N en généralisant),

M_o est un vecteur de dimension n où chaque coordonnée représente le marquage initial de chaque place du réseau.

Nous donnons ci-dessous l'exemple d'un RdP ordinaire marqué. Les applications *Pré* et *Pos* sont définies par deux matrices d'incidence avant W^- et arrière W^+.



$$W^- = \begin{array}{c} \begin{matrix} T_1 & T_2 & T_3 & T_4 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{matrix}$$

$$W^+ = \begin{array}{c} \begin{matrix} T_1 & T_2 & T_3 & T_4 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{matrix}$$

Mo = (1, 0, 0, 0, 0)

Figure 12: Représentation d'un Réseau de PETRI marqué, d'après [DAVID R. & al., 89]

Les Réseaux de PETRI disposent de propriétés et d'un vocable que nous rappelons à travers quelques définitions usuelles [DAVID R. & al., 89] :

- La matrice $W = W^+ - W^- = [w_{ij}]$ est la matrice d'incidence du réseau,
- Une séquence de franchissements réalisables S à partir d'un marquage M_i, est notée $Mi(S \rightarrow$,
- Un vecteur de franchissement $\underline{S}$ de dimension m de la séquence S est tel que sa $j^{ième}$ composante correspond au nombre de franchissements de la transition T_j dans la séquence S,
- Si la séquence de franchissements S est telle que $Mi (S \rightarrow M_k$ alors, on a l'équation fondamentale $M_k = M_i + W . \underline{S}$,
- Une place est *bornée* pour un marquage initial M_o si pour tout marquage accessible à partir de M_o, le nombre de marques dans la place est fini. Un réseau de PETRI est dit *borné* si toutes les places du réseau sont bornées,

- Un réseau est dit *sauf* (ou *binaire*) pour un marquage initial si pour tout marquage accessible, chaque place contient au plus une marque,

- Une transition T_j est *vivante* pour un marquage M_o si pour tout marquage M_i, accessible depuis M_o, il existe une séquence de franchissements S qui contient la transition T_j à partir de M_i. Un réseau de PETRI est *vivant* pour un marquage initial M_o si toutes ses transitions sont vivantes pour M_o,

- Un RdP a un état d'*accueil* M_a pour un marquage initial M_o si pour tout marquage accessible M_i depuis M_o, il existe une séquence de franchissements S_i, | M_i ($S_i \rightarrow M_a$,

- Un RdP est *réinitialisable* pour un marquage initial M_o si M_o est un état d'accueil,

- Le vecteur $F = (q_1, q_2, \ldots q_n)$ de dimension n est appelé « *vecteur de pondération* » des places d'un RdP. Chaque q_i, nombre entier positif ou nul, représente le poids associé à la place P_i. Soit $P(F)$ le sous-ensemble des places dont le poids n'est pas nul, alors $B=P(F)$ est une composante conservative si et seulement si $F^T.W = 0$. En effet, de l'équation fondamentale, $M_i = M_o + W.\underline{S}$, nous déduisons $F^T.M_i = F^T.M_o + F^T.W.\underline{S}$ or si $F^T.W = 0$ alors $F^T.M_i = F^T.M_o$ quelle que soit la séquence de franchissement $\underline{S}$. Un vecteur $F \geq 0$, qui est solution de l'équation $F^T.W = 0$ est appelé un *P-semi-flot*. De même, un vecteur $\underline{S}$ qui est solution de l'équation $W.\underline{S} = 0$ est appelé un *T-semi-flot*.

Nous rappelons également les définitions des RdP autonomes et non autonomes, des RdP interprétés, des graphes d'états et des graphes d'événements [DAVID R. & al., 89] :

- Un RdP *autonome* décrit le fonctionnement d'un système qui évolue de façon autonome, c'est-à-dire dont les instants de franchissement ne sont ni connus, ni indiqués,

- Un RdP *non autonome* décrit un système dont l'évolution est conditionnée par des événements externes ou par le temps. Un RdP *non autonome* est synchronisé et/ou temporisé : *synchronisé* signifie que le franchissement d'une transition s'effectue quand elle est validée et quand l'événement associé se produit. *Temporisé* signifie qu'à chaque place (P-Temporisé) ou à chaque transition (T-Temporisé) est associée une temporisation de durée éventuellement nulle,

- Un RdP *interprété* (RdPI) est synchronisé, P-Temporisé, et comprend une partie opérative dont l'état est défini par un ensemble de variables $V = \{V_1, V_2, \ldots, V_n\}$. Cet état est modifié par l'ensemble des opérateurs $O = \{O_1, O_2, \ldots, O_n\}$ associés aux places. Il détermine la valeur des conditions (prédicats) $C = \{C_1, C_2, \ldots, C_m\}$ associées aux transitions. A chaque place P_i est également associée une temporisation d_i. Le franchissement d'une transition T_j s'effectue quand T_j est validée et quand la condition C_j est vraie et quand l'événement E_j se produit. Si une marque est déposée dans la place P_i, alors on effectue l'opération O_i et la

marque est indisponible pendant la durée d_i. La *Figure 13. a* représente partiellement un RdP interprété,

- Un RdP est un *graphe d'états* si et seulement si toute transition a exactement une place d'entrée et une place de sortie (*Figure 13. b*). Dans un graphe d'états, il peut y avoir des conflits, mais pas de synchronisation [MARES M., 97],

- Un RdP est un *graphe d'événements* si et seulement si toute place a exactement une transition d'entrée et une transition de sortie (*Figure 13. c*). Un graphe d'événements peut comporter des synchronisations, mais pas de conflit. Il est dual d'un graphe d'états : sa matrice d'incidence est la transposée de la matrice d'incidence du graphe d'états dual et le calcul des T-invariants de l'un revient à calculer les P-invariants de l'autre [MARES M., 97].

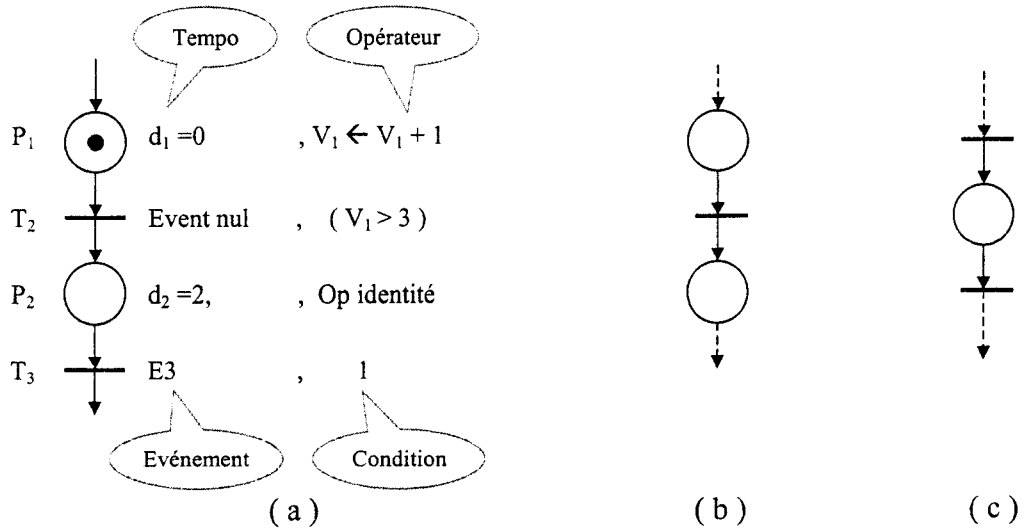


Figure 13: RdP interprété – Graphe d'états – Graphe d'événements, [DAVID R. & al., 89]

Les RdP ont fait et font l'objet de très nombreux travaux [COMMONER F., 71] [BRAMS GW. & al., 83] , tant au niveau spécification [BRAMS WG. & al., 83] [NOYES D., 90] [PROTH JM., 94] [BOUSSEAU F., 97] qu'au niveau de la vérification ou de la validation [MARES M., 97]. Ils ont donné lieu à de nombreuses extensions (RDP à capacité, à prédicats, colorés,...continus, hybrides...) [ALLA H. & al., 98] [DAVID R. & al., 89]. Les RdP possèdent un formalisme fort, mais l'hypothèse du franchissement d'une seule transition à la fois rend les RdP non déterministes, puisque le même vecteur de transitions peut conduire à des fonctionnements différents. Il faut alors envisager et pallier tous les cas de fonctionnement non souhaités, en interdisant par exemple les conflits(Graphes d'événements).

2.1.1.4. Le formalisme « GRAFCET ».

Le GRAFCET, acronyme pour GRAphe Fonctionnel de Commande Etape Transition est un formalisme développé par l'AFCET, normalisé au niveau national [UTE93, 93], [UTE95, 95] et international [CEI-848, 88]. Couramment et dans le même esprit qu'au paragraphe *1.3.5,* la communauté distingue le terme 'GRAFCET' qui représente le formalisme (ou l'outil de modélisation) du terme 'grafcet', résultat de la modélisation.

Règles graphiques d'établissement.

Le GRAFCET repose sur les notions d'étapes et de transitions (*Figure 14.a.*). Les étapes auxquelles sont associées une ou plusieurs actions (ou ordres) sont actives ou inactives : l'état d'une étape est représenté par un booléen (contrairement aux RdP où le marquage est numérique) et si l'étape est active (booléen vrai) alors les actions (ou les ordres) sont exécutées (ou émis). A chaque transition est associée une réceptivité (ou condition) qui peut dépendre de l'état du grafcet. Une étape est représentée par un rectangle, une transition par un trait. Un arc orienté relie soit une étape à une transition, soit une transition à une étape : c'est la règle impérative de l'alternance étape/transition et comme pour les RdP, l'ensemble forme un graphe biparti. La ou les étapes initiales sont représentées par un double rectangle.

Règles d'évolution.

Aux règles graphiques d'établissement sont associées cinq règles d'évolution :

R1 : Au début du fonctionnement, les étapes initiales sont activées.

R2 : Une transition est validée lorsque toutes étapes immédiatement précédentes sont actives. Elle est franchie si elle est valide et si la réceptivité associée à la transition est vraie.

R3 : Le franchissement d'une transition entraîne simultanément la désactivation de toutes les étapes immédiatement précédentes et l'activation de toutes les étapes immédiatement suivantes.

R4 : Plusieurs transitions simultanément franchissables sont simultanément franchies.

R5 : Si au cours du fonctionnement une même étape doit être simultanément activée et désactivée, alors elle reste active.

Compléments structurels.

La norme [UTE93, 93] complète le formalisme initial avec les notions d'étape source et d'étape puits, de transitions source et puits et de macro-étapes.

- Une étape source (respectivement puits) n'est pas précédée (respectivement suivie) par une transition. Elle peut-être initiale et permet entre autres d'introduire des comportements combinatoires conditionnés par leurs activités (*Figure 14.b*).

- Une transition source (respectivement puits) n'est pas précédée (respectivement suivie) par une étape. Par convention, une transition source est toujours valide, son franchissement est uniquement conditionné par sa réceptivité (*Figure 14*.c)

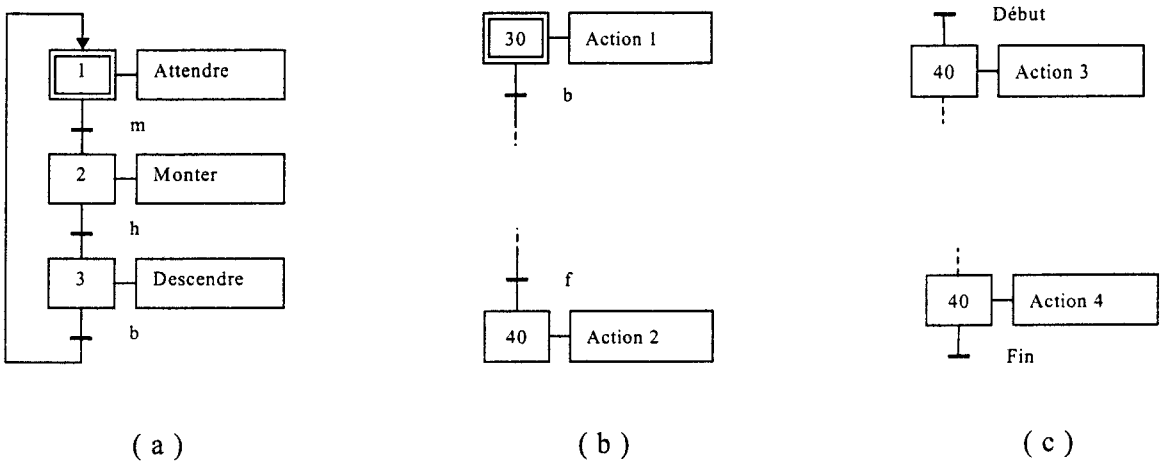


Figure 14 : Grafcet – Etapes source et puits – Transitions source et puits

- Le concept de macro-étape permet une description à plusieurs niveaux, sans se soucier de détails superflus, tout en restant compatible avec l'esprit GRAFCET. Une macro-étape (ME *Figure 15*) est l'unique représentation d'un ensemble unique d'étapes, éventuellement d'autres macro-étapes, et de transitions : cet ensemble est appelé '*expansion* de ME'. Cette expansion commence par une étape d'entrée et se termine par une étape de sortie qui représentent les seules connections structurelles avec le grafcet auquel appartient la macro-étape.

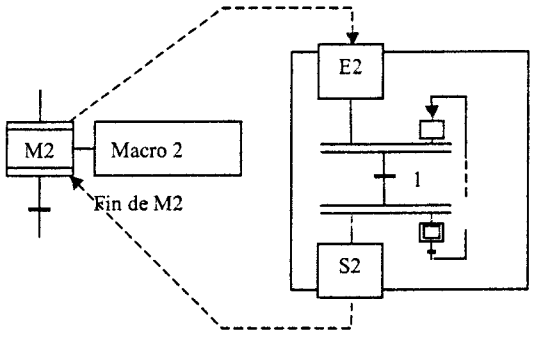


Figure 15 : Exemple de macro-étape, d'après [UTE93, 93]

Comportement temporel.

Cette même norme [UTE93, 93] renforce le formalisme GRAFCET en apportant des précisions sur le comportement temporel du grafcet. La spécification d'un système de

conduite est d'autant plus précise que les frontières de ce système sont clairement définies. L'isolement du système crée une partition de l'univers entre un monde interne et un monde externe. Un grafcet est alors caractérisé par ses entrées et sorties booléennes. Tout grafcet est déterministe au sens où, à une séquence de variation de ses entrées correspond une séquence unique de la variation de ses sorties.

La frontière d'isolement (*Figure 16*) d'un modèle constitue également une frontière temporelle entre une échelle de temps interne et une échelle de temps externe au modèle. Ces échelles sont sans commune mesure et s'appuient sur deux postulats [UTE93, 93]:

Postulat 1 : à l'échelle de temps externe, tout changement d'état des entrées est pris en compte par le système dés son apparition. La totalité des conséquences de cet événement sur le modèle est déterminée à temps nul. Depuis l'extérieur du modèle, les événements d'entrée et l'état des sorties qui en résultent sont vus à la même date.

Postulat 2 : A l'échelle de temps interne, la durée séparant l'instant où une transition est franchissable et l'instant où elle est franchie (appelée aussi durée d'évolution) est aussi petite qu'il est nécessaire, mais non nulle. En conséquence, la durée minimale de l'activité d'une étape ne sera jamais nulle.

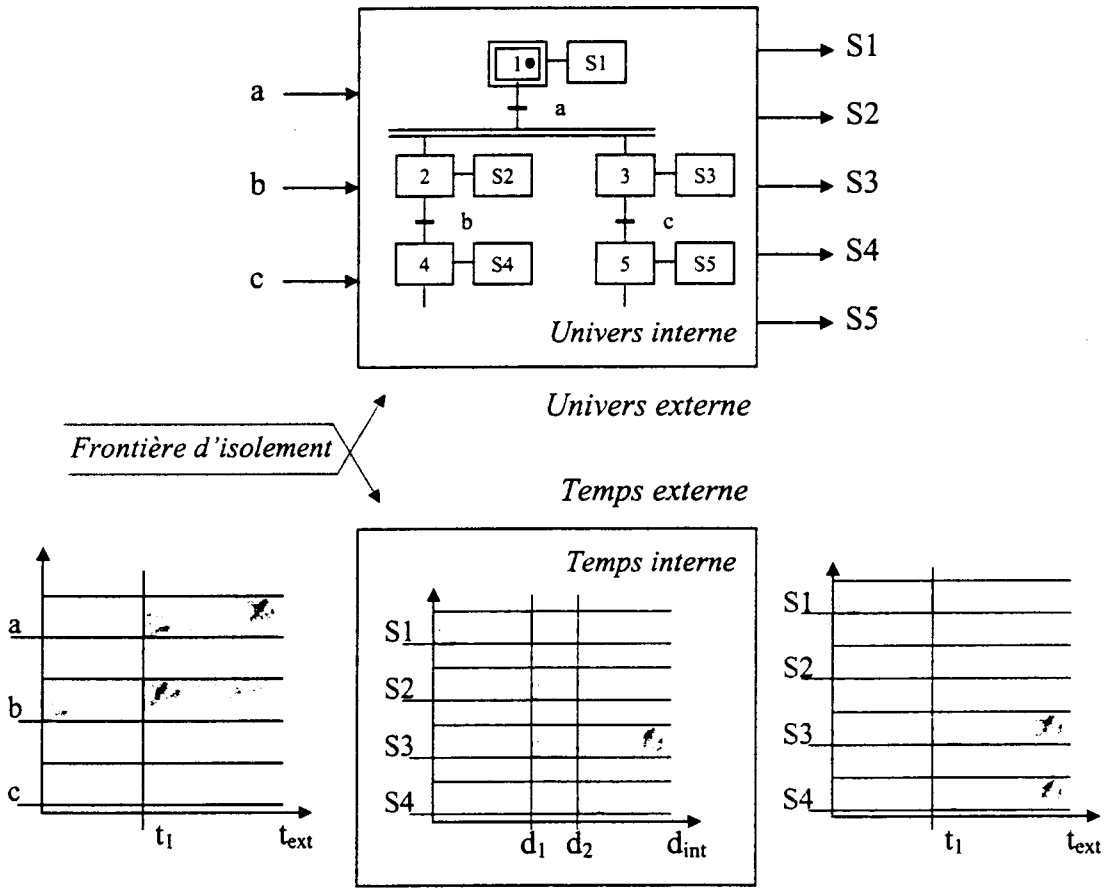


Figure 16: Frontière d'isolement et échelle de temps – d'après [ROUSSEL JM., 94]

Les deux échelles de temps sont sans commune mesure (*Figure 17*): il n'est donc pas possible de faire apparaître plusieurs dates externes et plusieurs dates internes sur un même chronogramme. Nous pouvons cependant représenter ce qui se passe à *l'échelle interne* à *une date externe* (cette notion de « loupe » est l'un des concepts de l'analyse non standard).

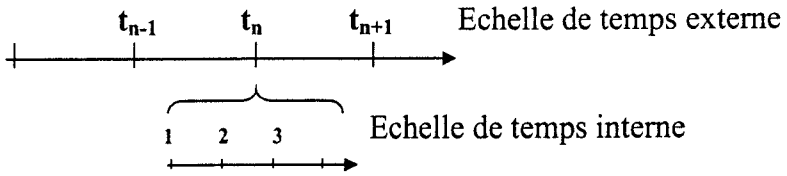


Figure 17 : Echelle interne à une date externe

Cette considération implique qu'à l'échelle de temps interne, le monde externe n'évolue pas. Pour le monde interne, les entrées sont figées et cela revient à dire qu'elles ont été acquises simultanément. D'un point de vue du monde interne, le modèle obtenu est synchrone au sens du synchronisme '*d'action*' décrit dans le paragraphe *1.3.3*.

Le premier postulat traduit qu'avec le GRAFCET, la *causalité* est considérée *à temps nul.* D'un point de vue du monde externe, le modèle obtenu est synchrone au sens du synchronisme '*système*' décrit dans le paragraphe *1.3.3*. Le monde interne ne peut pas omettre d'événements du monde externe.

Le second postulat garantit le respect des règles d'évolution du grafcet. A l'échelle de temps interne, les règles d'évolution relatives au franchissement simultané des transitions (*R4*) et la priorité de l'activation (*R5*) sont respectées.

La double échelle de temps a deux autres conséquences sur le modèle :

- Depuis le monde extérieur, le temps interne n'est pas appréciable et la durée entre deux dates est idéalement petite au sens de l'analyse non standard, [FRACHET JP. & al., 92]. Le monde extérieur ne peut donc pas estimer les durées d'évolution évoquées dans le postulat 2, si ce n'est par une valeur nulle. Il s'ensuit que *les sorties associées aux étapes appartenant à des situations instables ne sont pas émises.* C'est par exemple le cas de la sortie S2 associée à l'étape 2 *Figure 16.*

- De même, depuis le monde intérieur, le temps externe n'est pas non plus appréciable. C'est un idéalement grand petit au sens de l'analyse non standard. Le temps écoulé, entre deux dates distinctes de l'échelle externe, si rapprochées soient-elles, ne peut pas avoir une valeur nulle. Il s'ensuit que *s'ils ne sont pas corrélés, deux événements extérieurs ne peuvent se produire simultanément.*

Hiérarchisation.

Nous reprenons les définitions normées relatives au grafcet connexe, au grafcet partiel et au grafcet global afin d'expliquer les mécanismes et la portée des ordres de forçage.

Un grafcet *connexe* est un grafcet tel qu'il existe toujours une suite de liens graphiques entre deux éléments quelconques (étape, macro-étape ou transition) et un grafcet *partiel* est composé d'un ou plusieurs grafcets *connexes*. Un grafcet *global* d'un système est l'ensemble de tous les grafcets partiels qui définissent le comportement de la partie commande. Chacun des grafcets *connexes* d'un grafcet *global* appartient à un et un seul *grafcet* partiel [UTE93, 93]. Ces définitions sont illustrées *Figure 18,* et elles sont nécessaires à l'établissement des règles de forçage. En effet, l'appartenance d'un grafcet connexe à un grafcet partiel fixe son niveau hiérarchique (noté NHi) et délimite la portée des ordres de forçage.

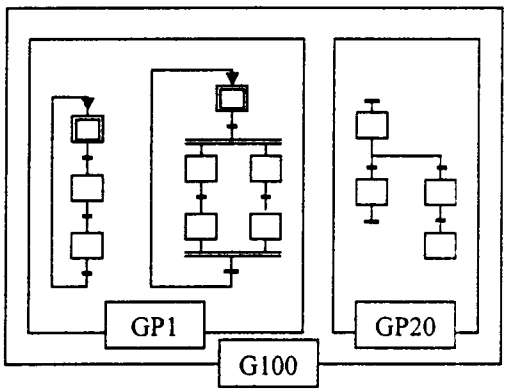


Figure 18: Grafcet global G100 constitué des grafcets partiels GP1 et GP20, d'après [UTE93, 93]

L'ordre de forçage permet d'imposer la situation d'un grafcet *partiel* donné, à partir de la situation d'un autre grafcet *partiel*. C'est un ordre interne dont l'exécution est *prioritaire* sur les autres règles d'évolution. Le grafcet forcé ne peut pas évoluer tant que l'ordre de forçage est présent. Soient les deux grafcets partiels *GP1* et *GP5* représentés *Figure 19*, où les étapes *2* et *5* sont initialement actives. L'occurrence de *a* active l'étape *3* de *GP1* et force le grafcet partiel *GP5* dans la situation *8*. Tant qu'on n'a pas l'occurrence de *b*, *GP5* reste dans la situation *8*, même si l'occurrence de *g* se produit. Lorsque *b* devient présent, le grafcet partiel *GP1* évolue à la situation *4* et alors le grafcet partiel *GP5* prend en compte l'occurrence de *g*.

La notion de forçage a été introduite pour permettre de structurer des problèmes complexes (par exemple, la spécification des modes de marche et d'arrêt). Il faut toutefois conserver la cohérence hiérarchique : le forçage réciproque (étreinte fatale) est impossible et, à un instant donné, un grafcet partiel ne peut-être forcé que par un, et un seul grafcet partiel.

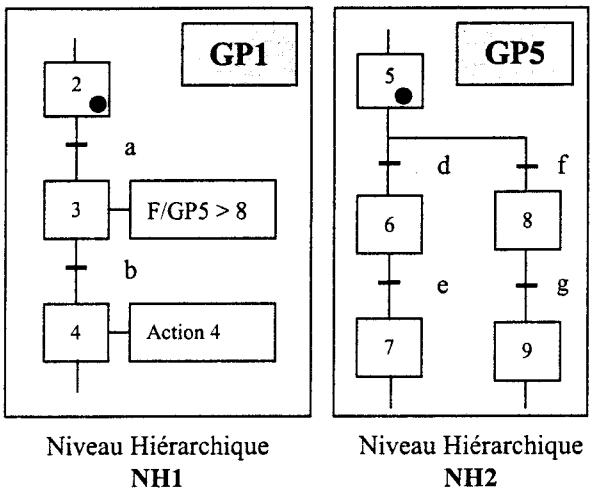
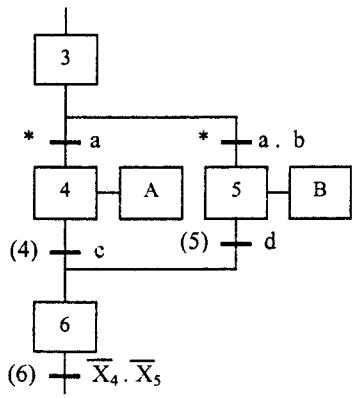


Figure 19 : Forçage du grafcet partiel GP5 à la situation 8 par le grafcet partiel GP1 d'après [UTE93, 93]

Parallélisme interprété.

Outre quelques définitions (réutilisation, partage de ressource, couplage...), les changements de statut de la première norme NFC03-190 [UTE95, 95] autorisent l'utilisation du parallélisme interprété.

Lorsque les réceptivités associées aux transitions validées par une ou plusieurs étapes ne sont pas exclusives, des évolutions simultanées peuvent se produire, conduisant à activer plusieurs étapes à la fois. Ce parallélisme, par opposition au parallélisme structural, est appelé *parallélisme interprété.* Les normalisateurs recommandent d'utiliser ce mode avec prudence, *car la plus grande difficulté réside en fait dans la spécification correcte de la façon dont il se termine.* La norme [UTE95, 95] nous en donne un exemple



A partir de l'étape 3 active :

- Si la condition **a** devient vraie sans que la condition **b** le soit, alors seule l'étape **4** est activée

- Si la condition **b** est préalablement vraie avant que la condition **a** le devienne, alors les étapes **4** et **5** sont activées simultanément.

La réceptivité associée à la transition (6) doit tenir compte des états inactifs des étapes 4 et 5 afin d'effectuer une re-synchronisation avant de poursuivre le séquence commune.

Figure 20: Exemple de parallélisme interprété, d'après [UTE95, 95]

Comme les normalisateurs, de nombreux auteurs nous invitent à la plus grande prudence quant à l'utilisation du parallélisme interprété. Ainsi BLANCHARD [BLANCHARD M., 79] écrit : *« le parallélisme interprété, dont l'intérêt se limite aux réceptivités compatibles*[7]*, est dangereux si on ne l'utilise pas avec prudence ; il ne faut pas oublier en effet de spécifier correctement comment il se termine, ce qui n'est pas toujours simple. Lorsque les évolutions parallèles sont bien déterminées, il est fortement conseillé d'utiliser un parallélisme structural. Dans tous les cas, une chasse aux parallélismes intempestifs, qu'on peut considérer comme des aléas, doit être entreprise ».*

Dans [DAVID R. & al., 89], DAVID formule lui aussi de nombreuses recommandations :

- *Eviter tout conflit dans un grafcet*, c'est à dire que si la validation de deux transitions est conditionnée par une même place, il faut faire en sorte que les réceptivités associées à ces deux transitions ne puissent être vraies simultanément (p 17).

- *Eviter d'utiliser les états internes dans les réceptivités, dans une même composante connexe* d'un grafcet. En effet, cela entraîne une relation qui n'est pas visualisée par le graphisme, entre la situation et les réceptivités. Cela peut-être une source d'erreurs (p 28).

- Ne pas utiliser d'événements internes tels que les fronts montants des variables booléennes associées à l'activité des étapes. Leur interprétation peut être ambiguë (p 42).

- *Ne pas construire des grafcets pouvant avoir des cycles instables.* Ceci est toujours possible si le comportement entrées/sorties est déterministe (p 53).

- *Le graphe indiquant les relations hiérarchiques entre grafcets partiels ne doit pas comporter de cycle.* En outre, il faut manier avec prudence le cas où deux grafcets partiels agissent sur un même troisième (p 73).

Ces conseils et recommandations traduisent le bon sens de leurs auteurs. S'il est vrai que la non-utilisation du parallélisme interprété, l'utilisation restreinte des variables internes ...etc. limitent le pouvoir descriptif du formalisme GRAFCET et s'il est vrai qu'ils rendent plus contraignante la traduction du cahier des charges, la contrepartie de cet effort de spécification non-ambiguë réside dans le déterminisme et la fiabilité du modèle obtenu.

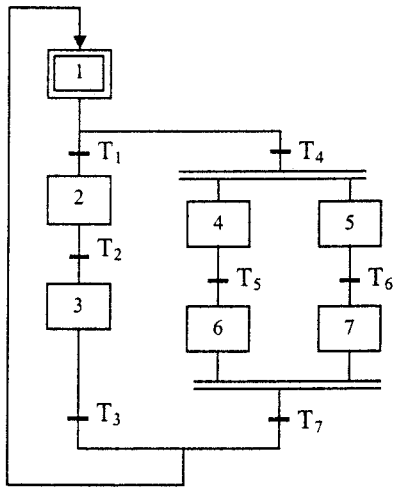
Description mathématique.

La norme [CEI-848, 88] propose une description mathématique du diagramme fonctionnel associé au modèle GRAFCET. C'est un graphe orienté biparti représenté par un quadruplet $< X, T, L, X_o >$ où, X et T forment les nœuds du graphe et sont deux ensembles disjoints,

[7] Deux réceptivités sont compatibles si toutes les combinaisons de leurs occurrences sont possibles : une ou l'autre est vraie, ni l'une ni l'autre n'est vraie ou toutes les deux sont vraies

- $X = \{ X_1, X_2, \ldots , X_n \}$ représente l'ensemble fini non vide des étapes,
- $T = \{T_1, T_2, \ldots , T_m\}$ représente l'ensemble fini non vide des transitions,
- $L = \{ L_1, L_2, \ldots , L_p \}$ représente l'ensemble fini non vide des liaisons orientées entre les nœuds, reliant une étape à une transition ou une transition à une étape,
- $X_o \subset X$, représente les étapes initiales.

Dans sa thèse [DERAIL Y., 84], DERAIL représente l'ensemble L par deux matrices de dimension $m \times n$, matrices que nous notons $ANT = [\ ant_{ij}\ ]$ et $POS = [\ pos_{ij}\ ]$. Elles sont associées respectivement aux liens en amont et en aval entre les transitions et les étapes : $ant_{ij} = 1$ s'il existe un arc orienté de l'étape j vers la transition i, et $ant_{ij} = 0$ dans le cas contraire. $pos_{ik} = 1$ s'il existe un arc orienté de la transition i vers l'étape k, et $pos_{ik} = 0$ dans le cas contraire. Nous retenons cette représentation que nous illustrons *Figure 21.*



$$ANT = \begin{array}{c} \begin{matrix} X1 & X2 & X3 & X4 & X5 & X6 & X7 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \end{matrix} \end{array}$$

$$POS = \begin{array}{c} \begin{matrix} X1 & X2 & X3 & X4 & X5 & X6 & X7 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \end{matrix} \end{array}$$

Figure 21 :Exemple de représentation de la structure d'un grafcet.

Remarque 1 : Interprétation structurelle des lignes et colonnes de *ANT* et *POS*.

- plusieurs '*1*' sur une même colonne de la matrice *ANT* représentent une divergence '*OU*'
- plusieurs '*1*' sur une même colonne de la matrice *POS* représentent une convergence '*OU*'
- plusieurs '*1*' sur une même ligne de la matrice *ANT* représentent une convergence '*ET*'
- plusieurs '*1*' sur une même ligne de la matrice *POS* représentent une divergence '*ET*'.

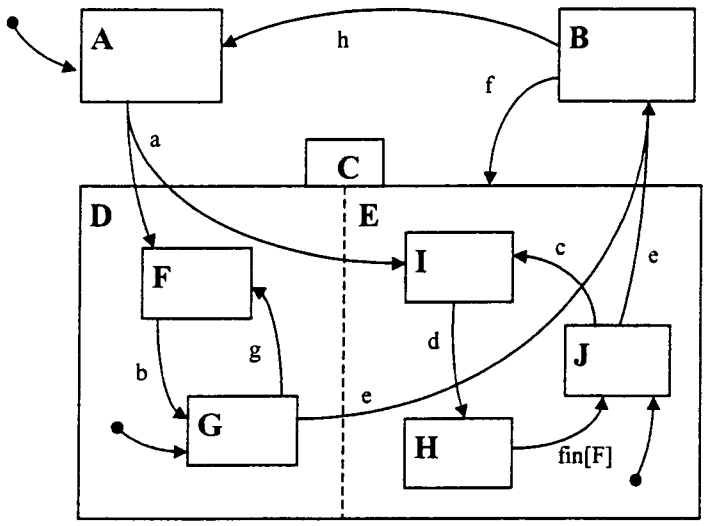
Remarque 2 : Similitudes avec une classe de RdP.

Les arcs d'un grafcet sont représentés par les matrices *ANT* et *POS*, ceux d'un RdP par les matrices d'incidence W^- et W^+, y compris ceux d'un RdP sauf (ou binaire). Dans ce cas et d'un point de vue structurel, on a : $ANT^T = W^-$ et $POS^T = W^+$.

2.1.1.5. Autres formalismes basés sur les systèmes de transitions.

La modélisation des systèmes de conduite fait appel à d'autres formalismes, parmi lesquels nous pouvons citer le Statechart [HAREL D., 87] et l'HyperGrafcet [DUMERY JJ., 99] : Le Statechart est une extension des diagrammes de type état-transition, développé au WEIZMAN institut (ISRAEL). Des rectangles représentent les états, auxquels sont associées les tâches. Les transitions, auxquelles sont associés des événements, sont représentées par des arcs étiquetés. Une transition est franchie si son état source (ou ses états sources) est (sont) actif(s) et s'il y a occurrence de l'événement associé à l'arc.

Le Statechart permet la représentation de la hiérarchie par « encapsulation » (super-états), autorisant la modélisation de systèmes de grande complexité. Il permet aussi d'exprimer l'exécution parallèle (sans explosion combinatoire), la synchronisation, la dépendance et/ou l'indépendance entre tâches. Nous présentons *Figure 22* un exemple de Statechart et un aperçu des mécanismes d'évolution.



Imaginons la suite des événements suivants : a, d, b, e, f, ...

L'état initial est A. L'occurrence de a active simultanément F et I qui peuvent évoluer indépendamment par exemple dans l'état (F, H). L'occurrence de b active G, mais l'événement fin[F] place alors le système dans l'état (G, J). L'occurrence de l'événement e activera alors l'état B. L'occurrence de f active l'état C, c'est à dire D et E dans lesquels les états G et J sont initiaux...etc.

Figure 22 : Exemple de Statechart et de son évolution

L'HyperGrafcet est quant à lui basé sur le modèle GRAFCET, auquel il emprunte en partie la représentation graphique. Il est fondé sur deux concepts clés : le concept d'hyperétape encapsulante, inspiré des super-états du Statechart et le concept de situation d'un développement :

- Une hyperétape est définie par son repère (*Hi*) et par son développement, c'est à dire par un ensemble unique d'étapes et d'hyperétapes (pouvant être elles-même raffinées),

- La situation d'un développement est représentée par l'état actif ou inactif des étapes ou des hyperétapes qui le composent. Les situations particulières d'entrée et de sortie du développement sont fixées par deux jeux d'étiquettes.

La situation d'entrée d'un développement est caractérisée par une étiquette de positionnement qui stipule dans quel état doit se trouver le développement lorsqu'on entre dans ce développement. En l'absence d'étiquette de positionnement, le développement occupera la situation par défaut, représentée par les étapes ou les hyperétapes « par défaut » (encadrement doublé par des pointillés). La situation de sortie précise dans quel état doit se trouver le développement pour pouvoir le quitter et elle est associée à une étiquette de test.

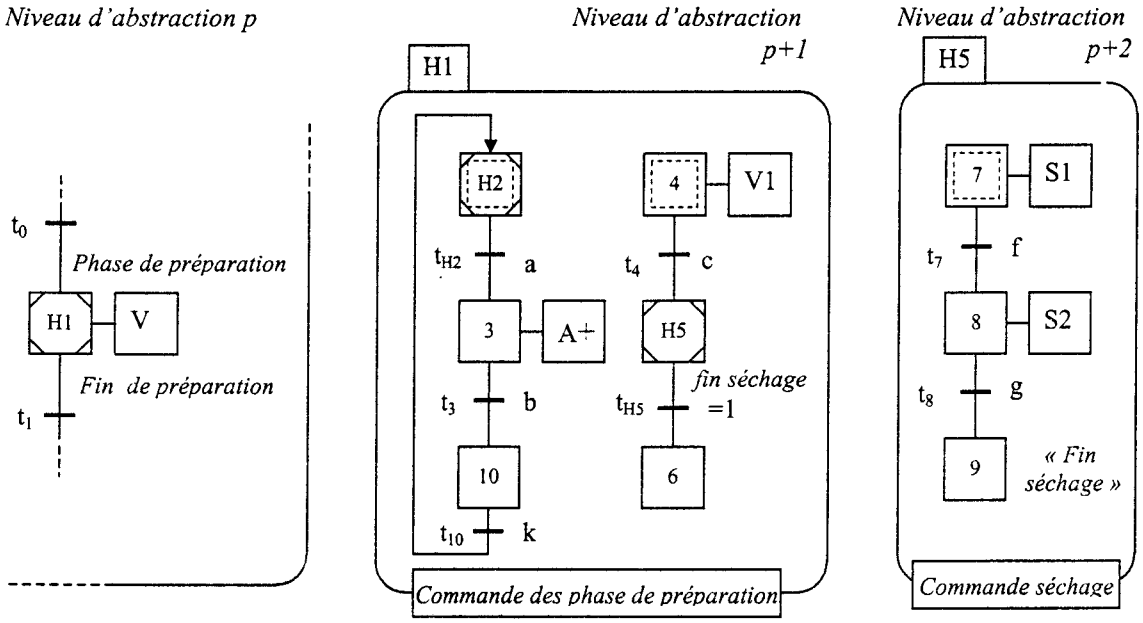


Figure 23 : Illustration d'un hypergrafcet, d'après [DUMERY JJ., 99]

Pour comprendre cet ingénieux mécanisme, nous reprenons *Figure 23*, l'exemple exposé dans sa thèse, par DUMERY : l'auteur associe à l'étiquette de positionnement « *Phase de préparation* » la situation { 3, H5 }, le franchissement de la transition t_0 positionnera le développement de *H1* dans la situation { 3, H5 }, mais l'hyperétape *H5* n'étant pas précédée par une étiquette, le développement de *H5* occupera la situation par défaut {7}. Lorsque *H5* se trouvera dans la situation {9}, l'étiquette de test associée à la situation {9} et précédant la transition t_{H5} sera vraie et cette transition pourra être franchie. Enfin, si l'étiquette de test « *fin de préparation* » est associée à la situation {10,6} du développement de *H1*, alors la transition t_1 sera franchie si et seulement si *H1* est active et si la situation {10,6} est établie.

La structuration hiérarchique en développement et les mécanismes de positionnement permettent d'exprimer clairement différents niveaux décisionnels de commande et confèrent à ce formalisme un grand pouvoir de description.

2.1.2. Les formalismes de type « Logique Temporelle ».

Nous rappelons en premier lieu les définitions de la logique classique et modale.

La logique classique comprend la logique des propositions (logique d'ordre zéro) et la logique des prédicats (logique d'ordre un). La logique des propositions étudie des énoncés (phrases déclaratives) qui sont soit *vrais*, soit *faux*. Les propositions constituent les phrases élémentaires du langage logique, appelées encore formules atomiques ou atomes. Enfin, les phrases logiques sont obtenues en reliant les propositions par des connecteurs logiques dont la sémantique est donnée ci-après : X et Y sont deux propositions,

Négation	$\neg$	si X est vraie alors $\neg$X est fausse
ET	Λ ou $\cap$	(XΛY) est vraie si et seulement si X et Y sont vraies
OU inclusif	V ou $\cup$	(XVY) est vraie si et seulement si X ou Y sont vraies
IMPLIQUE	$\Rightarrow$ ou $\supset$	(X$\Rightarrow$Y) est vraie, sauf si X est vraie et Y est fausse

Un prédicat est une proposition dotée d'un argument. La logique des prédicats utilise les mêmes connecteurs que la logique des propositions et contient cette dernière. La logique des prédicats utilise en outre le quantificateur universel $\forall$ et le quantificateur existentiel $\exists$.

En logique modale, les propositions sont caractérisées par la présence de *modes*, c'est-à-dire de termes qui modifient ou déterminent l'inhérence du prédicat. Ces modes sont par exemple associés aux notions de nécessité, de possibilité, de contingence (éventualité) ou d'impossibilité. Il existe différents types de modalités[8] :

- modalités *ontiques* : "il est nécessaire (possible, impossible, ...) que ...";
- modalités *temporelles*: "il a toujours été que le soleil brille", "il existe un instant dans le futur où la proposition *p* est fausse";
- modalités *épistémiques*: "Laurent sait que ...", "Richard croit que si ... est vrai";
- modalités *déontiques*: "il est obligatoire ...", "il est permis ...".

Les deux premières modalités sont associées à deux opérateurs modaux exprimant la *nécessité* et la *possibilité*, notés respectivement $\Box$ et $\Diamond$.

[8] Les raisonnements utilisant des modalités sont étudiés en logique philosophique et ce n'est qu'à partir de 1915 que C.I. LEWIS intégra l'étude des logiques modales dans la logique symbolique. Ces travaux permirent au début des années soixante à S. KRIPKE de définir la sémantique des mondes possibles. Actuellement les logiques modales (et certaines de leurs extensions) sont étudiées par exemple en Intelligence Artificielle (représentation des connaissances) et en Informatique Fondamentale (vérification et synthèse de programmes).

La logique temporelle est une logique modale dont les opérateurs modaux sont interprétés de façon temporelle. □ est interprété comme un *toujours* et ◊ comme un *éventuellement.*
Enfin, la logique temporelle se décline de deux façons, suivant qu'à un instant correspond un ou plusieurs futurs, donnant respectivement naissance à la logique temporelle linéaire ou à la logique temporelle arborescente.

2.1.2.1. Logique temporelle linéaire

Dans le cadre des systèmes réactifs, le temps de la logique temporelle linéaire est représenté par une suite discrète, linéaire[9] et ordonnée d'instants. Aux opérateurs modaux classiques sont ajoutés des opérateurs temporels adaptés à la classe de problèmes à spécifier. Par exemple, la logique '*Linear temporal Logic*' possède en supplément les deux opérateurs suivants :

- L'opérateur unaire *O*, « *à l'instant suivant* » : soit *f* une formule atomique, la formule *Of* est vraie à l'instant *t* si la formule *f* est vraie à l'instant *t+1*.

- L'opérateur binaire *U*, « *jusqu'à* » : soient *f* et *g* deux formules, la formule *f U g* est vraie à l'instant t_0 si la formule *g* est vraie à un instant t_k succédant à t_0 et que la formule *f* est vraie pour tous les instants de t_0 à t_{k-1}.

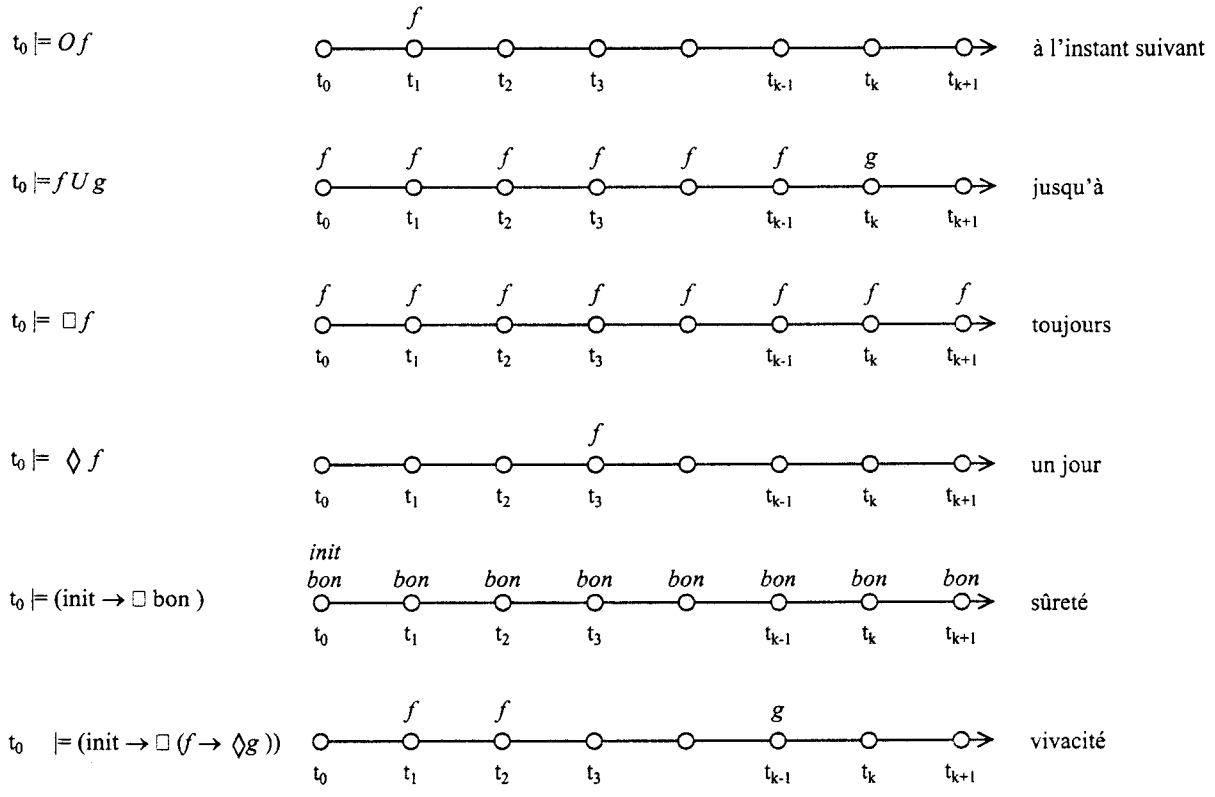


Figure 24 : Opérateurs[10] temporels en LTL, d'après [L'HER D., 97]

[9] Chaque instant ne possède qu'un successeur.

[10] Si on montre *F* sous l'hypothèse *H*, on le note *H* |= *F* - on lit : '*de H on peut déduire F* ' ou '*H satisfait F*'.

En logique temporelle linéaire, il est également possible de spécifier des propriétés. Ainsi, l'exemple de la *Figure 24* décrit les deux propriétés de sûreté et de vivacité :

$t_0 \models (init \rightarrow \Box \text{ bon})$ t_0 satisfait *init* qui entraîne toujours *bon*

$t_0 \models (init \rightarrow \Box (f \rightarrow \Diamond g))$ t_0 satisfait *init* qui entraîne toujours *f* qui entraîne un jour *g*.

2.1.2.2. Logique temporelle arborescente.

La logique temporelle repose sur une modélisation du temps sous forme arborescente, également appelé *'temps ramifié'*. Elle utilise la sémantique des mondes possibles dans lesquels le futur est indéterminé et peut suivre plusieurs voies différentes. *Computation Tree Logic* (CTL) a été introduite par EMERSON et HALPERN en 1985. Les opérateurs spécifiques de la logique arborescente sont les suivants (*Cf. Figure 25*):

- *AX* (dans tous les instants suivants) : une formule *AXf* est satisfaite à l'instant *t* si *f* est satisfaite par tous les instants immédiatement successeurs possibles de *t*.

- *EX* (pour au moins un état successeur) : une formule *EXf* est satisfaite à l'instant *t* si *f* est satisfaite par au moins un instant immédiatement successeurs possibles de *t*.

- $A(f_1 U f_2)$ est satisfaite à l'instant *t* si $f_1 U f_2$ est satisfaite sur tous les chemins issus de *t*.

- $E(f_1 U f_2)$ est satisfaite à l'instant *t* si $f_1 U f_2$ est satisfaite sur au moins un chemin issu de *t*.

Ces deux derniers opérateurs permettent d'exprimer les propriétés de sûreté et de vivacité.

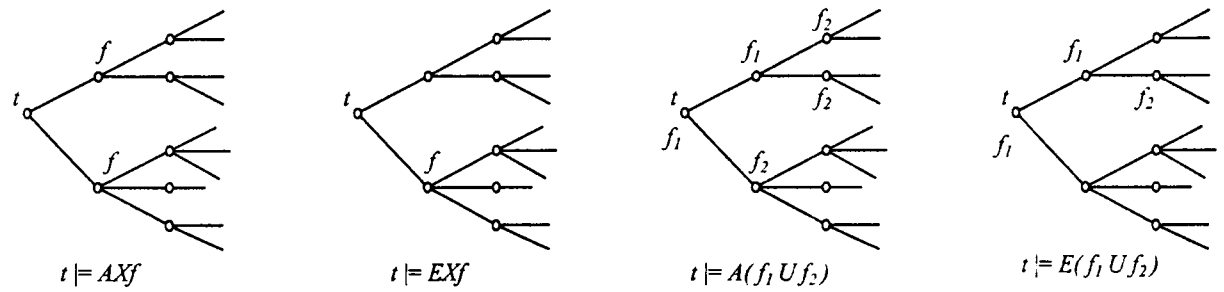


Figure 25 : Opérateurs de la logique temporelle arborescente – d'après [L'HER D., 97]

Outre la spécification du comportement des systèmes réactifs, les logiques temporelles (linéaires ou arborescentes) permettent de spécifier leurs propriétés usuelles. C'est là un avantage majeur de ce formalisme. Notons aussi l'existence de logiques temporelles dotées d'horloge globale qui fixe des bornes aux opérateurs temporels. Par exemple, $\Diamond \leq kq$ signifie qu'une occurrence de *q* surviendra sous k unités de temps à partir de l'instant courant, $\Box > kq$ signifie qu'il faut au moins k unités de temps avant d'obtenir l'occurrence de *q*.

2.2. Méthodes de conception.

Une méthode de spécification et de conception est une démarche de modélisation, appelée aussi stratégie, qui permet de construire de façon suffisamment formalisée, un modèle d'un système à partir du cahier des charges. Mais que cherche-t-on à modéliser ? Une usine ? Une unité de production composée de plusieurs machines ? Une machine ? Les limites entre ces différentes couches ou niveaux d'une application ne sont pas toujours clairement définies et la complexité de la méthode doit être en adéquation avec la taille et la nature du problème traité. Nous proposons de distinguer les méthodes de conception d'architectures de commande des systèmes complexes, les méthodes de conception des systèmes de production et enfin les méthodes de conception de graphes de commande de machines automatisées. Il n'est cependant pas exclu de transposer la philosophie d'une méthode d'un niveau à un autre.

2.2.1. Méthodes de conception d'architecture de commande.

Il existe a priori deux familles de méthodes basées soit sur un *modèle générique*, soit sur une *construction progressive*.

La première méthode consiste à spécialiser un modèle générique au problème à modéliser. Le modèle générique est le fruit d'une réflexion collective illustrée par exemple par le modèle BASE-PTA [11]. Les travaux du CCGA [12] ont conduit à la normalisation d'un modèle de référence des SAP de type *entité-relation* appelé BASE-PTA. Sur cette base, COUFFIN propose une méthode de spécialisation progressive d'un modèle de données de référence vers un modèle adapté à l'intégration de plusieurs activités spécifiques des SAP. Ce modèle générique est commun à toutes les activités du cycle de vie du SAP [COUFFIN F. & al., 00].

La seconde approche, héritée des informaticiens « temps réel », est une démarche constructive et prend en compte plus spécifiquement l'architecture de la partie commande. Les méthodes SART [13] modélisent les aspects fonctionnels, informationnels et événementiels des applications. La méthode développée par WARD & MELLOR [PEREZ JP., 90] propose d'embarquer l'aspect événementiel d'un système temps réel dans son aspect fonctionnel en représentant les flots de données et les flots d'événements et en y faisant figurer toutes les relations d'activation et de désactivation. Les événements sont de type interruption, la logique de contrôle est hiérarchisée et répartie en transformations de contrôle, spécifiées à l'aide de diagrammes et de matrices de transitions.

[11] Base Application et Standard d'Echange pour le Poste de Travail de l'Automaticien.

[12] Centre Coopératif de Génie Automatique

[13] Structure Analysis Real Time

La seconde méthode issue des SART est due à HATLEY et PIRBHAY [PEREZ JP., 90]. Les aspects événementiels (contrôle) et fonctionnels sont représentés séparément par un diagramme de flots de contrôles DFC et par un diagramme de flots de données DFD. Les contrôles sont de type 'changement de valeur' et les événements sont des combinaisons des contrôles. Une unité de contrôle est représentée par un diagramme état-transition (ou une matrice) regroupant les CSPEC- *Control SPECification*. La logique de contrôle est hiérarchisée à raison d'une unité au plus par niveau. Dans sa thèse, DENIS [DENIS B., 94] s'appuie sur la méthode SART-HP. Il élabore un modèle d'implantation, un modèle organique puis un modèle d'affectation. Ce dernier modèle conduit à l'évaluation de l'architecture de commande (rapidité de traitement, débits des transmissions, taille mémoire).
Ces méthodes font actuellement l'objet de nombreux travaux et mettent principalement l'accent sur l'intégration, la ré-utilisation et la commande distribuée : l'intégration parce que la complexité grandissante des projets fait appel à des domaines de connaissances hétérogènes, la ré-utilisation parce qu'il est économiquement souhaitable de capitaliser les savoir-faire et enfin la commande distribuée pour améliorer la flexibilité en exploitant l'offre technologique (démocratisation du protocole TCP/IP, par exemple).

2.2.2. Méthodes de conception des systèmes de production.

La conception préliminaire d'une cellule flexible ou d'une unité de production composée de plusieurs machines est assez représentative du domaine d'application de ces méthodes. PROTH et XIE ont une approche de construction modulaire et progressive du modèle [PROTH J.-M. & al., 95]. Ils proposent un catalogue des modèles RdP des composantes élémentaires d'un système de production (machines dédiées...machines d'assemblage, convoyeurs, ponts roulants, robots, systèmes de stockage, magasins d'outils). Ces modèles élémentaires RdP sont ensuite assemblés pour constituer l'unité. L'assemblage, semblable à la construction d'une *'phrase'*, utilise une *'syntaxe'* comme celle présentée par DAVID. Elle permet d'exprimer le parallélisme, la synchronisation, l'exclusion mutuelle (partage de ressource), la mémorisation, le concept lecture/écriture, la limitation de capacité...etc. [DAVID R. & al., 89]. Cette première phase de *modélisation physique* du système (c'est à dire son agencement) est souvent insuffisante car le modèle ainsi obtenu n'intègre pas l'aspect ordonnancement. En s'appuyant sur la méthodologie développée dans le projet CASPAIM[14], l'équipe PFM[15] du LAIL a mené de nombreux travaux destinés à la modélisation des SPFM[16].

[14] Conception Assistée des Systèmes de Production Automatisée.

[15] Production Flexible Manufacturière du Laboratoire d'Automatique et d'Informatique Industrielle de LILLE.

Du point de vue modélisation KORBAA [KORBAA O., 98] utilise le formalisme des RdP T-temporisés. Les transitions temporisés représentent les opérations et les places représentent soit un stock, soit une ressource. Pour franchir une transition, la durée du franchissement étant égale à la durée de l'opération, il faut un produit (une marque dans la place associée au stock) et une ressource (une marque dans la place associée à la ressource). L'horizon de production est forcé par l'utilisation de places et de transitions spécifiques.

JULIA [JULIA S., 97] décline le modèle précédent en définissant les RdP-temporels T-Temporisés. L'intérêt de ce *nouveau* modèle est de pouvoir représenter séparément les durées des opérations (associées aux transitions – aspect T-Temporisé) et les durées d'attente des produits dans les stocks intermédiaires (associées aux places – Aspect Temporel), ces dernières étant les inconnues du problème d'ordonnancement à déterminer.

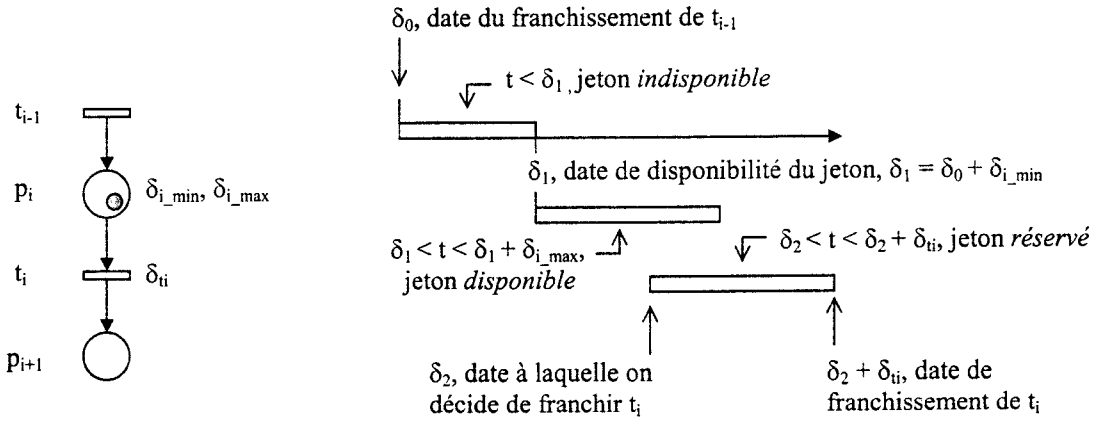


Figure 26 : RdP temporel T-temporisé, d'après [JULIA S., 97]

KORBAA calcule une commande cyclique sous les contraintes de minimisation du nombre des encours et d'optimisation des régimes transitoires. Afin de minimiser les encours, l'auteur introduit la notion de chevauchement de cycle. L'heuristique de recherche de l'ordonnancement est basée sur le placement progressif (au plus tôt ou au plus tard), sur l'exploration en profondeur de solutions et l'évaluation des solutions trouvées par une fonction de coût [KORBAA O., 98].

Pour définir l'ordonnancement d'une production, JULIA utilise un '*algorithme joueur de RdP-temporels T-Temporisés*' possédant un mécanisme de retour en arrière de type '*backtracking*'.

L'ensemble des modèles et méthodes liés aux problèmes d'ordonnancement sont décrits dans l'ouvrage collectif et récent dirigé par LOPEZ et ROUBELLAT [LOPEZ P. & al., 01].

[16] Système Flexible de Production Manufacturière.

2.2.3. Méthodes de conception des graphes de commande des SAP.

En reprenant la classification du paragraphe *2.2.1,* la méthode basée sur un *modèle générique* utilisée à ce niveau touche la gestion des modes de fonctionnement. L'ADEPA[17], forte de l'expérience industrielle acquise par ses membres, propose de regrouper l'étude des modes de fonctionnement des automatismes dans l'unique modèle de référence qu'est le GEMMA[18]. Ce modèle recense pratiquement tous les cas de figure envisageables, alors que le modèle d'une application n'utilisera sous ses propres conditions que les fonctionnalités qui le concernent.

Dans [MERLAUD CH1., 84], [TAILLARD P., 91] et [FROMENT B., 93], les auteurs proposent une *méthode progressive* de construction d'un modèle Grafcet d'une application. L'analyse des fonctionnalités du cahier des charges permet d'établir une matrice où sont consignées les relations d'antériorités existant entre les différentes tâches de l'application. A chaque étape est ensuite associé son contexte amont et aval réunissant respectivement les conditions d'activation et de désactivation de la tâche, la désactivation autorisant à son tour l'activation d'autres tâches. L'étape finale de la méthode consiste à juxtaposer ces tâches en respectant des règles de fusionnement des contextes.

Une autre approche, initiée par RAMADGE et WONHAM [RAMADGE J. & al., 89] et reprise dans les travaux de thèse de CHARBONNIER [CHARBONNIER F., 96] mérite notre attention. La théorie de la supervision permet à terme de construire le(s) graphe(s) de commande d'une application et de prouver a priori qu'un procédé commandé satisfait un cahier des charges imposé. Le principe de supervision est présenté sur la *Figure 27.*

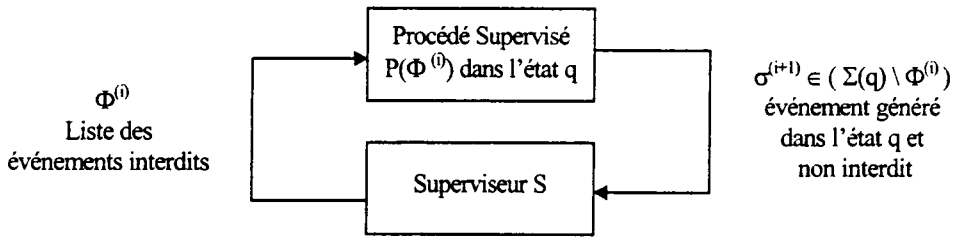


Figure 27 : Schéma de supervision - d'après [CHARBONNIER F., 96]

Le procédé supervisé $P(\Phi^{(i)})$ est supposé être dans un état q. Depuis cet état, $P(\Phi^{(i)})$ peut générer, à l'instant logique (i+1), l'événement $\sigma^{(i+1)}$. Cet événement est un élément de l'alphabet des événements autorisés ($\Sigma(q) \setminus \Phi^{(i)}$), c'est à dire de l'alphabet du procédé moins ceux qui sont interdits. L'occurrence de $\sigma^{(i+1)}$ peut conduire le superviseur S dans un nouvel état qui génère à son tour une nouvelle liste des événements interdits $\Phi^{(i+1)}$...etc.

[17] Agence nationale pour le DEveloppement de la Production Automatisée.

[18] Guide d'Etude des Modes de Marche et d'Arrêt

CHARBONNIER développe ensuite le concept de commande supervisée. Nous reproduisons le résultat d'une de ses études sur la *Figure 28.*

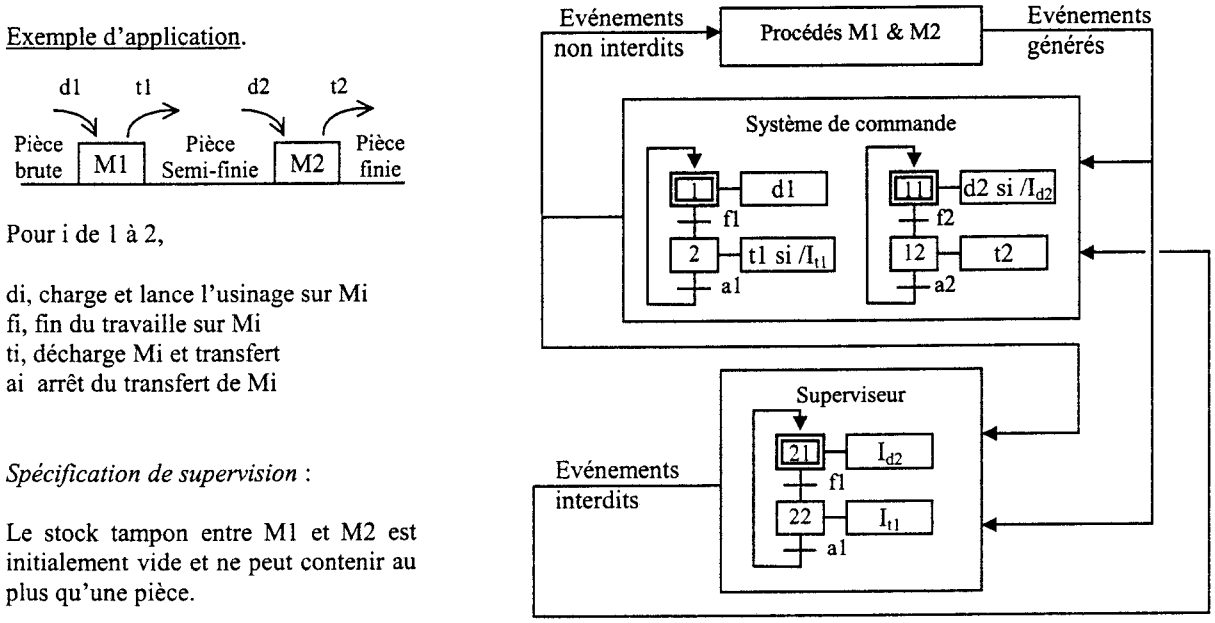


Figure 28 : Commande supervisée – d'après [CHARBONNIER F., 96]

Dans la construction classique d'une commande, on impose en fonction des stimuli et de l'état du système un comportement directif au procédé. L'approche RW (pour RAMADGE et WONHAM) autorise au contraire tous les comportements exceptés ceux qui sont interdits. Nous pourrions qualifier ce type de commande de permissive.

Nous signalons une dernière approche qui porte, il est vrai, sur la conception de RdP, mais qui peut être transposée à la construction d'un modèle Grafcet. Dans [MARES M., 97], l'auteur propose de construire un réseau par composition de réseaux élémentaires dont on connaît les propriétés. Il introduit des règles de compositions (par fusion ou par expansion) qui conservent les propriétés des réseaux composants au réseau composé.

2.3. Méthodes de vérification et validation.

Nous rappelons les définitions données dans [ZAYTOON J. & al., 97].

⇒ La vérification est la preuve de la cohérence intrinsèque d'un modèle indépendamment de son contexte d'utilisation et de son environnement. Les propriétés à vérifier sont :

- La réinitiabilité, qui est la possibilité de retourner à la situation initiale depuis n'importe quelle situation accessible,
- La stabilité, c'est à dire la non-existence de situation totalement instable[19],
- Le blocage, qui correspond à une situation depuis laquelle aucune évolution n'est possible.

⇒ La validation[20] est l'adéquation d'un modèle avec le cahier des charges qu'il est sensé représenter. Les propriétés à valider sont :

- La sécurité, c'est à dire l'assurance que ce qui ne doit pas être fait n'arrivera jamais.
- La vivacité, qui garantit au contraire que ce qui doit être fait sera effectivement fait.
- La célérité qui caractérise les temps de réponse.

Parmi les différents formalismes que nous venons succinctement de présenter, les uns ont été conçus pour que les modèles obtenus soient vérifiables et/ou validables (logiques temporelles, langages réactifs...), d'autres autorisent une vérification/validation partielle des propriétés (RdP...) et certains ne permettent aucune vérification (Grafcet...). L'expression et la vérification des propriétés nécessitent en effet un cadre formel mais ce cadre rigoureux pénalise l'ergonomie de la phase de spécification. C'est pourquoi de nombreux travaux visent à traduire les modèles Grafcet dans des formalismes capables de preuves, alliant ainsi le confort de la spécification à la possibilité de vérifier et de valider le modèle.

2.3.1. Méthode basée sur un modèle décrit en logique temporelle.

Le système et les propriétés à vérifier sont exprimés par des formules de la logique temporelle. La vérification d'une propriété consiste à déduire des formules représentant le système la formule représentant la propriété. Soit M le modèle du système et F la formule à vérifier : pour vérifier que M satisfait la formule F ($M \models F$), il faut trouver un ensemble de formules H_M tel que $\forall H \in H_M$, $M \models H$ et montrer en utilisant le système de déduction que F se déduit de H_M ($H_M \vdash F$). Cette technique, qui s'appuie uniquement sur les formules,

[19] Une situation est totalement instable si sur occurrence d'un seul événement externe, la même séquence d'évolution de produit plus d'une fois.

[20] A priori, la validation de propriétés nécessite donc la modélisation de l'environnement du système.

utilise un moteur de vérification appelé *'démonstrateur de théorème'*. Elle présente néanmoins des limites. Elle n'indique pas clairement les causes de non-vérification et elle n'est pas toujours automatisée. Les BDD (*Binary Decision Diagram*) constituent une autre approche permettant de manipuler et d'évaluer les formules. Les propriétés vérifiables sont celles que l'on peut exprimer (Cf. §*2.1.2.* sûreté et vivacité)

2.3.2. Méthode basée sur un modèle décrit à l'aide d'un langage réactif.

La compilation des langages synchrones produit un automate (ESTEREL, LUSTRE) ou un système d'équations polynomiales représentatif d'un automate (SIGNAL). L'obtention de l'automate (ou du système d'équations) garantit en premier lieu la justesse de la sémantique (absence de récursion, de blocage, de cycle de causalité). La vérification s'effectue en *observant* l'automate résultant (ou le système d'équations) à l'aide d'outils informatiques spécifiques à chaque langage. L'outil de vérification AUTO réduit la taille des automates issus des compilateurs ESTEREL et LUSTRE afin de les rendre plus observables alors que l'outil SIGNALI permet d'effectuer du calcul formel sur les systèmes d'équations produits par SIGNAL. Ces langages montrent principalement la propriété de sûreté.

2.3.3. Méthode ' Model Checking'.

Le *Model Checking* est une technique de vérification faisant appel ou non à la logique temporelle, le système étant défini par un système de transitions [L'HER D., 97].

Dans la première approche (*model checking de logique temporelle*), les propriétés sont définies en logique temporelle. La vérification consiste à établir si le modèle est un modèle de la formule logique exprimant la propriété.

Dans la seconde approche (*model checking de comparaison*), le système et la propriété sont exprimés par deux systèmes de transitions. Il faut alors établir si les deux systèmes ont des comportements équivalents (bisimulation).

2.3.4. Méthodes basées sur un modèle décrit à l'aide des RdP.

Il existe a priori trois méthodes pour rechercher les propriétés d'un RdP :

- La méthode basée sur l'établissement de l'arbre des marquages accessibles.
- La méthode basée sur l'algèbre linéaire.
- La méthode basée sur l'algèbre MAX+ restreinte aux RdP de type graphes d'événements temporisés (GET).

L'arbre des marquages est composé de nœuds qui correspondent aux marquages accessibles, et d'arcs correspondant aux franchissements des transitions, faisant passer d'un marquage à un autre [DAVID R. & al., 89]. Il est construit à partir du marquage initial et permet la construction du graphe de couverture par fusionnement des nœuds qui correspondent à un même marquage (Cf. *Figure 29*). L'examen du graphe de couverture permet de visualiser les situations de blocage, la bornitude, la réinitiabilité et la vivacité.

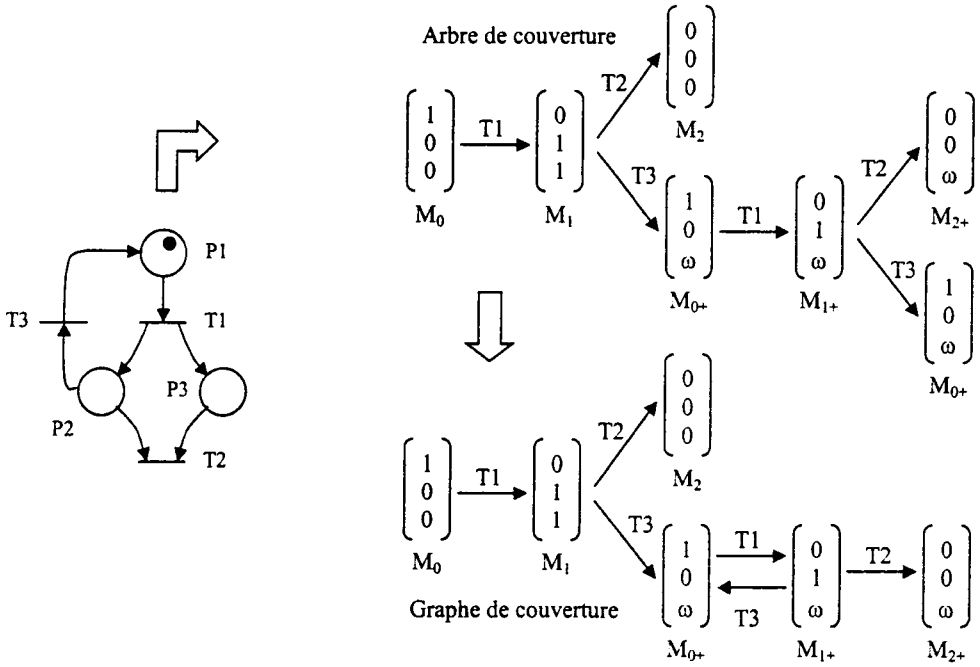


Figure 29 : Arbre[21] et graphe de couverture – d'après [DAVID R. & al., 89]

La seconde approche s'appuie sur l'algèbre linéaire exposée au paragraphe *2.1.1.3*. Elle permet le calcul des composantes conservatives minimales du RdP. Si le RdP peut être reconstruit par combinaison linéaire des supports de *P-semi-flots* minimaux alors ce RdP est borné. S'il existe un T-semi-flot $\underline{S}$ tel que S soit une séquence de franchissements à partir du marquage M, alors $M(S \rightarrow M$, ce qui signifie que le réseau est réinitialisable (le tirage de la séquence S à partir du marquage M conduit à nouveau au marquage M). La connaissance de ces invariants autorise également l'étude du temps de cycle (à vitesse maximale[22]) des RdP temporisés, et des graphes d'événements temporisés (GET) [MARES M., 97].

La construction du graphe des marquages ou le calcul des invariants peuvent devenir fastidieux voire difficile lorsque la taille du RdP s'accroît (explosion combinatoire). La

[21] Lorsque l'un des marquages est strictement supérieur au marquage initial M_0, on adopte le marquage par la lettre 'ω' pour chacune des composantes supérieures aux composantes correspondantes de M_0.

[22] A vitesse maximale, toute transition est franchie dés que les conditions nécessaires sont satisfaites.

parade est alors d'utiliser une méthode de réduction qui préserve les invariants de marquage et les propriétés de vivacité et de bornitude du RdP. Le respect d'un ensemble de règles permet de supprimer des places et/ou des transitions tout en conservant les propriétés précitées. La méthode de réduction est automatisable (programme) [DAVID R. & al., 89].

La dernière approche algébrique abordée est celle de l'algèbre MAX+ [MARES M., 97, MARES M. & al., 94], [MENGUY E., 97]. Cette approche est limitée aux graphes d'événements P-temporisés fonctionnant à vitesse maximale. L'objectif est d'établir une équation de type *état* de l'application. Les variables d'état sont les dates d'occurrence $x_i(k)$ de franchissement des transitions, $x_i(k)$ représentant la date du $k^{ième}$ franchissement de la transition T_i. A chaque place P_j est associée une temporisation d_j. Un jeton arrivé dans la place P_j est indisponible pendant la durée d_j. Le $k^{ième}$ franchissement de T_i peut se produire après la $k^{ième}$ arrivée d'un jeton dans les places en amont de T_i ; ces places auront donc reçu de leurs transitions amont (une pour chaque place) $k\text{-}M_0(P_j)$ jetons où $M_0(P_j)$ est le marquage initial de la place P_j.

Les auteurs travaillent dans l'ensemble $R \cup \{-\infty\}$. *Cet ensemble,* muni des lois *max* et + notées respectivement $\oplus$ et $\otimes$, est un *dioïde* dont les lois respectent les axiomes suivants : la loi $\oplus$ est associative, commutative, idempotente et admet un élément neutre ε ; la loi $\otimes$ est associative et distributive par rapport à la loi $\oplus$, elle admet un élément neutre e et ε est absorbant.

$\forall(a,b) \in R \cup \{-\infty\}, \quad a \oplus b = max(a\ ,\ b)$ et $a \otimes b = a + b$, $\varepsilon = -\infty$ et $e = 0$

La structure ($R \cup \{-\infty\}$, $\oplus$, $\otimes$), notée R_{max}, est un dioïde réel commutatif. On définit sur ce dioïde les matrices et le calcul matriciel classique.

Si $A \in (R_{max})^{nxp}$ et $B \in (R_{max})^{pxq}$ alors $C = A \otimes B \in (R_{max})^{nxq}$ avec $c_{ij} = max_k(a_{ik} + b_{kj})$

Si $A \in (R_{max})^{nxp}$ et $B \in (R_{max})^{nxp}$ alors $C = A \oplus B \in (R_{max})^{nxp}$ avec $c_{ij} = max(a_{ij}, b_{ij})$

Le vecteur d'état de l'application est alors le suivant : $X(k) = A \otimes X(k-1)$ avec $X(k) = \{x_i(k)\}$.

La matrice A, appelée matrice d'évolution du RdP, est obtenue par la transcription des équations d'évolution associées à chaque transition dont le principe est exposé *Figure 30.*

Par la suite, l'auteur [MARES M., 97] considère la valeur propre[23] λ du système dans R_{max} tel que $X(k+1) = \lambda \otimes X(k)$. En fonctionnement périodique, chaque transition est franchie toutes les λ unités de temps, soit $x_i(k+1) = \lambda + x_i(k)$. La valeur λ représente donc le temps de cycle du GET à vitesse maximale. L'auteur détermine par le calcul la première date qui respecte le

---

[23] $\lambda = max_i (min_k \{ (A^n_{ij} - A^k_{ij}) /(n-k) \}) \quad \forall j \in \{1..n\}, i \in \{1..n\}, k \in \{0..n-1\}$ avec $A^k = A \otimes \ldots \otimes A$, k fois

La soustraction et la division sont celles de l'algèbre$(R, +, .)$ et avec la remarque que $\varepsilon - \varepsilon = \varepsilon$ | MARES M.,94]

fonctionnement périodique et la première qui ne le respecte pas. Il en déduit respectivement une borne supérieure t_{sup} et une borne inférieure t_{inf} de la fin du régime transitoire. L'examen des dates de franchissement entre ces deux bornes permet de déterminer la date t_p de cette fin de transitoire : $t_{inf} < t_p \leq t_{sup}$. En dehors de la simulation, l'algèbre MAX+ est à notre connaissance le seul outil qui permette de calculer le temps du régime transitoire.

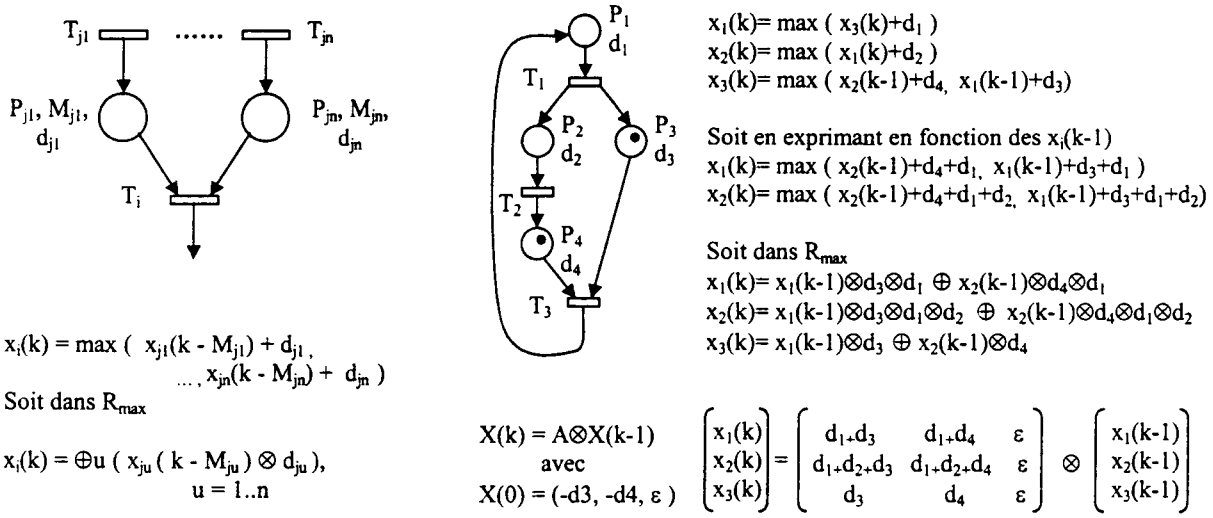


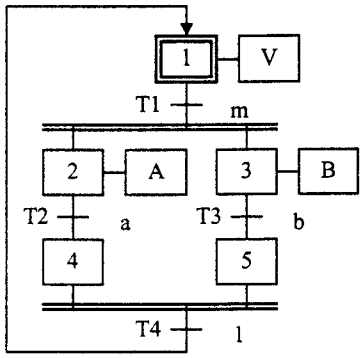
Figure 30 : Principe et exemple d'obtention de la matrice d'évolution.

Notons également que la représentation d'états ainsi obtenue présente de grandes analogies avec la théorie classique des systèmes linéaires continus. Dans [MENGUY E., 97], l'auteur développe une méthode basée sur la commande adaptative où il introduit la notion de correction proportionnelle et d'identification et l'applique à l'ordonnancement en juste à temps des SED modélisés par des GET. La commande permet ainsi au SED de suivre (de s'adapter) à une consigne de production.

2.3.5. Méthodes basées sur un modèle décrit à l'aide du GRAFCET.

La première méthode présentée dans [BLANCHARD M., 79] construit une table, puis un graphe des situations accessibles à partir de(s) l'état(s) initial(aux). Les nœuds correspondent aux situations. Ils sont reliés par des arcs orientés associés aux transitions qui permettent de passer d'une situation donnée à toutes les situations possibles (suivant les transitions sensibilisées[24]). BLANCHARD introduit la notion de *grafcet autonome* - *« On appelle grafcet autonome le grafcet obtenu en ne conservant que l'interprétation (synchronisation) interne, toutes les réceptivités externes prenant une valeur indifférente 0 ou 1 »*

[24] Une transition est sensibilisée si elle est validée et si la réceptivité interne associée est vraie.



Situation	transition sensibilisée	transition franchie	Nouvelle situation
1	T1	T1	2,3
2,3	T2, T3	T2	4,3
2,3	T2, T3	T3	2,5
2,3	T2, T3	T2, T3	4,5
4,3	T3	T3	4,5
2,5	T2	T2	4,5
4,5	T4	T4	1

Table des situations

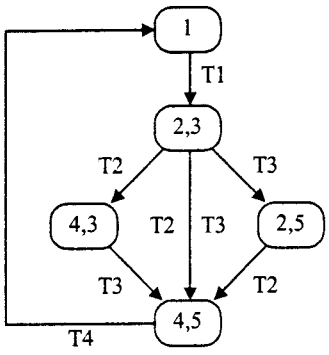


Figure 31 : Etablissement du graphe des situations.

L'analyse des propriétés se fait sur le graphe des situations accessibles issu du modèle autonome. On peut observer les propriétés de réinitiabilité (retour à la situation initiale), de non-blocage, de vivacité (toutes les étapes sont accessibles) et de sûreté (des états incompatibles ne sont pas accessibles simultanément). Puis en respectant un ensemble de règles, on déduit de ces résultats la validation (ou non) du cahier des charges.

On conçoit facilement qu'un modèle Grafcet présentant un fort degré de parallélisme fournisse un graphe des situations accessibles de taille inobservable (explosion combinatoire). On peut cependant réduire ce graphe par analyse de la chronologie et/ou par la suppression des évolutions impossibles (physiquement, temporellement impossibles).

Une autre approche est basée sur la simulation. Son principal intérêt est de libérer l'analyste de tous les calculs combinatoires et d'évolution. Il peut ainsi se concentrer sur les conséquences de la variation des entrées qu'il a proposée à l'évolution du modèle Grafcet simulé. Cependant, la non-exhaustivité des combinaisons d'occurrence des entrées (choix des scénarii) et les approximations dans la définition de l'environnent limitent la portée de la méthode.

Comme nous l'avons indiqué précédemment, ces méthodes ont leurs limites. C'est pourquoi de nombreux travaux portent sur la traduction des modèles Grafcet dans des formalismes davantage orientés vers la vérification.

2.3.5.1. Traduction du modèle Grafcet en RdP.

AYGALINC et DENAT proposent de transformer un modèle Grafcet en un modèle RdP autonome (Cf. *2.1.1.3*) puis en RdP P-temporisé pour évaluer les performances de l'application. [AYGALINC P. & al., 92]. Pour opérer à une telle transformation, le modèle Grafcet initial doit respecter certaines règles: 1) supprimer les étapes d'attente du modèle Grafcet. 2) modifier le marquage initial d'une façon appropriée aux RdP P-temporisés. 3)

Interdire[25] le parallélisme interprété. 4) Etablir un modèle Grafcet connexe. 5) interdire l'utilisation des étapes réinitialisables ($\{vrai \wedge vrai \rightarrow vrai\} \neq \{ 1 + 1 = 2 \}$).

Ces restrictions limitent quelque peu le pouvoir de spécification du formalisme GRAFCET, mais elles restent finalement peu contraignantes (excepté peut-être les impositions de la règle 4). Après transformation, les auteurs nous invitent à effectuer la validation *'par les techniques classiques liées aux RdP'*, puis à évaluer les performances temporelles '*par une temporisation des places, transformant le RdP autonome en RdP P-temporisé...les comportements transitoires pouvant être appréhendés par la simulation*'.

2.3.5.2. Traduction du modèle dans un langage réactif.

La méthode de traduction d'un modèle Grafcet en langage réactif SIGNAL fait l'objet de la thèse de LE PARC [LE PARC Ph., 94]. Schématiquement, pour traduire la structure du modèle Grafcet en langage SIGNAL, l'auteur associe une mémoire à marche prioritaire $S(E_i)$ à chaque étape E_i du modèle Grafcet. L'équation de la mémoire est la suivante :

$$S(E_i) := \{ S(E_i)\$1 \wedge not(Désactive(E_i)) \} \vee Active(E_i)$$

Où, $S(E_i)\$1$, représente la valeur de $S(E_i)$ à l'instant précédent

$Désactive(E_i)$:=*franchissement d'une transition aval de* E_i

$Active(E_i)$:= *franchissement d'une transition amont de* E_i

Le compilateur GRAFCET → SIGNAL écrit par LE PARC permet de représenter l'ensemble des caractéristiques d'un modèle Grafcet – Structure, réceptivités, actions, temporisations, ordres de forçage... - L'interprétation est faite avec ou sans recherche de stabilité. A son tour SIGNAL génère l'automate équivalent sur lequel les outils propres à SIGNAL vérifient les propriétés du modèle (MEC-SIGALI). Cet environnement est également doté d'un simulateur.

La traduction d'un modèle Grafcet vers le langage réactif ESTEREL est décrite dans la Thèse de GAFFE [GAFFE D., 96]. L'emploi du langage ESTEREL[26] a contraint l'auteur à étendre la notion de réceptivité[27] du formalisme GRAFCET. Cette extension du formalisme est

---

[25] L'article auquel nous nous référons a été écrit en 1992. Les auteurs s'appuyaient sur la norme IEC-60848 qui interdit (page 40) le parallélisme interprété. Depuis 1995, la norme NFC 03 190 l'autorise à nouveau. Pour assurer l'équivalence Grafcet→ RdP, il faut donc à nouveau interdire ce type de parallélisme

[26] ESTEREL n'est sensible qu'à l'occurrence d'événements, ce qui signifie que c'est la variation d'une condition (et non pas la condition) qui déclenche une évolution.

[27] Le modèle S-Grafcet utilise la notion '<événement >:<condition>' pour exprimer une réceptivité. L'*événement* garde la transition ou déclenche l'évolution. La *condition* est une expression booléenne classique qui fait intervenir les booléens d'entrées et les variables d'états.

appelée S-GRAFCET (S pour Synchronous). Le compilateur S-GRAFCET→ESTEREL génère l'automate équivalent. Les propriétés peuvent être vérifiées par la recherche des états accessibles et à l'aide de l'outil de preuves spécifique BAC. GAFFE utilise également le principe intéressant des observateurs, particulièrement adaptés à la vérification des propriétés de sûreté. L'observateur associé a son propre modèle Grafcet. Une (ou plusieurs étapes) de l'observateur correspond à une situation interdite. Les résultats de compilation du modèle initial et de l'observateur sont alors regroupés dans un même fichier, ultérieurement analysé par l'outil BAC. Le modèle initial et l'observateur évoluent simultanément. La propriété est vérifiée si les étapes '*interdites*' de l'observateur sont toujours inaccessibles.

2.3.5.3. Traduction du modèle Grafcet à l'aide d'algèbres.

Les travaux de ROUSSEL [ROUSSEL JM., 94] fournissent à partir d'un modèle Grafcet, un automate équivalent réduit, donc plus exploitable. Cette réduction s'effectue d'une part en éliminant les situations instables et d'autre part en analysant la chronologie des entrées de laquelle on déduit les évolutions impossibles (entrées évoluant deux fois de suite dans le même sens, entrées simultanées non corrélées...). L'analyse de la chronologie a nécessité la construction d'une nouvelle algèbre de BOOLE étendue[28], intégrant deux opérateurs unaires associés aux notions de fronts descendants et de fronts montants. La méthode de génération de l'automate et la grammaire ensembliste permettant de spécifier les propriétés à vérifier constituent la plate-forme informatique AGGLAE (Analyse de Grafcet par Génération Logique de l'Automate Equivalent [ROUSSEL JM. & al., 97]). AGGLAE permet la détection des blocages totaux ou partiels, les impossibilités de ré-initialisation, la détection des erreurs dans l'utilisation des actions mémorisées et la détection des émissions simultanées ou séquentielles de certaines sorties.

La Théorie du Signal Hyperfini constitue une seconde approche algébrique basée sur les corps finis d'ordre 2 et sur une modélisation du temps continu à dates discrètes (TSH [LAMPERIERE S., 98]). Cette théorie s'appuie sur les travaux relatifs à l'analyse non standard [FRACHET JP. & al., 92], [FRACHET JP. & al., 96] où l'on considère qu'un signal à une valeur finie (ici 0 ou 1) à une date donnée. LAMPERRIERE montre les aptitudes de la TSH à simplifier et résoudre des équations booléennes par calcul formel. L'auteur définit l'opérateur dérivé à gauche (et à droite) d'un signal binaire scalaire et exprime à l'aide de la dérivée à gauche les fronts montant et descendant d'un signal binaire scalaire (il démontre au passage les propriétés de l'algèbre de BOOLE étendue développée par ROUSSEL

[28] L'auteur démontre 14 propriétés : $(u \vee u\uparrow) = u$, $(u \wedge u\uparrow) = u\uparrow$, $\uparrow(\neg u) = \downarrow u$, $\uparrow(\Pi u_i) = \Sigma(\uparrow ui \,.\, \Pi u_j)$ avec $i \neq j$...

[ROUSSEL JM. & al., 93]). Le modèle Grafcet, vu comme un ensemble de mémoires à marche prioritaire, est traduit par un ensemble d'équations différentielles TSH : le cahier des charges est également traduit par un ensemble d'équations différentielles TSH. Chacun de ces deux ensembles d'équations permet de vérifier respectivement le modèle grafcet et la cohérence du cahier des charges. La validation est obtenue par comparaison de ces deux ensembles d'équations.

2.4. Conclusion partielle et points remarquables.

Nous venons de voir à travers les paragraphes *2.1, 2.2* et *2.3* l'étendue des travaux réalisés par une communauté qui ne se limite plus aux seuls automaticiens. Nous retiendrons de cette étude bibliographique une idée générale sur les formalismes et un ensemble de cinq points remarquables (ce qui ne signifie pas qu'il n'y en ait que cinq).

⇒ Conclusion partielle : nous sommes tentés de noter une *proportionnalité inverse* entre le confort d'utilisation d'un formalisme et sa capacité de preuve. Autant la logique temporelle possède de bonnes aptitudes pour valider les propriétés d'une application, autant la phase de spécification à l'aide de ce formalisme reste *rebutante* pour le non-spécialiste. A l'inverse, le formalisme GRAFCET est apprécié pour son pouvoir d'expression (aspect graphique), mais ses propres outils de validation sont limités à la seule écriture du graphe des situations accessibles (Cf. *Figure 32*).

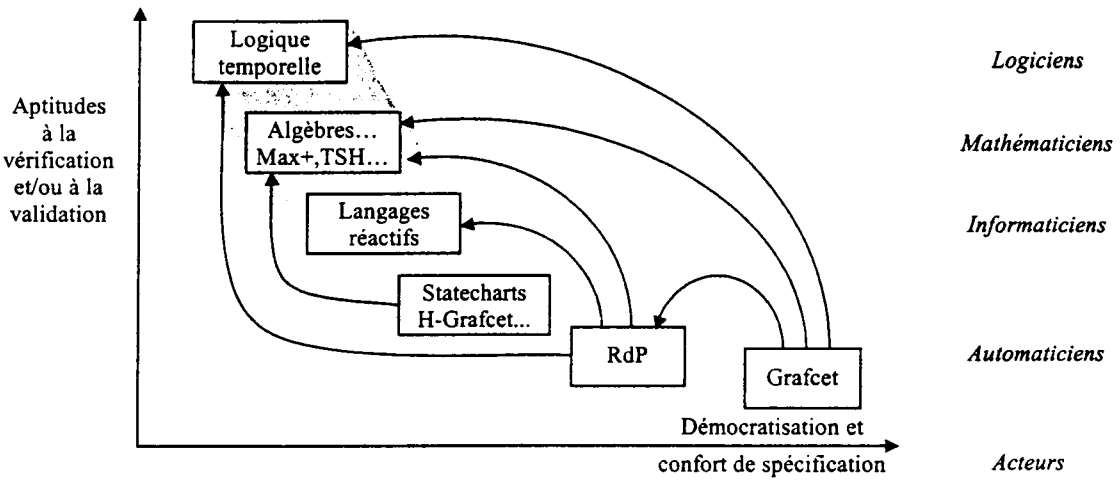


Figure 32 : Dilemme spécification-validation.

La réponse à ce dilemme se trouve en partie dans les nombreuses passerelles établies par la communauté entre les formalismes orientés '*spécifications*' (en particulier, le formalisme GRAFCET) et les formalismes orientés '*vérifications/validations*'.

Comme le rappelle effectivement DENIS [DENIS B. & al., 95], « *aucune méthode ne permet de couvrir tous les aspects du développement d'un SAP* ». Cette remarque justifie l'utilisation de plate-forme multi-modèle admettant le formalisme GRAFCET comme formalisme pivot (Cf. *Figure 33*).

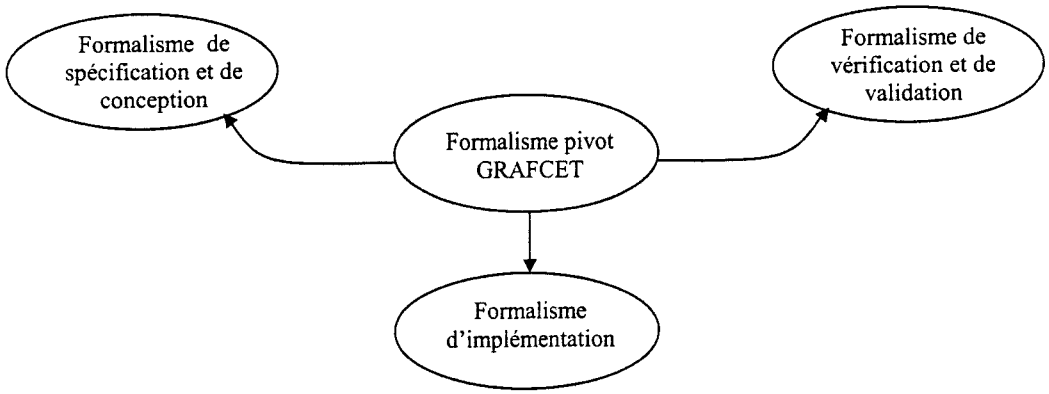


Figure 33 : Plate-forme multi-modèle

Nous observons également à la suite de cette étude, une possibilité de généraliser les principes utilisés par les méthodes de vérification/validation. Le modèle de l'application et le modèle de ses propriétés (à vérifier et/ou à valider) sont exprimés dans le même formalisme. La vérification/validation se fait alors :

- par l'examen de la cohérence des deux modèles (cohérence des deux ensembles d'équations en logique temporelle, cohérence du modèle application et de son observateur...)

ou

- par l'examen de la comparaison du comportement des deux modèles (comparaison de deux ensembles d'équations, comparaison de deux systèmes de transitions...).

⇒ Points remarquables : nous profitons de ce tour d'horizon (non-exhaustif) pour rappeler brièvement les cinq points remarquables qui ont particulièrement attiré notre attention.

1 □ Le concept d'HyperGrafcet de DUMERY au paragraphe *2.1.1.5*: l'encapsulation hiérarchise l'application et incite le concepteur à procéder par affinements successifs. Elle permet en quelque sorte de fixer une granulométrie à l'application étudiée. De plus, l'utilisation d'étiquettes de positionnement, sorte de paramétrage des états initiaux, est une spécification inhabituelle mais puissante.

2 □ Le cycle de vie SPIRALE, proposé par BERGER, paragraphe *1.3.4 : « rien ne sert de tout spécifier et de tout concevoir si une des principales fonctions n'est pas assurée ».* Pourquoi par exemple chercher à valider les propriétés de sûreté et de vivacité d'un système, si le temps de cycle de l'application ne correspond pas au niveau de productivité demandé ?

3 □ La notion d'observateur développée dans GAFFE au paragraphe *2.3.5.2* et celle de superviseur issue des travaux de RAMADGE et WONHAM, paragraphe *2.2.3* : la frontière du système de conduite étant fixée, nous ne pouvons agir sur le monde extérieur qui est du point de vue de la commande incontrôlable. Nous ne pouvons que réagir. L'utilisation d'un observateur, qui surveille le monde extérieur et /ou celle d'un superviseur qui interprète les événements incontrôlables issus de ce monde extérieur constituent une façon efficace de garantir la sécurité de fonctionnement.

4 □ La commande permissive également issue des travaux de RAMADGE et WONHAM dont un exemple est proposé par CHARBONNIER [CHARBONNIER F., 96]. Cette commande s'appuie sur les interdits, mais le complément de ce qui est interdit est plus vaste que ce qui est obligatoire. Il existe donc entre les deux, une marge de manœuvre qui garantit à la fois la sûreté et la vivacité : ce qui *ne doit pas* être fait, ce qui *doit être* fait et ce qui *peut être* fait.

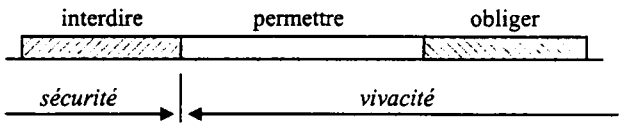


Figure 34 : Commande permissive.

5 □ Les conseils de bon sens énoncés par DAVID et BLANCHARD, paragraphe *2.1.1.4* et l'agrégation réglementée de composantes qui possèdent et conservent les bonnes propriétés, (MARES au paragraphe *2.2.3*) : du fait de leur simplicité, ces deux principes associés conduisent à l'écriture de modèles Grafcet explicites qu'il n'est pratiquement plus nécessaire de vérifier. Dans le même esprit, nous retenons également le parallélisme de type « *fork-join* » retenu par le langage ESTEREL, (GAFFE, paragraphe *2.1.1.1*).

3. Problématique et positionnement.

3.1. Problématique.

Nous évoquons la problématique sous l'angle du *besoin* et de l'évolution des *architectures.*

- Le besoin : la concurrence féroce et la nature évolutive et éphémère des produits fabriqués exigent des systèmes automatisés de production les performances suivantes : *productivité*[29], *flexibilité*[30] [CORIAT B., 91] et *agilité*[31] [BADOT O., 98]. Remarquons que ces exigences traduisent la réactivité du SAP, mais que cette réactivité porte sur des horizons temporels distincts. De plus, ces exigences sont contradictoires : à lui seul, l'objectif QCD, '*Qualité, Coût, Délais*', relève du compromis. En admettant que l'on construise la machine qui produit bien, vite et pour *pas* cher, il faut, en intégrant la flexibilité et l'agilité, que cette machine conserve ces aptitudes pour des produits différents... mais aussi pour de nouveaux produits. L'agilité implique en outre une bonne productivité de l'opération de développement des SAP qui contraint l'entreprise à pratiquer une politique de *capitalisation du savoir-faire.*

- L'architecture : les concepteurs éprouvent de plus en plus de difficultés à adapter un système de commande centralisé aux systèmes qu'ils doivent piloter, parce que la taille et la complexité de ces systèmes ne cessent d'augmenter, mais aussi parce que la durée de vie de ces systèmes, à l'image des produits fabriqués, tend à diminuer. L'offre technologique actuelle ouvre de nouvelles perspectives. La quasi- standardisation des réseaux locaux et des boîtiers d'entrées/sorties associés ont déjà permis de décharger efficacement le système de commande maître de certaines fonctions (diagnostique, sécurité et disponibilité...). Les réseaux locaux industriels autorisent donc en interne une véritable *distribution de l'intelligence.* Cette répartition s'organise autour du concept d'objet[32]. La modularité du concept et le mécanisme d'héritage de classe concourent efficacement à la réutilisation et à la capitalisation du savoir-faire de l'entreprise. Les *architectures distribuées* répondent donc favorablement aux soucis de flexibilité et d'agilité des systèmes automatisés de production.

---

[29] La productivité mesure la réalisation du triple objectif QCD : Qualité, Coût, Délais.

[30] Flexibilité : capacité à effectuer des tâches différentes, à conduire plusieurs machines correspondant à des opérations successives, à gérer le diagnostic, la maintenance ou la qualité. |CORIAT B.,91]

[31] Agilité : capacité à s'adapter à un environnement de plus en plus compétitif, sujet à des changements imprévisibles |BADOT O.,98].

[32] Une application orientée objet se caractérise par la modularité et l'héritage. La modularité, basée sur l'abstraction et l'encapsulation, est la propriété d'un système qui a été décomposé en un ensemble de modules cohérents et reliés entre eux. Un objet appartient à une classe et hérite des caractéristiques et des propriétés de cette classe.

3.2. Positionnement.

Nous avons précisé au paragraphe *1.4* le champ de notre étude, puis au paragraphe précédent la problématique générale de ce champ. Nous définissons à présent le niveau de l'entreprise concerné et ce à quoi nous nous intéressons. Nous donnons les grandes lignes de notre action.

3.2.1. Les niveaux de l'entreprise :

Nous adaptons la classification originelle des CIM (*Computer Integrated Manufacturing*) à une architecture distribuée similaire à l'architecture '*Transparent*[33] *Factory*' proposée par [LETOURMY A., 00]. Ainsi, l'activité économique est vue comme un ensemble d'entreprises, l'entreprise comme un ensemble d'unités dont celles de production, l'unité de production comme un ensemble de machines automatisées, la machine comme un ensemble d'actionneurs et de capteurs...etc.

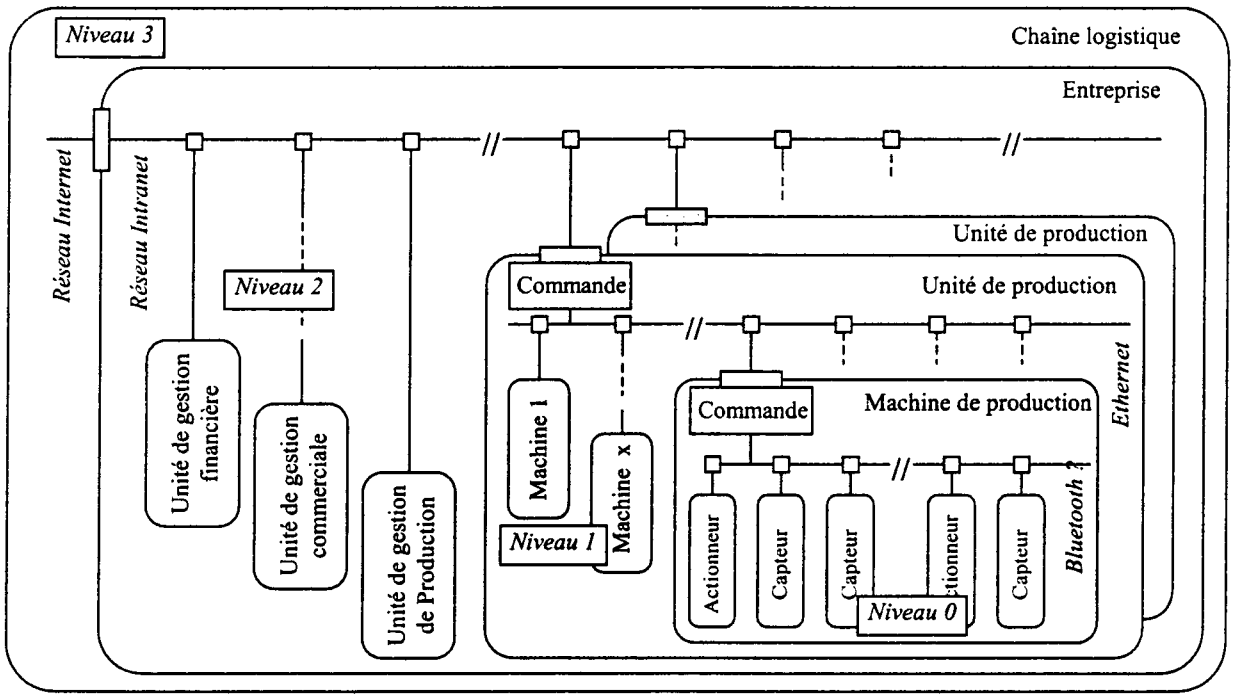


Figure 35 : Imbrication de niveaux distribués.

Nous sommes concernés par le niveau '*unité de production*' et le niveau '*machine de production*'. Ces niveaux *0* & *1* sont abordés indépendamment l'un de l'autre. Dorénavant, lorsque nous parlerons de SAP, nous considérerons *indifféremment* l'un ou l'autre de ces deux niveaux.

[33] Selon l'auteur, « *cette structure permet de mettre à disposition la bonne information, au bon endroit, au bon moment et à toute personne autorisée* ». Transparent Factory est basée sur la standard Ethernet TCP/IP.

3.2.2. Centre d'intérêt : productivité et flexibilité, objectif QCD.

La problématique est d'envergure. Les mots clés sont : *productivité, flexibilité, agilité, ré-utilisation et systèmes distribués*. Nous nous intéressons aux deux premières notions, laissant de côté pour cette étude, l'ensemble homogène[34] que constituent l'agilité, la ré-utilisation et l'organisation distribuée. Ces derniers points font l'objet de travaux d'actualité, entre autres, ceux [CHAPURLAT V. & al., 97] et [BLANC L. & al., 99]. Nous supposerons par la suite que nous sommes dans un environnement distribué où les frontières de l'objet sont connues.

Nous analysons l'impact de la flexibilité sur les facteurs *QCD* qui définissent la productivité. L'objectif est de déterminer s'il y a convergence ou conflit entre ces deux notions.

- *La qualité* : un produit de qualité est conforme en tout point à son propre cahier des charges et le SAP qui le produit doit avoir la capacité de contrôler cette conformité ou mieux, d'anticiper l'instant où cette conformité ne sera plus. La fiabilité intrinsèque du SAP espace ces instants et limite les interventions de maintenance, elles-mêmes déclenchées par les fonctions de diagnostic et/ou de contrôle. Ces notions, associées aux propriétés de disponibilité [35] et de sécurité, forment ce que VILLEMEUR définit comme la sûreté [VILLEMEUR A., 88]. En résumé, la qualité est l'apanage d'un système de production sûr, c'est-à-dire fiable, surveillé, maintenu, disponible donc vivace, et sécurisé. La qualité est difficile à évaluer : en pratique on mesure plutôt la non-qualité par les coûts qu'elle induit (chiffrage des rebuts, répercussion financière de l'allongement des délais).

- *Les délais* : les délais sont liés d'une part à la fiabilité du SAP (pas ou peu retard dû à des dysfonctionnements), à la rapidité d'exécution de l'opération de production (temps de cycle, célérité) mais aussi à sa flexibilité. La flexibilité, permet instantanément de changer de type de production, de réduire la taille des lots (et au passage les stocks) et donc de répondre aux demandes les plus diverses dans un temps très court.

- *Les coûts* : Ils sont la résultante des deux premiers facteurs mais dépendent aussi directement de la célérité du SAP. Ils baissent avec la taille des stocks (liés à la flexibilité) et augmentent lorsque le taux d'engagement machine diminue (Return On Investment).

---

[34] Nous le qualifions d'homogène, car l'agilité et la ré-utilisation sont favorisées par la définition des modules inhérente à une architecture distribuée.

[35] Aptitude d'une entité à accomplir une fonction requise dans des conditions données, à un instant donné ou pendant un intervalle de temps donné, en supposant que la fourniture des moyens extérieurs nécessaires soit assurée. Norme française X 60-500, octobre 1988. (côté système, la disponibilité s'appuie donc sur la propriété de vivacité).

A priori, La flexibilité participe aux mêmes objectifs que la productivité : La capacité à effectuer des tâches différentes agit avantageusement sur les *délais* et celle à gérer le diagnostic, la maintenance ne peut que favoriser la *qualité* : les deux aptitudes réunies participent à l'abaissement indirect des *coûts*. Cependant, la constitution du *coût* direct pose problème : Peut-on concevoir un SAP unique qui satisfasse à la fois la *réduction des stocks* et *l'augmentation du taux d'engagement*, tout en étant capable d'*effectuer plusieurs tâches distinctes* ?

Une autre façon de poser la question est de savoir si un SAP unique peut être aussi performante dans l'exécution des X fonctions que réaliseraient que X SAP dédiés. Si oui, sa flexibilité permet alors de réduire les stocks tout en possédant un taux d'engagement optimal.

C'est à cette dernière question que nous allons tenter de répondre.

3.2.3. Les grandes lignes de notre action.

- *Sur quoi pouvons nous agir ?*

Un SAP est constitué de ressources physiques. La réactivité imposée par la flexibilité ne nous permet d'intervenir que sur l'organisation[36] logique de ces ressources, c'est-à-dire sur l'ordre suivant lequel les différentes tâches sont allouées à ces ressources. Cette problématique est très proche de celle des ordonnanceurs et plus particulièrement des ordonnanceurs cycliques : comment et quand attribuer les tâches aux ressources pour assurer un *débit* de production maxi, à *encours* mini et pour un *plan de charge donné* ? Dans ce cadre, les opérations de transformation et/ou d'assemblage apportées par la machine ou par le système à un produit sont décrites par une gamme de fabrication propre à ce produit. Cette gamme regroupe l'ensemble des contraintes temporelles, des contraintes de précédences des différentes tâches et les possibilités d'allocation de ces tâches aux ressources potentielles.

Nous proposons donc de rendre un SAP flexible, en calculant et en implémentant une commande pour chaque nouvelle gamme qu'il est sensé réaliser. Notre proposition est en quelque sorte, une adaptation '*software*' du SAP au produit à fabriquer. Remarquons que nous agissons uniquement sur la célérité de la machine ou du système, qui nous l'avons vu, a des implications directes sur les coûts et les délais. La réalisation de la sûreté globale (vivacité, sécurité...) sera confiée à des observateurs dédiés soit au SAP, soit aux gammes 'produit '.

[36] Notre champ d'étude est limité à la commande, nous n'envisageons pas (pour le moment), une ré-organisation physique de la machine (machine modulable-RNUR) ou une ré-organisation physique de l'atelier (îlot re-configurable ...etc.)

➢ *Comment procédons-nous ?*

Nous adoptons le formalisme GRAFCET comme formalisme de modélisation des systèmes de commande : il est le pivot central d'une plate-forme multi-modèle et il est normalisé.
Nous répondons à la flexibilité, non pas par un Grafcet de commande unique et correspondant à de multiples gammes, mais par un choix multiple de Grafcets élémentaires, chacun d'eux correspondant à une gamme unique : La démarche est résumée par l'équivalence suivante :

$$Grafcet\{\cup choix\} \Leftrightarrow \cup Grafcet\{choix\}$$

Chaque Grafcet élémentaire est dédié à une gamme. En terme de célérité, l'organisation des tâches peut donc être optimale pour cette gamme, ce qui n'est pas forcément le cas pour une organisation unique chargée d'un ensemble de gammes.

La conception d'un SAP *flexible* et *productif* revient donc à définir pour chaque gamme 'produit', le modèle Grafcet associé. Les premiers effets de cette modularité se traduisent par la possibilité d'aborder des problèmes de taille réduite, ce qui facilite les phases d'étude, de validation et de mise au point. En second lieu, l'adjonction, la suppression ou la modification d'un produit correspond à l'adjonction, la suppression ou la modification de son unique modèle Grafcet, sans retoucher a priori le reste de l'application. Ce deuxième point confère à la méthode de bonnes aptitudes à la re-conception et à la ré-utilisation.

Nous schématisons sur la *Figure 36* les différentes étapes qui pour une gamme opératoire donnée permettent d'obtenir son Grafcet de commande.

- Un premier module de traitement réalise l'ordonnancement cyclique de la gamme opératoire. La méthode d'ordonnancement cyclique que nous avons définie est adaptée à notre contexte : elle est notamment capable de traiter des gammes[37] de fabrication non linéaires. Le résultat de cet ordonnancement est donné sous la forme d'un diagramme de GANTT : l'ordonnancement est optimal[38] dans le sens où nous cherchons à saturer la ressource critique utilisée par le SAP (système ou machine).

- Un second module extrait systématiquement du diagramme de GANTT un graphe d'événements dépourvu de choix et de conflit. Le formalisme de ce graphe est issu du formalisme de COMMONER [COMMONER F., 71].

[37] Au sens graphique, une gamme non linéaire intègre des opérations d'assemblages ou de désassemblage.

[38] Le deuxième critère d'optimalité qui porte sur le nombre d'encours minimal n'est pas maîtrisé.

- Le module numéro *3* est chargé de construire le Grafcet de commande à partir du graphe d'événements précédent. Cette opération est également systématique. Elle peut être conduite dans l'autre sens, ce qui est favorable à l'activité de re-conception.

- Le dernier module (n°*4*) exploite le graphe d'événements. Il permet d'étudier l'architecture (connexité, circuits, isthmes, points d'articulation...) et de calculer le temps de cycle d'une application (célérité). Ce module vérifie également la vivacité du modèle (graphe vivant et sauf).

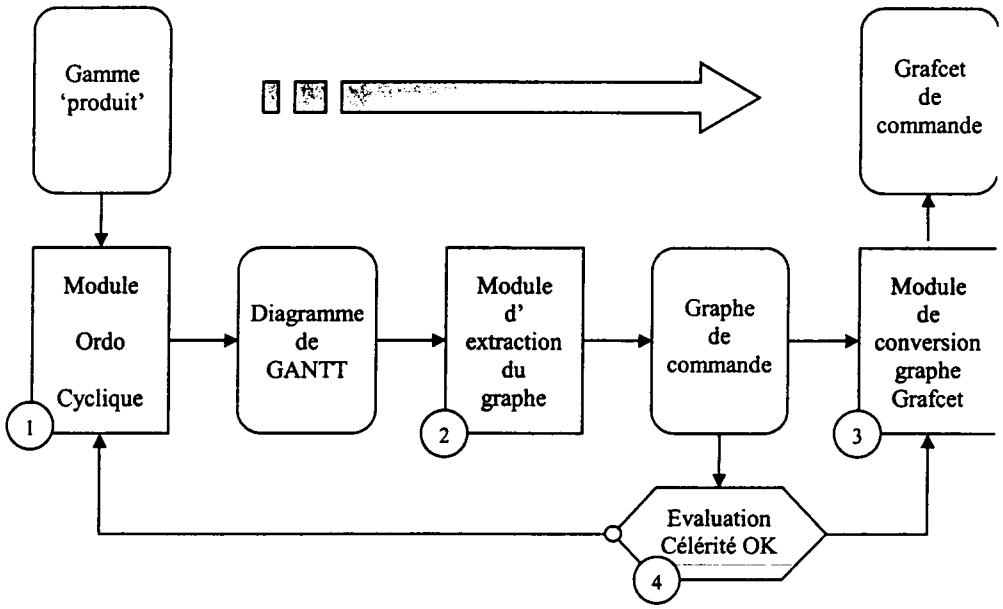


Figure 36 : De la gamme 'produit ' au Grafcet de commande.

Ces quatre modules donnent lieu au développement d'applications informatiques spécifiques. Le module *1* réalise l'ordonnancement cyclique en utilisant un algorithme qui s'appuie sur le comportement social des colonies de fourmis (*Ant system*). Le module *2* nécessite la définition d'un formalisme basé sur les graphes, formalisme sur lequel travaillent les modules *3* & *4*. Ces deux derniers modules exploitent les résultats de la Théorie des graphes.

➢ *Quelle est la ré-utilisation de notre 'étude bibliographique ?*

Notre démarche repose en partie sur l'étude bibliographique synthétisée au paragraphe *2.4*.

- Le choix d'une plate-forme multi-modèle admettant le formalisme GRAFCET comme pivot central, donne accès à l'ensemble des travaux axés sur la vérification et la validation des modèles Grafcet (idée générale § *2.4*).

- Les concepts d'encapsulation (1^{ier} point remarquable § *2.4*) autorise la construction d'une commande modulaire où chaque entité peut-être décrite simplement.

- La philosophie du cycle de vie SPIRALE (2ième point remarquable § *2.4*) nous guide dans le choix de la célérité comme première propriété à vérifier.

- L'utilisation d'observateurs pour vérifier la propriété de sécurité (3ième point remarquable § *2.4*) peut-être étendue à la modélisation d'autres fonctions (surveillance, diagnostic...)

- La notion de commande permissive (4ième point remarquable § *2.4*) nous encourage dans la recherche d'une nouvelle méthode de conception, basée en particulier sur les phénomènes aléatoires.

- Enfin, nous gardons à l'esprit les conseils de bon sens donnés par DAVID et BLANCHARD (5ième point remarquable § *2.4*) - la simplicité avant tout.

4. Conclusion.

Au terme de cette première partie nous dressons un premier bilan de notre activité :

- La communauté des personnes concernées par les SAP n'est plus limitée aux seuls automaticiens. Les travaux qui en découlent sont très nombreux et la littérature du domaine est abondante. L'informatique fondamentale et ses applications prennent une place importante dans ce secteur en s'appuyant sur des modèles mathématiques.

- Dans ce contexte, nous avons localisé notre champ d'étude à la partie commande des systèmes à événements discrets, réactifs, synchrones et automatisés de production. Nous considérons indifféremment et indépendamment les niveaux *0* et *1* de la classification CIM, regroupant ainsi sous l'appellation générique SAP, aussi bien une machine automatisée qu'un atelier de production automatisé.

- La problématique générale touche les performances et l'architecture des SAP : productivité, flexibilité, agilité, ré-utilisation et distribution de la décision. Dans l'hypothèse d'un environnement réparti, nous limitons cette problématique aux aspects de productivité et de flexibilité.

Nous organisons notre d'étude en quatre étapes :

1) Définition d'un formalisme graphique.
2) Développement d'outils d'analyse des graphes.
3) Elaboration d'une méthode de conception basée sur l'ordonnancement cyclique.
4) Application des outils et méthodes à une étude de cas.

5. Bibliographie.

[AFCET, 77] AFCET, 77 :*"Normalisation de la représentation du cahier des charges."*. Automatique et Informatique Industrielle., Vol Novembre - décembre 1977.,

[ALLA H. & DAVID R., 98] ALLA H. & DAVID R., 98 : *"Continuous and hybrid PETRI nets"*. Journal of circuits and computers, Vol 8, n°1.

[ARNOLD A., 92] ARNOLD A., 92: *"Système de transitions finis et sémantique des processus communicants."*. MASSON.

[AYGALINC P. & DENAT J-P., 92] AYGALINC P. & DENAT J-P., 92: *"Validation de modèles GRAFCET fonctionnels et évaluation des performances du système associé par l'utilisation des réseaux de PETRI."*. GRAFCET'92, PARIS, 25-26 mars 1992.

[BADOT O., 98] BADOT O., 98: *"Théorie de l'entreprise agile"*. L'harmattan Editeur - PARIS.

[BERGER Ch., 99] BERGER Ch., 99: *"Spécifier, c'est programmer."*. *Nouvelles percées dans les langages pour l'automatique.*, AMIENS, 25 novembre 1999.

[BERRY G., 99] BERRY G., 99: *"The ESTEREL V5 Language Primer. Version 5.21 release 2.0"*. Centre de mathématiques appliqués - Ecole des mines and INRIA - SOPHIA ANTIPOLIS.

[BLANC L., PERALDI-FRATI M-A. & ANDRE C., 99] BLANC L., PERALDI-FRATI M-A. & ANDRE C., 99: *"R-PROTS: une plate-forme d'aide à la conception d'applications réparties."*. MSR'99, ENS CACHAN,

[BLANCHARD M., 79] BLANCHARD M., 79: *"Comprendre et appliquer le Grafcet"*. CEPADUES.

[BOUSSEAU F., 97] BOUSSEAU F., 97: *"Modélisation des systèmes complexes : Une approche par les réseaux de PETRI"*. Thèse de *Doctorat, 254, Ecole doctorale de NANTES - ISTIA, ANGERS.*

[BOYER H., NORBERT M. & PHILIPPE R., 76] BOYER H., NORBERT M. & PHILIPPE R., 76: *"Construction Electrique - Tome 6. 3ième édition"*. Editions de la CAPITELLE.

[BRAMS GW. & ouvrage collectif., 83] BRAMS GW. & ouvrage collectif., 83: *"Réseaux de PETRI: Théorie et pratique - Tome 1 : Théorie et analyse"*. MASSON.

[BRAMS WG. & ouvrage collectif., 83] BRAMS WG. & ouvrage collectif., 83: *"Réseaux de PETRI: Théorie et pratique - Tome 2 : Modélisation et applications"*. MASSON.

[CEI-848, 88] CEI-848, 88: *"CEI 848 - Etablissement de diagrammes fonctionnel pour système de commande"*. GENEVE IEC Editions, Suisse.

[CHAPURLAT V. & PRUNET F., 97] CHAPURLAT V. & PRUNET F., 97 : *"Un modèle pour la spécification et la validation des systèmes de contrôle/commande répartie: le modèle ASCY-R".* RAIRO-APII-JESA, Vol 30, n°1.

[CHARBONNIER F., 96] CHARBONNIER F., 96: *"Commande supervisée des systèmes à événement discrets".* Thèse de *Doctorat, Institut National Polytechnique de GRENOBLE.*

[CHARBONNIER F., ALLA H. & DAVID R., 99] CHARBONNIER F., ALLA H. & DAVID R., 99: *"The supervisor control of discrete event dynamics systems".* IEEE, Vol 7, n°2.

[COMMONER F., 71] COMMONER F., 71 :*"Marked Directed Graphs".* Journal of Computer and System Science, Vol 5,

[CORIAT B., 91] CORIAT B., 91: *"Penser à l'envers. Travail et organisation dans l'entreprise japonaise."* Christian BOURGOIS Editeur, Paris, 1991, 186p

[COUFFIN F., LAMPERIERE S., HOSPITAL E. & FAURE JM., 00] COUFFIN F., LAMPERIERE S., HOSPITAL E. & FAURE JM., 00 :*"Réutilisation d'un modèle de référence en génie automatique. Application à l'intégration des activités de conception des SAP.".* JESA, Vol 34, n°1.

[DAVID R. & ALLA H., 89] DAVID R. & ALLA H., 89: *"Du Grafcet aux réseaux de PETRI".* HERMES.

[DENIS B., 94] DENIS B., 94: *"Assistance à la conception et à l'évaluation de l'architecture de conduite des systèmes de production complexes".* Thèse de *Doctorat, Ecole doctorale d'informatique, automatique, électronique et mathématiques de l'Université de Nancy 1, NANCY.*

[DENIS B. & LESAGE JJ., 95] DENIS B. & LESAGE JJ., 95: *"Un panorama de la recherche en conception de la conduite des systèmes de production". Congrès international de Génie Industriel de Montréal, MONTREAL,* 7 & 9 octobre 1995.

[DERAIL Y., 84] DERAIL Y., 84: *"La commande séquentielle répartie - Synthèse du Grafcet - Utilisation des réseaux locaux".* Thèse de *Docteur Ingénieur, Institut National Polytechnique de GRENOBLE.*

[DUMERY JJ., 99] DUMERY JJ., 99: *"Un langage de spécification pour la conception structurée de la commande des systèmes à événements discrets.".* Thèse de Doctorat, Ecole Centrale de Paris - CESTI / ISMCM, PARIS.

[FRACHET JP. & COLOMBANI G., 92] FRACHET JP. & COLOMBANI G., 92: *"Eléments pour une sémantique temporelle GRAFCET et des systèmes dynamiques fondée sur une analyse non standard". GRAFCET'92 - AFCET, PARIS, 25 & 26 mars 1992.*

[FRACHET JP., LAMPERIERE S. & FAURE JM., 96] FRACHET JP., LAMPERIERE S. & FAURE JM., 96: *"Le signal Hyperfini : Application à la modélisation du comportement des systèmes à événements discrets"*. MSR'96, BREST, 28 & 29 mars 1996.

[FROMENT B., 93] FROMENT B., 93: *"Structural Design with Grafcet :Towards a global method"*. Automatic Control Production Systems Review AFCET HERMES, Vol 27, n°1.

[GAFFE D., 96] GAFFE D., 96: *"Le modèle Grafcet: Réflexion et intégration dans une plate-forme multiformalisme synchrone."*. Thèse de Doctorat, Université de NICE-Sophia Antipolis, NICE.

[GONTHIER G., 88] GONTHIER G., 88: *"Sémantiques et modèles d'exécution des langages réactifs synchrones. Application à ESTEREL"*. Thèse de *Doctorat*, Université de Paris Sud, ORSAY.

[GREPA, 91] GREPA, 91: *"Le Grafcet: Nouveaux concepts"*. CEPADUES.

[HAINQUE O., 98] HAINQUE O., 98 : *"Présentation de ESTEREL, un langage synchrone pour la spécification et le développement des systèmes réactifs."*. Réseaux - ESI, Vol 22,

[HAREL D., 87] HAREL D., 87: *"Statechart: a visual formalism for complex systems."*. Science of computer programming, Vol 8,

[JULIA S., 97] JULIA S., 97: *"Conception et pilotage de cellules flexibles à fonctionnement répétitif modélisées par Réseaux de Petri."*. Thèse de *Doctorat, LAAS-CNRS, TOULOUSE.*

[KORBAA O., 98] KORBAA O., 98: *"Commande cyclique des systèmes flexibles de production manufacturière à l'aide des réseaux de Pétri: De la planification à l'ordonnancement des régimes transitoires."*. Thèse de *Doctorat*, 2295, *Ecole Centrale de LILLE - URA CNRS D1440, LAIL - LILLE.*

[LAMPERIERE S., 98] LAMPERIERE S., 98: *"De la vérification de cahier des charges des systèmes à événements discrets à la validation des spécifications décrites en GRAFCET"*. Thèse de *Doctorat, Ecole Normale Supérieur de CACHAN, CACHAN.*

[LE GUERNIC P., GAUTIER T., LE BORGNE M. & LE MAIRE C., 91] LE GUERNIC P., GAUTIER T., LE BORGNE M. & LE MAIRE C., 91 : *"Programming real time applications with signal."*. Procedings of the IEEE, Vol

[LE PARC Ph., 94] LE PARC Ph., 94: *"Apports de la méthodologie synchrone pour la définition et l'utilisation du langage Grafcet."*. Thèse de Doctorat, 1024, Université de RENNES 1 - Mention informatique - IFSIC, RENNES.

[LETOURMY A., 00] LETOURMY A., 00 :*"Architectures d'automates industriels: évolutions et grandes tendances."*. SCHNEIDER ELECTRIQUE SA., Vol

[L'HER D., 97] L'HER D., 97:*"Modélisation du Grafcet temporisé et vérification des propriétés temporelles."*. **Thèse** de Doctorat, 1780, Université de RENNES 1 - Mention informatique - IFSIC, RENNES.

[LOPEZ P. & ROUBELLAT F., 01] LOPEZ P. & ROUBELLAT F., 01: *"Ordonnancement de la Production."*. HERMES Science, Publication.

[MARES M., 97] MARES M., 97: *"Sur la modélisation et l'analyse des systèmes complexes à événements discrets par réseaux de PETRI"*. Thèse de *Doctorat, 225, Ecole doctorale des Sciences de l'ingénieur de NANTES, LISA - ANGERS.*

[MARES M. & FERRIER JL., 94] MARES M. & FERRIER JL., 94 : *"Réseaux de PETRI et algèbre (max,+) : Deux approches pour l étude des systèmes à événements discrets"*. APII, Vol 28, n°4.

[MENGUY E., 97] MENGUY E., 97: *"Contribution à la commande des systèmes linéaires dans les dioïdes"*. Thèse de *Doctorat, 431, Ecole doctorale des Sciences de l'ingénieur de NANTES, LISA - ANGERS.*

[MERLAUD CH1., 84] MERLAUD CH1., 84 : *"Une méthodologie d'analyse descendante appliquée à la description du fonctionnement des systèmes automatisés. Partie 4."* TECHNOLOGIE et FORMATION, Vol 1,

[MEYER B. & BAUDOIN C., 84] MEYER B. & BAUDOIN C., 84: *"Méthodes de programmation"*. EYROLLES.

[MICHEL G., 88] MICHEL G., 88: *"Les API. Architecture et applications des automates programmables industriels."*. DUNOD - BORDAS.

[NASLIN P., 70] NASLIN P., 70: *"Circuits logiques et automatismes à séquences."*. DUNOD.

[NOURY P., 99] NOURY P., 99: *"Généralisation des blocs fonctionnels dans un environnement distribué."*. *Nouvelles percées dans les langages pour l'automatique.*, AMIENS, 25 novembre 1999.

[NOYES D., 90] NOYES D., 90 :*"Les réseaux de PETRI et la modélisation modulaire de production"*. RAIRO APII JESA, Vol 24, n°6.

[PATANKAR A.K. & ADIGA S., 95] PATANKAR A.K. & ADIGA S., 95 :*"Enterprise integration modeling: a review of theory and practice."*. CIM Systems, Vol 8, n°1.

[PEREZ JP., 90] PEREZ JP., 90: *"Système temps réel, méthode de spécification et de conception"*. DUNOD informatique industrielle - BORDAS.

[PETRI C.A., 62] PETRI C.A., 62: *"Kommunikation mit Automaten, Schriften des rheinisch."*. Westfalischen Institut fur Instrumentelle Mathematik and der Universität Bonn.

[PROTH J.-M. & XIE X., 95] PROTH J.-M. & XIE X., 95: *"Les réseaux de PETRI pour la conception et la gestion des systèmes de production."*. MASSON.

[RAMADGE J. & WOHNAM W., 89] RAMADGE J. & WOHNAM W., 89 :*"The control of discret event system"*. IEEE Procedings, Vol 77, n°1.

[ROUSSEL JM., 94] ROUSSEL JM., 94: *"Analyse de Grafcet par génération logique de l'automate équivalent."*. Thèse de *Doctorat, Ecole doctorale de l'ENS CACHAN - LURPA.*

[ROUSSEL JM. & LESAGE JJ., 93] ROUSSEL JM. & LESAGE JJ., 93 : *"Une algèbre de Boole pour l'approche événementielle des systèmes logiques"*. APII, Vol 37, n°5.

[ROUSSEL JM. & LESAGE JJ., 97] ROUSSEL JM. & LESAGE JJ., 97: *"AGGLAE : Un outil d'aide à la validation des Grafcets de spécification"*. ALISA'97, NANCY, 25 & 26 mars 1997.

[SHINGO S., 86] SHINGO S., 86: *"Maîtrise de la production et méthode KANBAN"*. Les éditions d'organisation.

[SUTRE G., 99] SUTRE G., 99: *"Vérification des automates à file réactifs: un modèle pour les systèmes réactifs écrits en ELECTRE."*. MSR'99, CACHAN.,

[TAILLARD P., 91] TAILLARD P., 91 : *"Description fonctionnelle par Grafcet : Méthodologie d'analyse globale"*. TECHNOLOGY et FORMATION, Vol 41,

[UTE93, 93] UTE93, 93: *"NF-C03-191 - Etablissement de diagrammes fonctionnel pour systèmes de commande, diagramme fonctionnel Grafcet- Extension des concepts de bases"*. Union Technique de l'électricité - juin 1993,

[UTE95, 95] UTE95, 95: *"NF-C03-190 - Diagramme fonctionnel " GRAFCET " pour la description des systèmes logiques de commande"*. UTE - Sept 1995,

[VELU J., 99] VELU J., 99: *"Méthodes mathématiques pour l'informatique"*. DUNOD 3ème édition.

[VERILOG, 97] VERILOG, 97: *"Lustre language reference manual Version 3,3+,4,5"*. VERILOG.

[VIDAL-NAQUET G. & CHOQUET-GENIET A., 92] VIDAL-NAQUET G. & CHOQUET-GENIET A., 92: *"Réseaux de PETRI et systèmes parallèles."*. ARMAND COLIN.

[VILLEMEUR A., 88] VILLEMEUR A., 88 : *"Sûreté de fonctionnement des systèmes industriels. Fiabilité - Facteurs humains - Informatisation."*. Edition EYROLLES Collection de la Direction des Etudes et Recherches d'Electricité de France., Vol

[ZAYTOON J., LESAGE JJ., FAURE JM. & LHOSTE P., 97] ZAYTOON J., LESAGE JJ., FAURE JM. & LHOSTE P., 97 : *"Vérification et validation du Grafcet"*. RAIRO-APPI-JESA, Vol 31, n°4.

Deuxième partie : Optimisation de la productivité des Systèmes Automatisés de Production.

Table des matières.

1. Introduction.

Nous rappelons que notre objectif est d'améliorer la *productivité* et la *flexibilité* des systèmes automatisés de production. Notre contribution porte sur la phase de conception ou de re-conception des SAP, phase volontairement restreinte à la vérification des propriétés temporelles de multiples architectures de commande. Chacune de ces commandes est dédiée à la réalisation d'une production unique et le système est rendu flexible par la multiplicité de ces commandes. Nous limitons l'étude à la première boucle du cycle de vie SPIRALE où nous évaluons la célérité de la commande dédiée à chaque production. Nous examinons en premier lieu quels sont nos moyens d'investigation et quels peuvent être les critères d'optimisation retenus.

➢ *Formalisme, théorie des graphes et ordonnancement cyclique*

Une machine automatisée ou un atelier est constitué de ressources physiques. Nous n'intervenons que sur l'organisation logique de ces ressources, c'est-à-dire sur l'agencement des tâches ou sur l'ordre suivant lequel les différentes tâches sont affectées sur ces ressources.

- La façon la plus simple et la plus explicite de représenter cet agencement est une représentation graphique, au sens de la définition mathématique des graphes : nous donnons à ce graphe une interprétation adaptée à notre domaine : le graphe et son interprétation constituent le formalisme qui bénéficie de l'ensemble des travaux et outils développés dans le cadre de la théorie des graphes.

- Enfin, l'ordre suivant lequel sont établies les affectations tâche-ressource relève de la problématique de l'ordonnancement, qui de plus est cyclique, si la machine ou l'atelier exécute des tâches répétitives.

➢ *Optimisation*

- Qui dit '*optimisation*', dit '*critère*'. Selon notre objectif, le premier critère que nous considérons est temporel : il concerne la mesure du temps de cycle d'une application donnée et il reflète l'aspect productivité de cette application. Le second critère concerne le nombre d'encours et la complexité du graphe de commande en dépend directement (cf. paragraphe *4.5*). Ces deux critères sont mesurables, mais le second est une conséquence de l'agencement obtenu en considérant le premier.

- Le troisième critère caractérisant la flexibilité est plus informel : il porte sur l'ensemble des commandes mises en jeu. Pour juger d'une bonne organisation, il faut juger les résultats du système de production (triple objectif QCD – Qualité, Coût, Délais) sur un horizon qui met en oeuvre l'exécution de l'ensemble de ces commandes. Ce critère est

difficilement mesurable car il est lié à la diversité possible des scénarii de production. De plus, ce critère est sûrement corrélé aux deux premiers.

➢ *Intégration dans une plate-forme multi-modèle.*

L'introduction d'un nouveau formalisme doit se faire en considérant les formalismes existants et leurs aptitudes (spécification, preuves, implémentation). Nous avons évoqué l'intérêt d'une plate-forme multi-modèle admettant le formalisme GRAFCET comme formalisme pivot (Cf. *Figure 33 – partie 1*). Nous y adjoignons donc le formalisme graphique que nous appelons GMT pour *Graphe Marqué* et *Temporisé.*

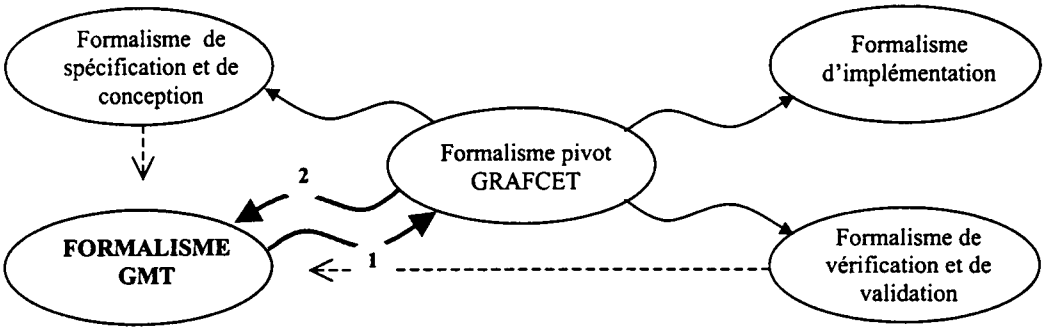


Figure 1 : Intégration du formalisme dans la plate-forme multi-modèle.

L'intégration de ce formalisme dans la plate-forme nécessite la création d'une procédure de traduction du modèle graphique dans le formalisme GRAFCET (repérée *1* sur la *Figure 1*). Ainsi, une application décrite dans le formalisme GMT, conçue et analysée respectivement à l'aide de l'ordonnancement cyclique et des outils de la théorie des graphes, trouve directement son équivalent dans le formalisme GRAFCET.

A l'inverse, la procédure repérée *2* sur la *Figure 1,* favorise les activités de re-conception. Une application existante est traduite dans le formalisme GMT, revue et analysée dans ce formalisme, puis traduite à nouveau dans le formalisme GRAFCET de départ.

Bien que nous ayons défini ces procédures pour des graphes de type exclusivement d'événements ou d'états (Cf. annexe *1*), la nature des graphes obtenus au paragraphe *4.5* nous a conduit à limiter ces procédures à la traduction de graphes de type événements.

Nous allons à présent définir plus précisément :

- le formalisme retenu et son intégration (chapitre *2*),
- les outils d'analyse issus de la théorie des graphes (chapitre *3*)
- la méthode d'ordonnancement cyclique et d'optimisation développée (chapitre *4*)

L'aptitude de ces outils à concevoir une commande de productivité optimale est finalement présentée sur deux exemples d'application complémentaires (chapitre *5*).

2. Le formalisme GMT et sa passerelle.

2.1. Le formalisme GMT, origine et définition.

Le formalisme graphique proposé est issu du formalisme décrit dans [COMMONER F., 71] . Dans ce formalisme, un graphe fini et orienté G est défini mathématiquement par le couple $G(V,E)$ où V représente l'ensemble fini des sommets et E l'ensemble fini des arcs. La notation $a \xrightarrow{e} b$ indique que l'arc $e \in E$ sort du sommet $a \in V$ et entre dans le sommet $b \in V$.
Nous adaptons ces graphes 'mathématiques' à nos besoins de spécification en associant une tâche à chaque nœud du graphe et en introduisant deux notions : le marquage et la temporisation. Le marquage représente l'activité de la tâche et la temporisation la durée de son exécution.

➢ *Définition du formalisme GMT.*

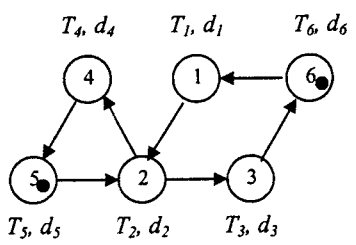
Un graphe du formalisme GMT (Cf. *Figure 2*) est un graphe d'événements[1] représentant les liens logiques d'antériorité existant dans l'exécution des différentes tâches d'une application de production.
Nous le définissons par le quadruplet $< V, E, T, M_0 >$.

- $V = \{ v_1, v_2, \ldots v_n \}$ est l'ensemble des nœuds ou sommets v_i de cardinal n, $i \in (1..n)$

- $E = \{ e_1, e_2, \ldots e_m \}$ est l'ensemble des arcs orientés e_k tels que $e_k = g(v_i, v_j)$: l'arc e_k a pour origine le sommet v_i et pour extrémité le sommet v_j, g est une application de $V*V$ dans E. Ce graphe est représentable par une matrice d'antériorité notée A, telle qu'un élément a_{ij} de A est égal à *1* si l'arc $v_j \xrightarrow{et} v_i$ existe et à *0* dans le cas contraire.

- T est l'ensemble des couples (T_j, d_j) où T_j est une tâche de durée d_j. Le couple (T_j, d_j) est associé au sommet v_j, d_j ou $d(v_j)$ est un réel positif ou nul.

- M_0 est un vecteur de dimension n représentant le marquage initial et la coordonnée $M_0(v_i)$ est un entier positif ou nul représentant le marquage initial du sommet v_i.



A	v1	v2	v3	v4	v5	v6
v1	0	0	0	0	0	1
v2	1	0	0	0	1	0
v3	0	1	0	0	0	0
v4	0	1	0	0	0	0
v5	0	0	0	1	0	0
v6	0	0	1	0	0	0

T	Ti	di
v1	T1	d1
v2	T2	d2
v3	T3	d3
v4	T4	d4
v5	T5	d5
v6	T6	d6

Mo	
v1	0
v2	0
v3	0
v4	0
v5	1
v6	1

Figure 2 : Exemple d'un graphe GMT et de sa matrice d'antériorité.

[1] Nous justifions cette limitation à des graphes d'événements au paragraphe *4.5.*

2.2. Interprétation et héritage.

➢ *Interprétation du marquage.*

Le marquage est destiné à suivre l'évolution du graphe. Dans la définition proposée par [COMMONER F., 71], un nombre de jetons $M(e) \in N$ (entier ≥ 0) est associé à chaque arc. Le marquage, c'est-à-dire l'état du graphe, est porté par les arcs ; le tirage des sommets permet de faire évoluer ce marquage, donc l'état du graphe, et les règles de mise à feu imposent de gérer à la fois les arcs et les sommets.

Dans notre formalisme, *les arcs ne représentent que des liens statiques d'antériorité entre sommets,* c'est pourquoi nous avons déporté le marquage sur les sommets, en associant à chaque sommet v un nombre de jetons $M(v) \in N$ (entier ≥ 0). L'interprétation de ce nouveau marquage est possible si le type de graphes représentés est réduit au type graphe d'événements *ou exclusivement* au type graphe d'états. Un graphe de type événements (respectivement un graphe de type états) admet dans le formalisme de PETRI la représentation donnée *Figure 3a* (respectivement *Figure 3b*). Nous ne considérons par la suite que *l'interprétation en graphe d'événements*.

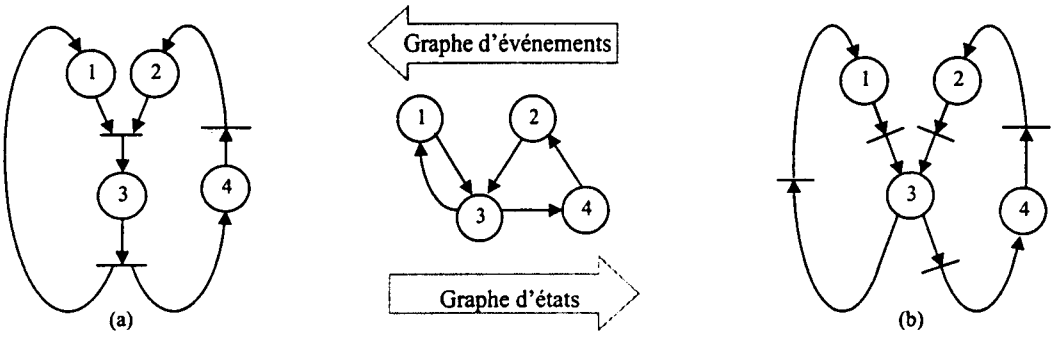


Figure 3 : Interprétation du graphe en graphe d'événements ou en graphe d'états.

Cette interprétation est cohérente avec la définition de la mise à feu d'un sommet tirable donnée par [COMMONER F., 71]. Cette mise à feu se traduit alors dans notre formalisme, par deux définitions et deux opérations d'activation et de désactivation. Le mécanisme d'évolution du marquage est décrit Figure 4.

- Un sommet est dit activable si le marquage de chacun des sommets directement en amont est strictement positif.

- Un sommet est dit désactivable, si son marquage est strictement positif.

- L'activation d'un sommet activable consiste à retirer un jeton de chacun des sommets directement en amont et à ajouter un jeton à ce sommet.

- La désactivation[2] d'un sommet désactivable consiste à retirer un jeton de ce sommet et à ajouter un jeton à chacun des sommets directement en aval.

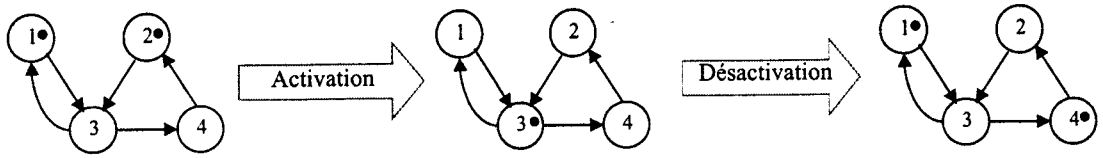


Figure 4 : Evolution du marquage.

➢ *Héritage de COMMONER.*

Nous rappelons trois théorèmes démontrés dans [COMMONER F., 71].

- Théorème 1 : *Un marquage est vivant si et seulement si le nombre de jetons de chaque circuit est positif.*

- Théorème 2 : *Un marquage vivant est sauf si et seulement si chaque arc du graphe appartient à un circuit dont le nombre de jetons est égal à 1.*

- Théorème 4 : *Pour chaque graphe fini, orienté et fortement connexe, il existe un marquage vivant et sauf.*

Par rapport à la définition de COMMONER, le marquage du formalisme GMT a simplement migré des arcs aux sommets. Ce formalisme hérite donc des théorèmes «graphe vivant » et « graphe sauf » donnés par COMMONER.

- Théorème 1 : *Un marquage est vivant si et seulement si le nombre de jetons de chaque circuit est positif.*

- Théorème 2 : *Un marquage vivant est sauf si et seulement si chaque sommet du graphe appartient à un circuit dont le nombre de jetons est égal à 1.*

- Théorème 4 : *Pour chaque graphe fini, orienté et fortement connexe, il existe un marquage vivant et sauf.*

➢ *La temporisation.*

La temporisation du graphe est réalisée en associant à chaque sommet v_j du graphe, un réel positif ou nul, noté d_j qui représente hors aléa, le temps d'exécution de la tâche T_j associée au sommet v_j.

Nous pouvons déjà entrevoir que l'étude de la célérité (temps de cycle) et de la vivacité (graphe vivant et graphe sauf) d'une application passe par l'étude des circuits du graphe GMT de cette application.

[2] La désactivation peut exiger des sommets de synchronisation, Cf. graphe expansé *§2.4.3.1.*

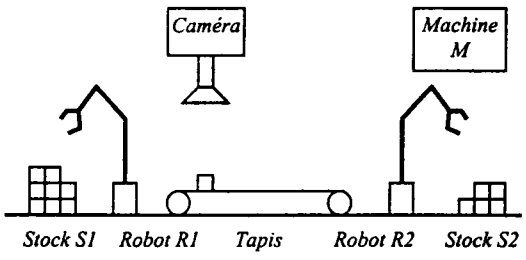
2.3. Représentation d'une application.

L'identification des tâches et plus précisément le niveau de découpage des tâches d'une application s'appuie sur des critères d'ordre topologique, fonctionnel et informationnel : bien qu'il puisse par la suite être remis en cause, nous supposons ici que ce travail décisif est déjà accompli.

[MERLAUD CH1., 84], [TAILLARD P., 91], [FROMENT B., 93] ont depuis quelques années déjà, introduit le concept d'antériorité entre les tâches d'une application pour en exprimer la séquentialité. Aux premiers stades d'une analyse, la démarche déductive ou descendante est en effet très naturelle. Nous nous associons à cette approche pour spécifier de la façon la plus élémentaire, les relations de précédence entre les diverses tâches d'une application. Ces relations peuvent alors être représentées dans un graphe GMT, lui-même consigné dans sa matrice d'antériorité *A*.

Nous nous appuyons sur l'exemple donné par AYGALINC pour exposer la démarche [AYGALINC P. & al., 92]. Initialement, le cahier des charges du système de production exposé par AYGALINC comportait deux types de pièces *P1* et *P2*. MARES réduit cet exemple à une seule pièce, ce qui rend le graphe non disjonctif, de type événements [MARES M., 97].

Ti	*Description de la tâche*	*Durée di*
T1	R1 saisit une pièce dans S1	d1 = 5 u
T2	R1 retourne à sa position de départ	d2 = 3 u
T3	R1 dépose une pièce sur le tapis	d3 = 1 u
T4	Convoyage (marche tapis)	d4 = 11u
T5	Identification par la caméra	d5 = 0 u
T6	R2 saisit une pièce sur le tapis	d6 = 2 u
T7	R2 contribue à l'usinage sur M et retour en position initiale	d7 = 19u



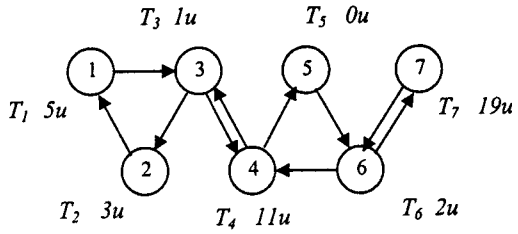
Les pièces sont saisies dans le stock S1 par le robot R1 et elles sont déposées sous le système de reconnaissance. Le tapis les achemine vers le robot R2. Le robot R2 charge la pièce sur la machine M, participe à l'usinage et dépose la pièce usinée dans le stock S2.

Figure 5 : Cahier des charges d'une application – d'après [MARES M., 97]

Le cahier des charges est lu séquentiellement : « Après avoir pris une pièce (tâche T_1) et arrêté le tapis (fin de tâche T_4), le robot *R1* la dépose sur le tapis (tâche T_3). Simultanément, le robot *R1* retourne à sa position de repos (tâche T_2) et le tapis démarre (tâche T_4)...etc.»

Chaque tâche de l'application est associée à un sommet du graphe et l'existence d'une antériorité entre deux tâches T_i et T_j (tâche T_i antérieure à la tâche T_j) est représentée sur le graphe par un arc orienté du sommet v_i vers le sommet v_j.

S'il existe, l'arc orienté (v_j, v_i) est consigné dans la *matrice d'antériorité A* en imposant la valeur *1* à l'élément a_{ij} (*0* dans le cas contraire).



A	v1	v2	v3	v4	v5	v6	v7
v1	0	1	0	0	0	0	0
v2	0	0	1	0	0	0	0
v3	1	0	0	1	0	0	0
v4	0	0	1	0	0	1	0
v5	0	0	0	1	0	0	0
v6	0	0	0	0	1	0	1
v7	0	0	0	0	0	1	0

Figure 6 : Spécification dans le formalisme GMT et matrice d'antériorité A.

➢ *Interprétation et lecture de la matrice d'antériorité.*

- La lecture de la matrice (*Figure 7*) nous renseigne sur les obligations et autorisations de l'application. Sur l'exemple présenté *Figure 6*, la lecture horizontale de la troisième ligne signifie que les nœuds v_1 et v_4 précèdent le nœud v_3, autrement dit, qu'avant de faire la tâche T_3 associée au nœud v_3, il faut *nécessairement* faire les tâches T_1 et T_4 associées aux nœuds v_1 et v_4. De même, la lecture de la sixième colonne nous indique que les nœuds v_4 et v_7 succèdent au nœud v_6. En termes d'application, la fin de la tâche T_6 associée au nœud v_6 autorise le lancement des tâches T_4 et T_7 associées aux nœuds v_4 et v_7. C'est ici une autorisation et non pas une obligation. Si T_7 peut démarrer dès la fin de T_6, T_4 doit également attendre la fin de T_3 (synchronisation).

- Lorsque la matrice d'antériorité A est '*ré-organisée*', c'est-à-dire lorsque la largeur de sa bande diagonale est minimale, cette matrice laisse apparaître les circuits du graphe GMT visualisés *Figure 6* par des zones ombrées (la zone supérieure gauche représente le circuit 1-3-2). Nous verrons au paragraphe *3.4* que la méthode DSM (Design Structure Matrix) permet de réaliser cet arrangement.

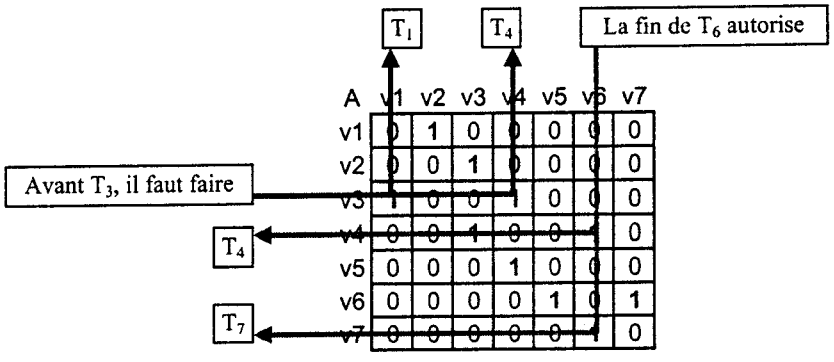


Figure 7 : Lecture de la matrice d'antériorité.

2.4. La passerelle GMT-Grafcet.

Nous allons à présent construire les deux procédures qui permettent au formalisme GMT de s'intégrer dans la plate-forme multi-modèle. Ces deux procédures ne portent que sur l'architecture du graphe : *nous ne considérons pas ici les aspects marquage et temporisation.* Nous examinons au préalable quelques propriétés issues de la matrice d'antériorité A et de sa transposée B.

2.4.1. Préliminaires, notations.

Nous nous intéressons dans ce paragraphe à la matrice A, à sa matrice transposée $B = A^T$ et aux matrices produit d'entrée et de sortie respectivement $E = A * A^T$ et $S = A^T * A$. Si l'élément de B, $b_{ij} = 1$ alors le nœud v_i admet le nœud v_j comme successeur. On note u_e et u_s respectivement le nombre de termes égaux à « 1 » sur les diagonales des matrices E et S.

➢ *Proposition 1.*

Les termes e_{ii}, éléments de E, représentent le nombre de prédécesseurs que possède le nœud v_i, c'est-à-dire le nombre d'arcs entrants sur le nœud v_i. Si le terme e_{ii} est nul, alors le nœud v_i est un nœud source (pas de prédécesseurs). En termes de graphe, e_{ii} est égal à $d_G^-(v_i)$, le demi-degré intérieur du nœud v_i.

➢ *Proposition 2.*

Les termes s_{ii}, éléments de S, représentent le nombre de successeurs que possède le nœud v_i, c'est-à-dire le nombre d'arcs sortants du nœud v_i. Si le terme s_{ii} est nul, alors le nœud v_i est un nœud puits (pas de successeur). En terme de graphe, s_{ii} est égal à $d_G^+(v_i)$, le demi-degré[3] extérieur du nœud v_i.

➢ *Proposition 3.*

trace(E) = trace(S) = k où k représente le nombre d'arcs du graphe.

Nous justifions ces propositions.

➢ *Justification de la proposition 1.*

Si $i = j$ alors $a_{ik} = b_{kj}$, si $e_{ij} = a_{ik} * b_{kj} = 1$ alors $a_{ik} = 1$, c'est-à-dire qu'il existe un arc sortant du nœud v_k et entrant sur le nœud v_i.

La somme $\sum_{i=1}^{i=n} a_{ik} * b_{kj}$ représente donc le nombre d'arcs entrants sur le nœud v_i soit $d_G^-(i)$.

[3] $e_{ii} + s_{ii} = d_G^+(v_i) + d_G^-(v_i) = d_G(v_i)$ où $d_G(v_i)$ est le degré du sommet v_i. Cf. §3.1.

On montre de la même façon que le terme e_{ij} représente le nombre de prédécesseurs communs aux nœuds v_i et v_j.

➢ *Justification de la proposition 2.*

Si $i = j$ alors $b_{ik} = a_{kj}$, si $s_{ij} = b_{ik} * a_{kj} = 1$ alors $b_{ik} = 1$, c'est-à-dire qu'il existe un arc sortant du nœud v_i et entrant sur le nœud v_k.

La somme $\sum_{i=1}^{i=n} b_{ik} * a_{kj}$ représente donc le nombre d'arcs sortants du nœud v_i soit $d_G^+(i)$.

On montre de la même façon que le terme s_{ij} représente le nombre de successeurs communs aux nœuds v_i et v_j.

➢ *Justification de la proposition 3.*

La *trace(E)* (respectivement *trace(S)*) est égale à la somme de tous les arcs entrants (respectivement sortants) sur tous les nœuds du graphe : les arcs ayant chacun une destination (respectivement une origine), la somme des destinations (respectivement des origines) des arcs est donc égale au nombre d'arcs *k*. Soit $trace(E) = trace(S) = k$.

2.4.2. Etablissement GMT → Grafcet.

Nous rappelons que nous ne considérons que l'aspect architecture du modèle GMT, soit, uniquement les nœuds et les arcs du modèle sans considération de marquage et de temporisation.

Un Grafcet est un graphe *G(V, E)* biparti où *V* et *E* représentent respectivement l'ensemble des nœuds et l'ensemble des arcs. L'ensemble des nœuds *V* est partitionné en deux sous-ensembles *Q* et *R*, tels que :

$$V = Q \cup R, Q \cap R = \varnothing \quad \text{et} \quad \forall (i,j) \in E, i \in Q \Rightarrow j \in R, i \in R \Rightarrow j \in Q$$

où *Q* et *R* représentent respectivement l'ensemble des étapes et l'ensemble des transitions du modèle Grafcet. La structure d'un modèle Grafcet est donc complètement représentable et définie par deux matrices d'incidence disjointes que nous notons *I* et *O*:

- *I* est la matrice d'entrée, telle que si l'étape q_j est une étape antérieure à la transition r_i alors $i_{ij} = 1$ sinon $i_{ij} = 0$.

- *O* est la matrice de sortie, telle que si l'étape q_j est une étape postérieure à la transition r_i alors $o_{ij} = 1$ sinon $o_{ij} = 0$.

La construction des matrices *I* et *O* nécessite l'expansion de la matrice d'antériorité *A*, notée *AE* : *AE* est de dimension *q* et *q* représente le nombre d'étapes de l'ensemble *Q*. Nous calculerons ensuite le nombre de transitions *r* de l'ensemble *R*.

2.4.2.1. Expansion de la matrice d'antériorité A.

La matrice *AE* est de dimension *q* telle que $q = n+s$ où *n* représente le nombre de nœuds (ou tâches) et *s* représente le nombre d'étapes de synchronisation.

Ce nombre *s* est égal à la somme des demi-degrés intérieurs supérieurs à '*1*'», c'est-à-dire aux termes diagonaux e_{ii} de *E* tel que $e_{ii}>1$. En effet, si $d_G^-(i) > 1$, alors la tâche T_i ne peut être lancée qu'après la fin d'exécution des $d_G^-(i)$ tâches directement antérieures et de ce fait nécessite $d_G^-(i)$ étapes de synchronisation. Nous pouvons donc énoncer une première règle.

REGLE 1 : $\boxed{q = trace(E) + n - u_e}$

Le nombre d'étapes du Grafcet est égal au nombre de nœuds du graphe d'antériorité, augmenté du nombre d'arcs arrivant sur chaque nœud, si ce nombre est supérieur à '*1*'.

La construction de la matrice *AE* passe alors par le respect de deux principes :

- *L'expansion de la matrice A.*

Pour tout nœud v_i, si $e_{ii} > 1$, alors on ajoute e_{ii} lignes et e_{ii} colonnes à la matrice *A*. Le sommet v_i 'attend' les sommets v_j, v_k, v_l... ce qui entraîne la création de e_{ii} sommets d'attente des nœuds v_j, v_k, v_l... vers v_i : ces nœuds sont notés v_{ji}, v_{ki}, v_{li}...

- *La création des liens de la matrice AE.*

Pour tout nouveau nœud v_{ji}, $ae_{i\,ji} := 1$ et $ae_{ji\,j} := 1$

la colonne associée au nœud v_{ji} reçoit un « 1 » sur la ligne associée au sommet v_i et la ligne associée au sommet d'attente v_{ji} reçoit un « 1 » sur la colonne associée au sommet v_j.

2.4.2.2. Obtention des matrices I et O.

Définissons tout d'abord le nombre *r* de transitions du modèle Grafcet postulé. Dans le modèle Grafcet, chaque étape associée au nœud v_i (hormis les nouveaux nœuds v_{ji}) est précédée et suivie par deux transitions antérieure et postérieure (*Figure 8a*) notées respectivement r_{ai} et r_{pi}.

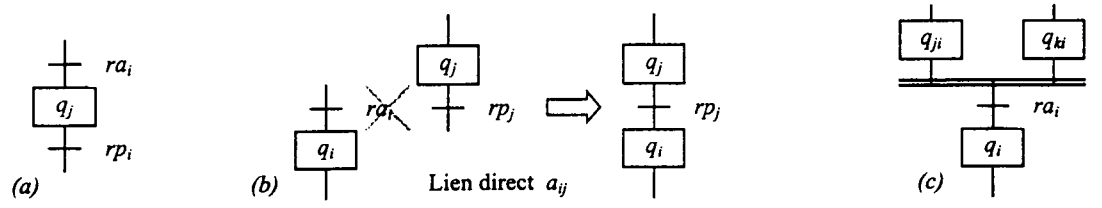


Figure 8 :Obtention des matrices I et O.

Si deux nœuds v_i et v_j (hormis les nouveaux nœuds v_{ji}) sont directement liés par le lien a_{ij}, alors la transition r_{pj} est confondue avec la transition r_{ai} (*Figure 8b*). On supprime les transitions antérieures r_{ai} aux étapes associées à des nœuds postérieurs v_i à des sommets v_j si le lien a_{ij} est direct (ce choix est arbitraire, on aurait pu aussi supprimer la transition r_{pj}). Par contre, la transition r_{ai} subsiste si l'étape associée au nœud v_i est précédée par des étapes associées à des nœuds de synchronisation v_{ji} (*Figure 8c*).

Nous énonçons une seconde règle.

REGLE 2 : $\boxed{r = 2.n - u_e}$

Le nombre de transitions d'un Grafcet issu d'un graphe d'antériorité est égal à deux fois le nombre de nœuds de ce graphe, diminué du nombre de liens directs, c'est-à-dire du nombre de nœuds qui n'ont qu'un seul arc d'entrée.

La construction des matrices *I* et *O* distingue alors quatre cas :

➢ *La création des liens de la matrice I :*

- Pour toute transition antérieure r_{ai}, $i_{ix} := 1$, si la colonne x est associée à un sommet de synchronisation antérieur au nœud v_i,

- Pour toute transition postérieure r_{pi}, $i_{ix} := 1$, x étant la colonne associée au nœud v_i.

➢ *La création des liens de la matrice O :*

- Pour toute transition postérieure r_{pi}, $o_{ix} := 1$, si la colonne x est associée à un sommet de synchronisation postérieur ou à un nœud v_j directement postérieur au nœud v_i.

- Pour toute transition antérieure r_{ai} , $o_{ix} := 1$, x étant la colonne associée au nœud v_i,

La démarche est décrite par l'algorithme de l'annexe 1 – paragraphe *3.2*. Les seules données d'entrée sont le nombre de nœuds n et la matrice d'antériorité A. L'algorithme fournit alors les deux matrices d'entrée et de sortie *I* et *O*.

2.4.3. Etablissement Grafcet → GMT.

Comme précédemment, cette procédure ne touche que les *aspects architecturaux* des modèles. Nous rappelons que le modèle Grafcet doit être un *graphe d'événements*. L'établissement du modèle GMT à partir du modèle Grafcet s'effectue en trois phases :

- L'ordonnancement[4] des matrices d'entrées sorties *I* et *O*.
- La construction de la matrice expansée *AE* à partir des matrices ordonnées *I* et *O*.
- L'obtention de la matrice d'antériorité *A* par la compression de la matrice *AE*.

[4] Cette opération est inutile si le Grafcet est issu de la production automatique décrite au paragraphe 2.4.2.

2.4.3.1. Obtention de la matrice expansée.

➢ *L'ordonnancement des matrices I et O.*

L'ordonnancement a pour but d'une part, d'isoler les étapes de synchronisation du Grafcet sur les dernières colonnes des matrices *I* et *O,* d'autre part, d'isoler les transitions antérieures sur les dernières lignes. L'ordre dans lequel sont arrangées les étapes et les transitions est quelconque, pourvu que ce soit le même pour les deux matrices. La permutation des colonnes et des lignes doit respecter les liens antérieurs et postérieurs étape/transition des matrices d'entrées et de sorties initiales.

➢ *Obtention de la matrice AE*

La matrice expansée des antériorités entre les tâches d'une application est égale au produit de la transposée de la matrice *O* par la matrice *I, O* et *I* étant les matrices ordonnées d'entrée et de sortie du grafcet non disjonctif de l'application.

$$AE(q,q) = O^T(q,r) * I(r,q)$$

En effet, si $ae_{ij} = 1$ alors le sommet v_j est antérieur au sommet v_i. L'alternance étape-transition du formalisme GRAFCET impose l'existence d'une *seule* transition r_k reliant l'étape q_j (associée au nœud v_j) à l'étape q_i (associée au nœud v_i). Le lien q_j - r_k est représenté par i_{kj} dans la matrice *I* et le lien r_k - q_i est représenté par o_{ki} dans la matrice *O*.

L'antériorité de q_j sur q_i implique d'une part que le coefficient ae_{ij} soit égal à 1, et d'autre part que i_{kj} et o_{ki} soient aussi égaux à 1 : donc, si q_j est antérieur à q_i, alors $ae_{ij} = 1 = o_{ki} . i_{kj}$ sinon, $ae_{ij} = 0 = o_{ki} . i_{kj}$.

Ce qui est équivalent à dire: $\forall i, j$, $ae_{ij} = o_{ki} . i_{kj} = o^T_{ih} . i_{kj}$, *soit* $AE = O^T * I$.

Remarque : De la même manière, le produit $I(r,q) * O^T(q,r)$ conduit à la matrice des antériorités sur les transitions.

2.4.3.2. Compression de la matrice expansée AE.

Cette opération consiste à remplacer tous les liens v_{ji} - v_i par les liens v_j - v_i où v_{ji} représente un nœud de synchronisation du nœud v_j vers le nœud v_i. Le coefficient $ae_{i\ ji}$ glisse alors horizontalement sur la ligne *i* de la colonne associée au sommet v_{ji} à la colonne associée au sommet v_j. Nous supprimons ensuite dans la matrice *AE* toutes les lignes et toutes les colonnes associées aux sommets de synchronisation. La démarche est décrite par l'algorithme de l'annexe 1 – paragraphe *3.1.*

2.4.4. Exemple d'application.

Nous considérons à nouveau l'exemple présenté Figure 5 et Figure 6 du paragraphe 2.3. Nous cherchons le modèle Grafcet associé au modèle GMT.

➢ *Calcul de la matrice des entrées E et des dimensions q et r.*

A	v1	v2	v3	v4	v5	v6	v7
v1	0	**1**	0	0	0	0	0
v2	0	0	**1**	0	0	0	0
v3	**1**	0	0	**1**	0	0	0
v4	0	0	**1**	0	0	**1**	0
v5	0	0	0	**1**	0	0	0
v6	0	0	0	0	**1**	0	**1**
v7	0	0	0	0	0	**1**	0

$E=A*A^T$

E	v1	v2	v3	v4	v5	v6	v7
v1	**1**	0	0	0	0	0	0
v2	0	**1**	0	1	0	0	0
v3	0	0	**2**	0	1	0	0
v4	0	1	0	**2**	0	0	1
v5	0	0	1	0	**1**	0	0
v6	0	0	0	0	0	**2**	0
v7	0	0	0	1	0	0	**1**

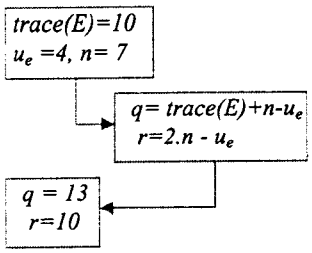


Figure 9 : Détermination du nombre d'étapes et de transitions du grafcet postulé.

➢ *Expansion.*

Nous construisons la matrice *AE* à partir de la matrice *A* en appliquant des règles du paragraphe *2.4.2.1.*

AE	v1	v2	v3	v4	v5	v6	v7	v13	v43	v34	v64	v56	v76
v1	0	**1**	0	0	0	0	0	0	0	0	0	0	0
v2	0	0	**1**	0	0	0	0	0	0	0	0	0	0
v3	***0***	0	0	***0***	0	0	0	**1**	**1**	0	0	0	0
v4	0	0	***0***	0	0	***0***	0	0	0	**1**	**1**	0	0
v5	0	0	0	**1**	0	0	0	0	0	0	0	0	0
v6	0	0	0	0	***0***	0	***0***	0	0	0	0	**1**	**1**
v7	0	0	0	0	0	**1**	0	0	0	0	0	0	0
v13	**1**	0	0	0	0	0	0	0	0	0	0	0	0
v43	0	0	0	**1**	0	0	0	0	0	0	0	0	0
v34	0	0	**1**	0	0	0	0	0	0	0	0	0	0
v64	0	0	0	0	0	**1**	0	0	0	0	0	0	0
v56	0	0	0	0	**1**	0	0	0	0	0	0	0	0
v76	0	0	0	0	0	0	**1**	0	0	0	0	0	0

Figure 10 : Construction de la matrice expansée.

Exemple d'application des règles au nœud v_3 : le nœud v_3 a deux nœuds qui le précédent : v_1 et v_4. On crée alors deux nœuds de synchronisation v_{13} et v_{43}.

Nous raisonnons sur le nœud v_4. Le lien v_4-v_3 est supprimé (zéro en italique à l'intersection des deux flèches) pour être remplacé par le lien v_4-v_{43} (indiqué par la flèche verticale) et par le lien v_{43}-v_3 (indiqué par la flèche horizontale).

➢ *Obtention de des matrices I et O.*

Les règles énoncées au paragraphe 2.4.2.2 permettent finalement d'obtenir les deux matrices antérieure et postérieure du modèle Grafcet.

I	*q1*	*q2*	*q3*	*q4*	*q5*	*q6*	*q7*	*q13*	*q43*	*q34*	*q64*	*q56*	*q76*
rp1	1	0	0	0	0	0	0	0	0	0	0	0	0
rp2	0	1	0	0	0	0	0	0	0	0	0	0	0
rp3	0	0	1	0	0	0	0	0	0	0	0	0	0
rp4	0	0	0	1	0	0	0	0	0	0	0	0	0
rp5	0	0	0	0	1	0	0	0	0	0	0	0	0
rp6	0	0	0	0	0	1	0	0	0	0	0	0	0
rp7	0	0	0	0	0	0	1	0	0	0	0	0	0
ra3	0	0	0	0	0	0	0	1	1	0	0	0	0
ra4	0	0	0	0	0	0	0	0	0	1	1	0	0
ra6	0	0	0	0	0	0	0	0	0	0	0	1	1

O	*q1*	*q2*	*q3*	*q4*	*q5*	*q6*	*q7*	*q13*	*q43*	*q34*	*q64*	*q56*	*q76*
rp1	0	0	0	0	0	0	0	1	0	0	0	0	0
rp2	1	0	0	0	0	0	0	0	0	0	0	0	0
rp3	0	1	0	0	0	0	0	0	0	1	0	0	0
rp4	0	0	0	0	1	0	0	0	1	0	0	0	0
rp5	0	0	0	0	0	0	0	0	0	0	0	1	0
rp6	0	0	0	0	0	0	1	0	0	0	1	0	0
rp7	0	0	0	0	0	0	0	0	0	0	0	0	1
ra3	0	0	1	0	0	0	0	0	0	0	0	0	0
ra4	0	0	0	1	0	0	0	0	0	0	0	0	0
ra6	0	0	0	0	0	1	0	0	0	0	0	0	0

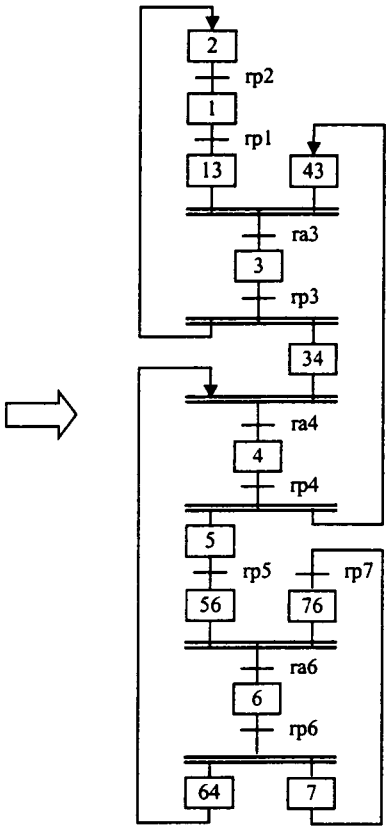


Figure 11 : Matrices antérieure et postérieure du modèle Grafcet.

➢ *Retour au modèle GMT.*

Le produit $O^T * I$ des matrices postérieure et antérieure représentées Figure 11 conduit à la matrice expansée donnée Figure 10. Nous supprimons les nœuds de synchronisation en restituant les liens pour revenir à la matrice d'antériorité de départ *A*.

AE	*v1*	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*	*v13*	*v43*	*v34*	*v64*	*v56*	*v76*
v1	0	1	0	0	0	0	0						
v2	0	0	1	0	0	0	0						
v3	1	0	0	1	0	0	0	1	1				
v4	0	0	1	0	0	1	0			1	1		
v5	0	0	0	1	0	0	0						
v6	0	0	0	0	1	0	1					1	1
v7	0	0	0	0	0	1	0						
v13													
v43													
v34													

A	*v1*	*v2*	*v3*	*v4*	*v5*	*v6*	*v7*
v1	0	1	0	0	0	0	0
v2	0	0	1	0	0	0	0
v3	1	0	0	1	0	0	0
v4	0	0	1	0	0	1	0
v5	0	0	0	1	0	0	0
v6	0	0	0	0	1	0	1
v7	0	0	0	0	0	1	0

Figure 12 : Retour à la matrice d'antériorité par compression de AE.

2.4.5. Univocité des formalismes.

2.4.5.1. Univocité des architectures.

Nous ne considérons que l'architecture des modèles GMT ou Grafcet, tous deux de type événements. Le passage du formalisme GMT au formalisme GRAFCET peut être représenté par une application $\mathcal{A}$ de l'ensemble des graphes GMT à l'ensemble des Grafcets. Cette application est elle-même la composée de deux applications f et g permettant respectivement de passer de la matrice d'antériorité A à la matrice expansée AE, puis de la matrice expansée AE aux deux matrices I et O représentatives du modèle Grafcet.

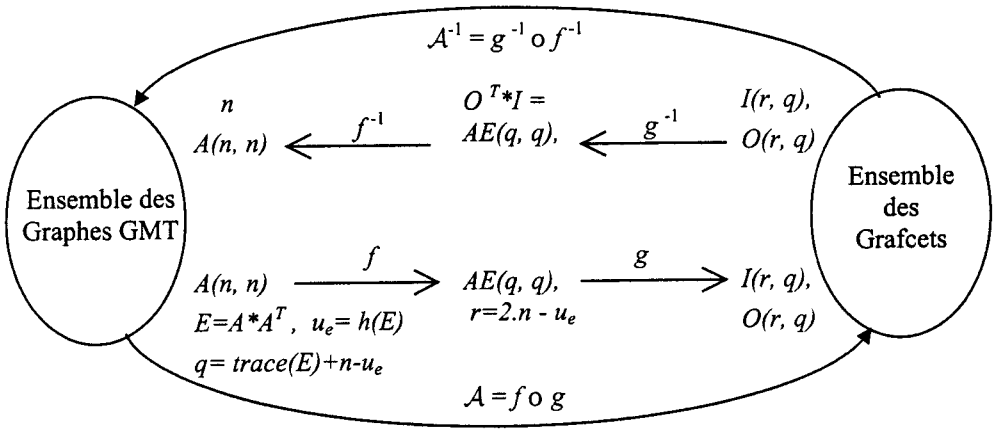


Figure 13 : Application de {GMT} dans {Grafcet}.

Nous montrons que l'application $\mathcal{A}$ est bijective.

➢ *L'application g est bijective. En effet,*

$\forall A$, l'image AE de A par g existe et est unique. En effet,

$\forall i, j \in [1..n]^2 \quad g(a_{ij}) = (ae_{iv}, ae_{vj})$

$\forall i, j \in [1..n]^2$ *si* $a_{ij} . e_{ii}=1$ alors $ae_{ij} := a_{ij}$, $ae_{jj} := a_{jj}$

sinon $ae_{iv} := 1$, $ae_{vj} := 1$ *et* $v \in [n+1..q]$

$\forall AE$, l'image A de AE par g^{-1} existe et est unique. En effet,

$\forall i, j \in [1..n]^2$, $\forall v \in [n+1..q]$, $g^{-1}(ae_{iv}, ae_{vj}) := a_{ij}$ avec $a_{ij} := ae_{iv} . ae_{vj}$

$\forall i, j \in [1..n]^2$, $g^{-1}(ae_{ij}, \varnothing) = a_{ij}$ *où* $\varnothing$ représente l'arc nul

➢ *L'application* f *est bijective. En effet,*

$\forall AE$, l'image (I, O) de AE par f existe et est unique. En effet,

$\forall i, j \in [1..q]^2$, $f(a_{ij}) = (i_{ui}, o_{ui})$ *avec* $u \in [1..r]$

$\forall i \in [1..n]$, $i_{ii} := 1$, $\forall j \in [1..q]$ $o_{ij} := ae_{ji}$

$\forall i \in [1..n]$, *si* $e_{ii} > 1$ *alors* $o_{ui} := 1$ *et* $\forall j$, $i_{uj} := ae_{ij}$, $u \in [n+1..r]$

$\forall$ le couple (I, O), l'image AE *de* (I, O) par f^{-1} existe et est unique. En effet,

$\forall i, j \in [1..q]^2$, $\forall u \in [1..r]$, $ae_{ij} := \sum i_{uj} \cdot o_{ui}$

Les applications f et g étant bijectives, l'application composée $\mathcal{A} = f$ o g l'est aussi. Sur le *plan architecture*, les deux formalismes sont *univoques.*

2.4.5.2. Image des marquages et des temporisations.

Nous considérons en premier lieu les propriétés liées au marquage (*'graphe vivant', 'graphe sauf'*). Nous avons vu précédemment (§ *2.2*) qu'un marquage d'un graphe GMT est vivant et sauf si et seulement si tous les arcs appartiennent à des circuits et si le nombre de jetons de chaque circuit est égal à 1.

Nous montrons que l'image d'un circuit par l'application $\mathcal{A}$ est également un circuit du modèle Grafcet. En effet, un circuit du modèle GMT peut être représenté par un vecteur de dimension n, où chaque coordonnée prise dans {0, 1} précise la présence ou non du nœud correspondant dans ce circuit : $Cj = (1, 0, \ldots 0, 1, 0, \ldots, 1)$. Lorsque le degré intérieur d'un nœud v_i de Cj est supérieur à 1, c'est-à-dire lorsqu'il y a plusieurs arcs entrants sur ce nœud v_i, alors l'image $\mathcal{A}(Cj)$ du circuit comprend une étape supplémentaire v_{ui}, avec $n \prec u \leq q$.

Mais par définition des nœuds de synchronisation, le nœud v_{ui} ne peut appartenir qu'à l'image de Cj, situé exactement entre v_u et v_i : l'image de Cj reste donc un circuit du modèle Grafcet.

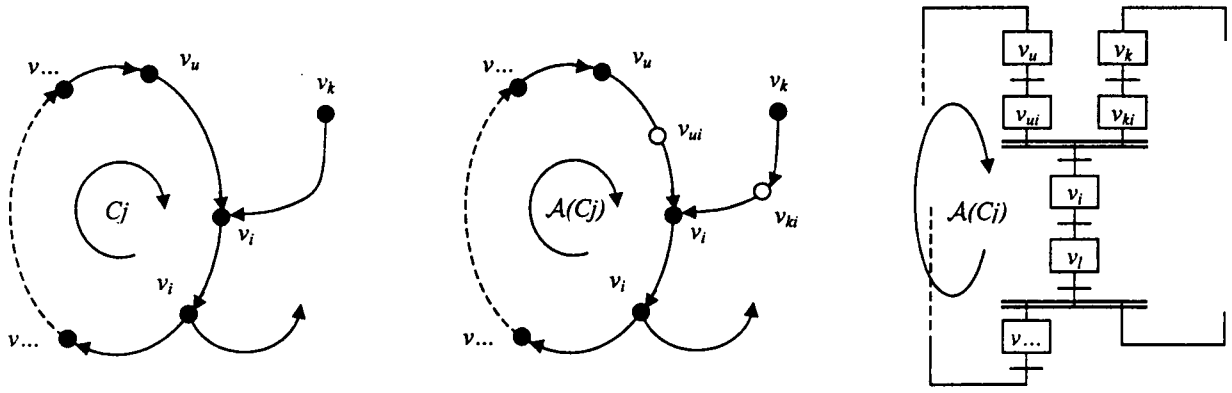


Figure 14 :Conservation des circuits par l'application $\mathcal{A}$.

Si le nombre de jetons d'un circuit du modèle GMT Cj est égal à 1, alors le nombre de jetons du circuit 'image' $\mathcal{A}(Cj)$ du modèle Grafcet est aussi égal à 1.

Si tous les sommets du graphe GMT appartiennent à des circuits, alors tous les sommets du modèle Grafcet appartiennent aussi à des circuits.

Il s'ensuit que si un modèle GMT est un graphe vivant et sauf, alors le modèle 'image' Grafcet est un graphe vivant et sauf.

Le raisonnement inverse pouvant être fait, il s'ensuit que les propriétés 'graphe vivant' et 'graphe sauf' sont conservées par l'application bijective $\mathcal{A}$.

$\mathcal{A}(Cj) = (1, 0, \ldots 0, 1, 0, \ldots, 1, \mathit{0, 0, 1, \ldots, 0})$ est une composante conservative équivalente à un P-semi-flot du formalisme RdP : la fin du vecteur $\mathcal{A}(Cj)$ (en italique) représente les nœuds de synchronisation. (Cf. § 2.1.1.3. première partie)

Nous examinons à présent l'aspect temporel. Les durées d_i des tâches T_i associées aux nœuds v_i du graphe GMT sont reportées sur les étapes q_i de l'image Grafcet. Hors aléa et en considérant le problème central[5], le temps de cycle d'un circuit isolé d'un graphe GMT est égal à la somme de durées d_i des tâches T_i associées aux nœuds v_i de ce circuit. Cette somme représente le temps de cycle minimal.
Si ce circuit n'est plus isolé, le degré intérieur d'un nœud v_i de Cj peut être supérieur à 1. L'image $\mathcal{A}(Cj)$ du circuit comprend une étape supplémentaire v_{ui}, avec $n < u \leq q$. La durée associée à ce nœud est la plus grande des valeurs d_{xi}, x représentant l'indice des nœuds participant à la synchronisation sur le nœud v_i. Nous ne pouvons donc rien déduire du temps de cycle de l'image d'un circuit. Par contre, le plus grand des temps de cycle des circuits du modèle GMT représente le temps de cycle du modèle GMT et le temps de cycle de son image Grafcet. En régime établi, les cycles dont les temps ne sont pas les plus grands, sont donc esclaves du cycle le plus lent.

2.5. Conclusion sur le formalisme et la passerelle GMT-Grafcet.

Nous venons de décrire un formalisme de spécification préliminaire d'une application, conformément à la première phase d'étude du cycle de vie en SPIRALE. La représentation graphique rend ce formalisme explicite et les aspects 'marquages et temporisations' permettent une caractérisation suffisante du modèle, caractérisation par ailleurs conservée par la passerelle GMT-Grafcet. Cette passerelle ouvre le champ à de nombreux formalismes qui admettent comme point de départ un modèle Grafcet et qui sont susceptibles d'approfondir la conception de l'application suivant le cycle SPIRALE.

[5] Un problème est dit central si on a l'assurance que les ressources externes ne manqueront pas.

3. L'apport de la théorie des graphes.

Un bon dessin vaut mieux qu'un long discours ! Cet adage ne devait pas prendre en compte le discours possible sur le dessin. La résolution d'une curiosité mathématique donnait à EULER en 1736, l'occasion d'écrire le premier article sur les graphes : Comment parcourir une fois et une seule fois les sept ponts de la ville de KOENIGSBERG ?

Figure 15 : Le problème d'EULER : les sept ponts de KOENIGSBERG.

Le hasard a voulu qu'exactement deux siècles plus tard (1936), D. KÖNIG formalise la théorie des graphes dans le premier ouvrage qui leur soit dédié. L'après-guerre et ses problèmes concrets de reconstruction ont conduit les spécialistes de la recherche opérationnelle à développer cette théorie : KHUN (1955), FORD et FULKERSON (1956), BERGE (1958) et ROY (1959) [GONDRAN M. & al., 95] (il serait équitable d'ajouter à cette liste les auteurs de l'ouvrage auquel nous venons de faire référence).

La théorie des graphes est donc une discipline récente et en étroite relation avec la recherche opérationnelle. Nous avons vu que les formalismes basés sur les systèmes de transitions s'appuient largement sur l'expression graphique naturellement offerte par les graphes : nous souhaitons aussi utiliser partiellement les résultats de la théorie qui permettent d'en analyser l'architecture.

Nous rappelons certaines définitions relatives aux graphes et à leur architecture et nous exposons quel en est l'intérêt pour l'architecture d'un modèle GMT.

Nous montrons ensuite comment il est possible de vérifier si le marquage d'un graphe GMT est vivant et sauf. Nous présentons également le principe d'évaluation des temps de cycle.

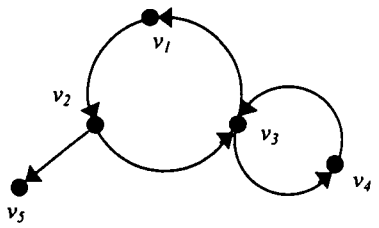
3.1. Définitions préliminaires.

Nous rappelons succinctement quelques définitions usuelles de la théorie des graphes, définitions limitées aux graphes *orientés*.

➢ *Graphe orienté, matrice d'antériorité et degré d'un sommet (Cf. Figure 16)*

Soit un graphe orienté $G(V, E)$ où V est l'ensemble fini des nœuds v_i de cardinal n et E celui des arcs $e_k = (v_i, v_j)$ de cardinal m. Les arcs $e_k = (v_i, v_j)$ sont orientés : v_i et v_j représentent respectivement l'origine et la destination de e_k.

$$A = \begin{array}{c} \begin{matrix} v1 & v2 & v3 & v4 & v5 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} v1 \\ v2 \\ v3 \\ v4 \\ v5 \end{matrix}$$



	d-	d+
v1	1	1
v2	1	2
v3	2	2
v4	1	1
v5	1	0

Figure 16 : Matrice d'antériorité, graphe et degrés.

On appelle matrice d'antériorité d'un 1-graphe ($\forall$ v_i & v_j, il n'existe pas plus d'un arc v_i-v_j) la matrice A telle que $a_{ij} = 1$ si l'arc $(v_j, v_i) \in E$ et *zéro* dans le cas contraire.

On appelle degré intérieur du nœud v_i, noté $d_G^-(v_i)$ le nombre d'arcs entrants sur v_i.

On appelle degré extérieur du nœud v_i, noté $d_G^+(v_i)$ le nombre d'arcs sortants de v_i.

On appelle degré du nœud v_i, noté $d_G(v_i)$ la somme $d_G^-(v_i) + d_G^+(v_i)$.

➢ *Chemin élémentaire, circuit élémentaire, chaîne. Cf. Figure 17a*

Un chemin élémentaire Ch_i de longueur l_c (de cardinalité l_c) est une séquence de l_c arcs telle que : $Ch_i = \{ e_1, e_2, e_3, \ldots\ldots, e_{lc} \}$ avec $e_1 = (v_{i0}, v_{i1})$ $e_2 = (v_{i1}, v_{i2})$ …… $e_{lc} = (v_{ilc-1}, v_{ilc})$ où chaque sommet v_i est de degré *2*. Ex : $\{v_4 - v_3, \quad v_3 - v_1, \quad v_1 - v_2, \quad v_2 - v_5, \quad v_5 - v_6\}$.

Un chemin source est tel que le degré intérieur de son origine est nul. Ex : $\{v_{11}–v_{12}, v_{12}–v_{13}\}$

Un chemin puits est tel que le degré extérieur de sa destination est nul. Ex : $\{v_5–v_6, v_6–v_7\}$

Un circuit élémentaire est un chemin élémentaire dont les extrémités initiale et finale coïncident. Ex : $(v_1 . v_2 . v_3)$

Une chaîne est une séquence d'arcs pour lesquels on ignore l'*orientation*. Ex : $\{v_8 - v_7, v_7 - v_6\}$

➢ *Connexité, forte connexité, complétude.*

On appelle 'composante connexe' le sous-ensemble de sommets tel qu'il existe une *chaîne* entre deux sommets quelconques de ce sous-ensemble. Le nombre de connexité d'un graphe est le nombre de composantes connexes de ce graphe.

Un graphe est connexe s'il ne comprend qu'une seule composante connexe. Le graphe de la *Figure 17a* est connexe.

Une composante fortement connexe est un sous-ensemble de sommets tel qu'il existe un *chemin* entre deux sommets quelconques de ce sous-ensemble. Les différentes composantes fortement connexes constituent une partition de l'ensemble *V*. Le sous-graphe (v_1, v_2, v_3, v_4) de la *Figure 17c* est une composante fortement connexe.

Un graphe est fortement connexe si pour tout couple de sommets (v_i, v_j), il existe un chemin joignant v_i à v_j. Le graphe de la *Figure 17a* n'est pas fortement connexe.

Si quel que soit un couple de sommets (v_i, v_j), il existe un arc v_i - v_j ou un arc v_j - v_i alors ce graphe est complet. Un 1-graphe est complet si et seulement si : $(v_i, v_j) \notin E \Rightarrow (v_j, v_i) \in E$.

➢ *Point d'articulation - Isthme*

Un point d'articulation est un sommet dont la duplication augmente le nombre de composantes connexes. Ce sommet et sa(ses) duplication(s) appartiennent à chacune des composantes connexes. Cf. *Figure 17b.*

Un isthme est un arc dont la suppression a le même effet. Cf. *Figure 17c.*

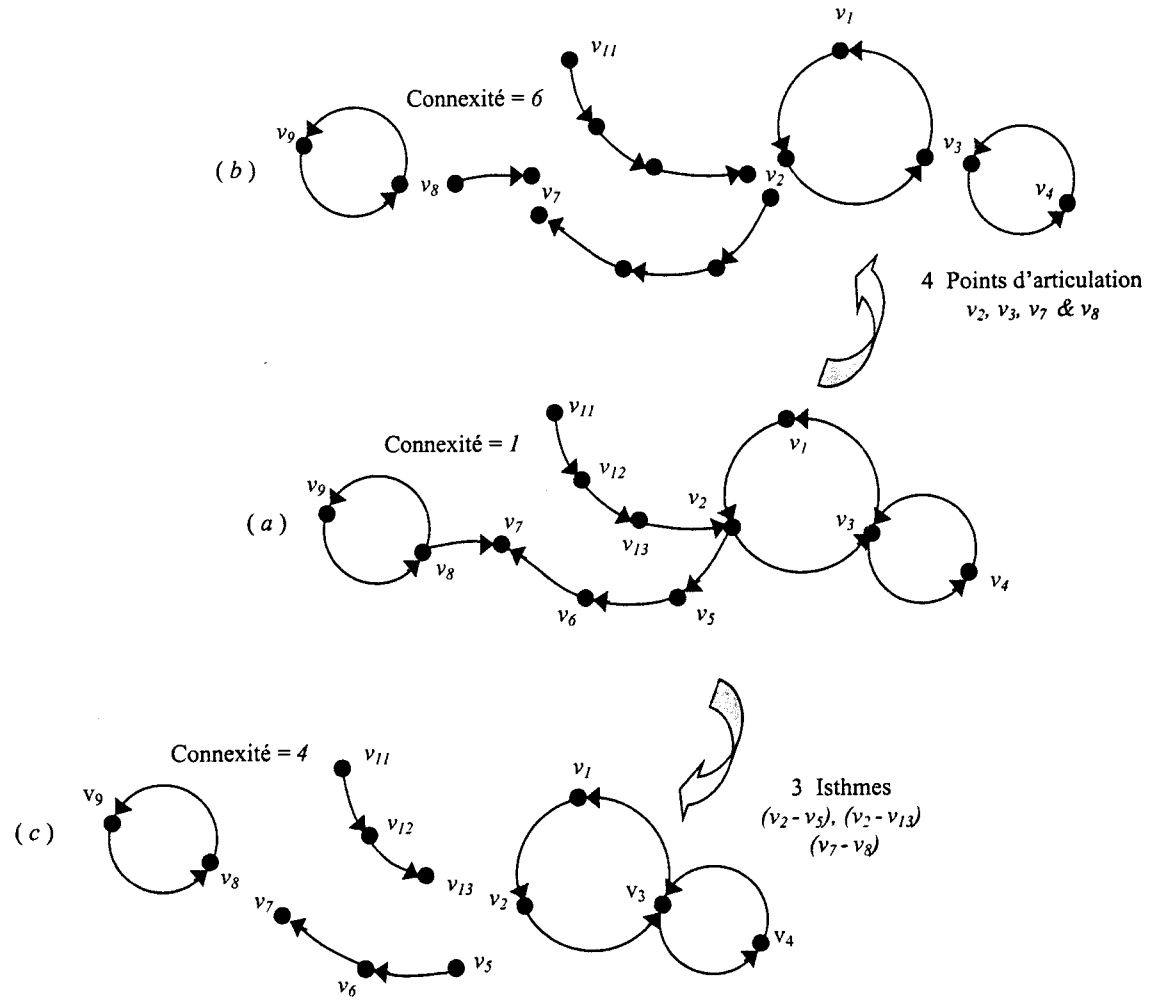


Figure 17 : Graphes, Points d'articulation et isthmes.

Un graphe est k-connexe (respectivement k-arc-connexe) s'il reste connexe par suppression de moins de k+1 sommets (respectivement k+1 arcs).

Nous étudions à présent les outils qui nous permettent d'analyser l'architecture d'un graphe GMT, soit pour agir sur cette architecture en augmentant ou en réduisant la connexité du graphe, soit pour évaluer certaines caractéristiques de ces composantes (temps de cycle, marquage).

3.2. Analyse du graphe.

La théorie des graphes fournit de nombreux algorithmes dédiés à différentes problématiques : recherche des composantes connexes (TREMAUX 1882, TARJAN 1962) et points d'articulation (TARJAN 1972), recherche du plus court chemin (FORD 1956, MOORE-DIJKSTRA 1957, BELLMAN 1958 ...), recherche du circuit de longueur moyenne (KARP 1978)...etc. et nous nous en sommes souvent inspiré. GONDRAN et MINOUX ont classé ces algorithmes en fonction de l'objectif poursuivi, c'est-à-dire en fonction de la nature des problèmes résolus : existence, énumération, dénombrement et optimisation. Le problème de l'optimisation consiste en la recherche d'un chemin le plus court, d'un flot maximal...etc. Cet objectif conduit généralement à travailler sur des arcs pondérés.

Notre problématique porte davantage sur les problèmes d'existence et d'énumération exhaustive des composantes du graphe. A la différence des algorithmes proposés par la littérature, nous examinons la caractéristique de la composante après l'avoir trouvée et nous ne nous appuyons pas sur celle-ci pour effectuer la recherche (les arcs du formalisme GMT sont unitaires). Nous sommes donc amenés à orienter les travaux de la théorie des graphes à notre problématique en les adaptant à notre formalisme.
Les procédures que nous avons développées ne permettent pas de traiter des graphes très denses et comportant beaucoup de sommets, mais elles restent adaptées à la taille[6] des problèmes qui nous concernent. A chaque fois que cela est possible, nous pallions cette limitation[7] dans la complexité des problèmes traités , en réduisant au maximum l'espace de recherche. Ces procédures ne sont pas l'objet de notre mémoire, elles sont pour nous des outils d'analyse que nous allons succinctement présenter. Le détail est fourni en annexe *1*.
Nous présentons *Figure 18* l'organisation des différents modules d'analyse de l'architecture.

---

[6] Avec 128 Mo de RAM, nous traitons des graphes de 50 nœuds comportant 250 arcs (densité de 20%) dans des temps très acceptables (par exemple, le temps de recherche de tous les circuits est d'environ une seconde).

[7] Excepté les cas qui ne nous viennent pas à l'esprit, une application comportant de très nombreuses tâches peut très souvent bénéficier du mécanisme d'encapsulation, posant ainsi une succession de problèmes de tailles raisonnables.

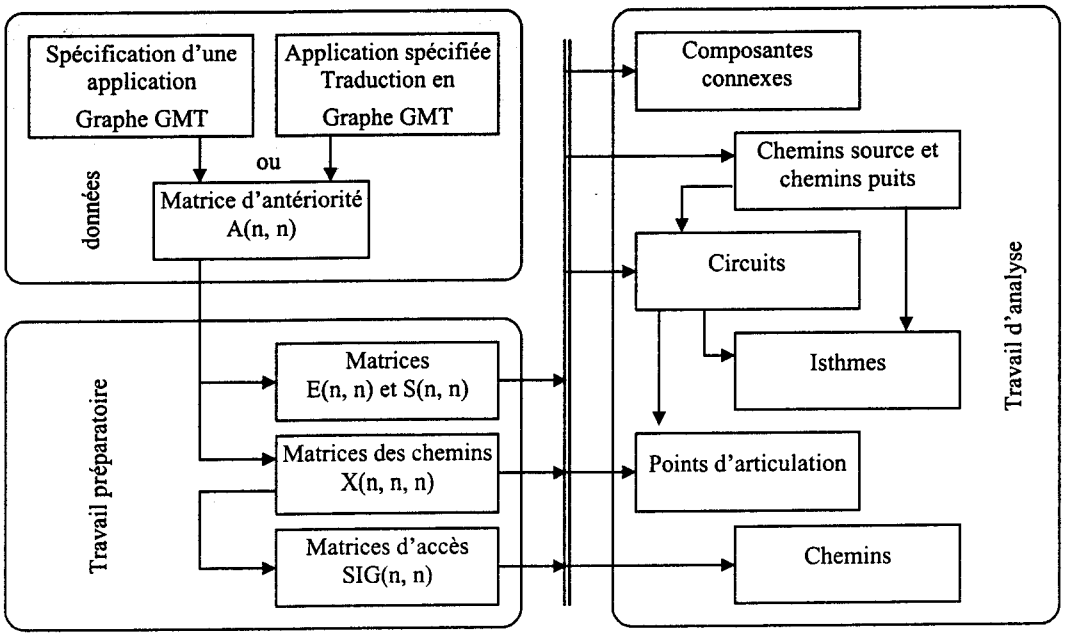


Figure 18 : Organisation des modules d'analyse de l'architecture.

Le point de départ de l'analyse d'un graphe GMT issu de la spécification ou de la traduction d'un modèle Grafcet par la passerelle Grafcet-GMT est donné par sa matrice d'antériorité (ou sa matrice d'adjacence dans la terminologie de la théorie des graphes).

A partir de cette matrice d'antériorité nous générons les matrices d'entrées/sorties (Cf. §*2.4.1*) et la matrice des chemins *X(n, n, n)* où *X(i, n, n)* représente la matrice *A(n ,n)* élevée à la puissance *i*. Nous rappelons que pour $u=1$, $A^u(i, j)=1$, signifie que v_j précède v_i.

On montre alors par récurrence que

Si $A^{u-1}(i, k)\neq 0$ implique « ∃ un chemin de *(u-1)* arcs joignant le nœud v_k au nœud v_i »

ET

Si $A(k,j)=1$, c'est-à-dire que v_j précède v_k

Alors $A^u(i,j)=\Sigma_k A^{u-1}(i, k).A(k,j) \neq 0$, signifie « ∃ au moins un chemin de u arcs joignant le nœud v_j au nœud v_i ». Ce chemin est appelé un *'u-chemin'*.

Le corollaire est que

Si $A^u(i,j)=\alpha$

ET

Si $i=j$

Alors $\exists\ \alpha$ circuits de u arcs contenant le nœud v_i. Ces circuits sont appelés *'u-circuit'*.

La matrice d'accès *SIG(n, n)* est la somme sur u des matrices A^u : $SIG = \Sigma_u A^u$. Les trois matrices *A*, *X* et *SIG* sont exploitées par les différentes procédures de recherche qui sont détaillées par la suite.

3.2.1. Connexité.

La procédure 'connexité' exploite la matrice d'accès *SIG*. Un terme *SIG(i, j)≠0* signifie qu'il existe un chemin (dont on ignore la longueur) qui permet de joindre le nœud v_j au nœud v_i. Pareillement, *SIG(j, i)≠0* signifie qu'il existe un chemin qui permet de joindre le nœud v_i au nœud v_j. Il s'ensuit que si *SIG(i, j)=0* et *SIG(j, i)=0* alors il n'existe pas de chaîne qui permet de joindre les nœuds v_i et v_j. Ils appartiennent donc à des composantes connexes distinctes. Nous étudions cette condition pour tout couple (i, j) : à chaque fois que la condition est vraie, il suffit alors de gérer, en fonction de l'appartenance ou non de chaque nœud à une composante connexe, la création ou le fusionnement des composantes connexes. Cet algorithme est de complexité $O(n^2/2)$.

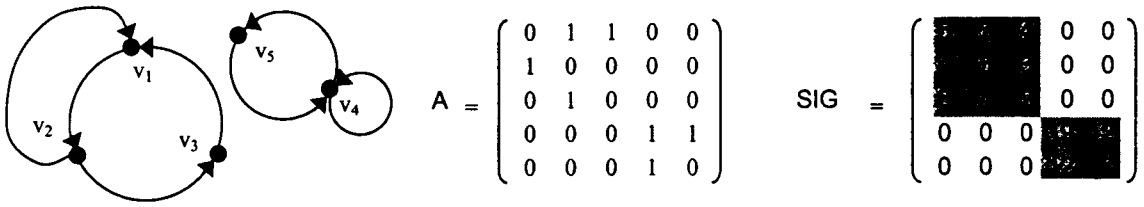


Figure 19 : Exemple de partition[8] – connexité 2.

3.2.2. Chemins sources, Chemins puits.

L'origine d'un chemin source (respectivement l'extrémité d'un chemin puits) a un degré intérieur (respectivement extérieur) nul. Pour tout i, si *E(i, i)=0* alors le nœud v_i est un nœud source. Il suffit alors de parcourir la matrice A pour trouver la suite des nœuds v_j, successeurs du nœud v_i, jusqu'à ce que le degré intérieur ou extérieur du nœud v_j ne soit plus égal à *1*. Le chemin $v_i,\ldots v_j$ est un chemin source.

Les chemins puits sont issus d'un raisonnement analogue. Pour tout i, si *S(i, i)=0* alors le nœud v_i est un nœud puits. On remonte les prédécesseurs v_j du nœud v_i, jusqu'à ce que le degré intérieur ou extérieur de v_j ne soit plus égal à *1*. Le chemin $v_j,\ldots v_i$ est un chemin puits.

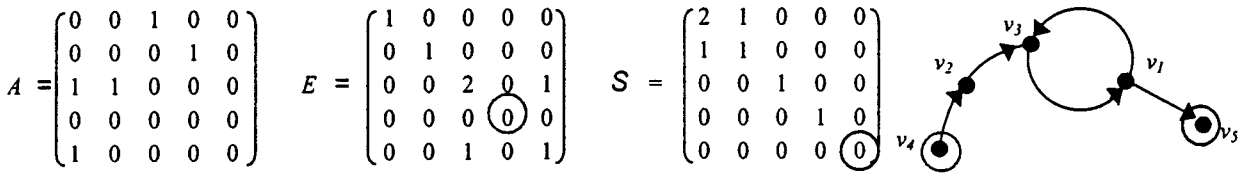


Figure 20 : Chemin source et chemin puits.

[8] Seuls les coefficients nuls de la matrice SIG présentent un intérêt pour le calcul du degré de connexité.

3.2.3. Circuits.

On supprime tout d'abord de la matrice *A* les arcs a_{ij} qui appartiennent aux chemins sources et aux chemins puits. Les circuits du graphe sont recherchés à partir de la matrice *A réduite* et de la matrice *X*.

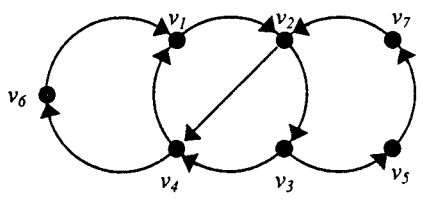
La recherche des circuits se fait en fonction du nombre *k* d'arcs formant les *k*-circuits. On commence par *k=1*, pour obtenir les boucles, puis *k=2* pour obtenir les circuits de rang 2...etc. Pour chaque rang *k*, la recherche s'effectue en profondeur dans un *k*-arbre qui ne contient que les nœuds impliqués dans des *k*-circuits, c'est-à-dire des nœuds v_i tels que le terme *X(k, i, i)* est différent de zéro. La racine du *k*-arbre est le nœud appartenant à un *k*-circuit qui est le plus impliqué dans le graphe. On parcourt l'arbre à l'aide de deux procédures 'monter' et 'descendre' qui gèrent classiquement des pointeurs de visite.

A la découverte de chaque *k*-circuit, la diagonale de la matrice *X[k, n, n]* est mise à jour en retranchant à chaque élément diagonal sa participation à ce *k*-circuit. De ce fait, la recherche de *k*-circuits dans un *k*-arbre est de plus en plus aisée, puisque à chaque découverte d'un *k*-circuit, le *k*-arbre est de moins en moins dense. La recherche de *k*-circuits dans le *k*-arbre s'arrête après examen de la diagonale de la matrice *X[k, n, n]*. Si cette diagonale ne contient pas au moins *k* éléments non nuls alors il n'existe pas ou il n'existe plus de *k*-circuit.

La recherche fournit deux résultats, *CIR[n, ncirmax]* et *TVC[n, ncirmax]*, de type matrice de listes. *CIR[k, i]* représente l'i^{ème} circuit à *k* arcs et *TVC[k, i]* le vecteur associé à ce circuit. A titre d'exemple, supposons qu'un graphe à *7* nœuds contienne *3* circuits de rang *4*, alors le deuxième circuit *(i=2)* de rang *4 (k=4)* est consigné dans les matrices *CIR[n, ncirmax]* et *TVC[n, ncirmax]* par les éléments suivants : *CIR[4,2]= 1 / 2 / 4 / 6 / 0 / 0 / 0* et *TVC[4,2]= 1 / 1 / 0 / 1 / 0 / 1 / 0*. Le premier élément mémorise l'ordre et les composantes du circuit alors que le second ne mémorise que les composantes. La seconde forme s'avère plus commode lorsqu'on souhaite supprimer et/ou évaluer des circuits.

k	*i*	--	--	--	--	--	--	--
3	1	1	2	4				
4	1	2	3	5	7			
4	2	1	2	4	6			
4	3	1	2	3	4			
5	1	1	2	3	4	6		

Table des circuits *CIR[k, i]*



k	*i*	v_1	v_2	v_3	v_4	v_5	v_6	v_7
3	1	1	1	0	1	0	0	0
4	1	1	1	0	0	1	0	1
4	2	1	1	0	1	0	1	0
4	3	1	1	1	1	0	0	0
5	1	1	1	1	1	0	1	0

Table des vecteurs *TVC[k, i]*

Figure 21 : Tables des circuits et des vecteurs circuits.

En résumé, la recherche des circuits s'effectue sur des *sous-ensembles* du graphe de départ et *la taille de chaque sous-ensemble décroît* au fur et à mesure de cette recherche.

Elle peut également se faire sur des matrices A restreintes à chaque composante connexe. L'exploration complète et non redondante de cette procédure de recherche a été testée sur des graphes complets et symétriques ($\forall i$, $\forall j$, $\exists$ l'arc (v_i, v_j)) et nous avons pu vérifier que le nombre des k-circuits ainsi trouvés dans un graphe de n sommet vérifiait l'expression $\alpha_n^k = \frac{n!}{k.(n-k)!}$. (Voir démonstration en annexe *1* § *4.3.*)

3.2.4. Isthmes.

La recherche des isthmes ne peut se faire qu'après celle des chemins 'source', des chemins 'puits' et des circuits. Nous effectuons cette recherche sur le graphe initial dépourvu des circuits et des chemins 'source' et 'puits'. Cette suppression ne porte pas sur les nœuds du graphe mais sur les arcs impliqués dans les structures à supprimer. Les isthmes se voient davantage à marée basse !

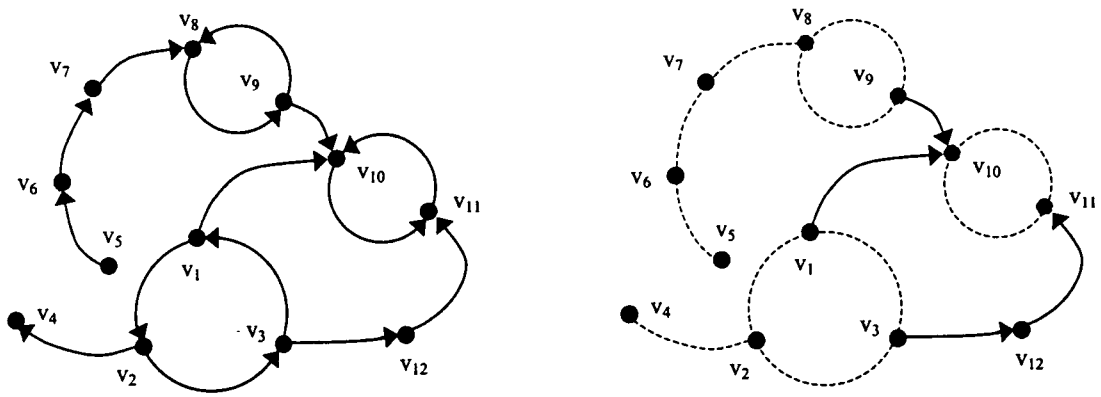


Figure 22. Recherche des isthmes.

La *Figure 22* présente le graphe initial (à gauche) et le graphe démunis des circuits et des chemins 'source' et 'puits' (à droite). Les isthmes[9] restants ne sont cependant pas de même nature. Nous avons vu au paragraphe *3.1* la notion de k-arc-connexité. La suppression du chemin v_9 - v_{10} augmente le nombre de connexité du graphe. Par contre la suppression du chemin v_1 -v_{10} OU(exclusif) du chemin v_3 - v_{12} - v_{11} n'augmente pas le nombre de connexité du graphe. Nous nous intéressons uniquement[10] au premier cas et pour tester si nous sommes dans ce cas, il suffit de vérifier que la suppression des arcs associés à l'isthme augmente effectivement la connexité (Cf. §*3.2.1*).

[9] Par abus de langage, le chemin v_3 - v_{12}- v_{11} est considéré comme un isthme, alors que l'isthme du graphe orienté est un arc : les arcs v_3-v_{12} ou v_{12}-v_{11} sont deux vrais isthmes.

[10] les isthmes doublés présentent peu d'intérêt pour la simplification des architectures (cf. § *3.3.1*) .

3.2.5. Points d'articulation.

La recherche des points d'articulation nécessite la connaissance des circuits. La procédure développée examine une par une, les différentes intersections existant entre tous les circuits du graphe pris deux à deux. Comme pour les isthmes nous nous intéressons uniquement au cas où il n'existe qu'une seule intersection. Cette procédure travaille sur les vecteurs-circuits du graphe. Elle retourne une liste des points d'intersection qui contient pour chacun de ces points, les deux circuits impliqués dans l'intersection. Nous examinons ensuite si ces points d'intersection sont des points d'articulation, c'est-à-dire si la duplication de chaque point augmente effectivement le nombre de connexité du graphe.

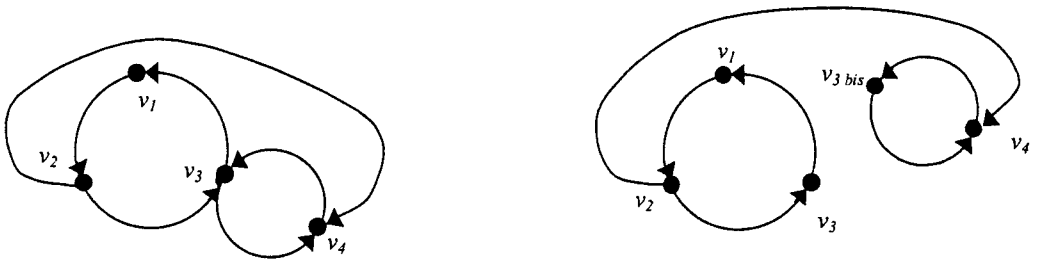


Figure 23.Recherche des points d'articulation.

Le fait qu'il n'y ait qu'une seule intersection entre deux circuits ne signifie pas que ce point est un point d'articulation. En effet, il peut exister un lien entre ces deux circuits qui fasse appel à un troisième circuit ou à un chemin.

3.2.6. Chemins, robustesse.

Nous cherchons dans le graphe initial l'ensemble des chemins joignant un nœud de départ v_d à un nœud d'arrivée v_a. La procédure développée retourne également la robustesse de la liaison v_d - v_a en calculant le nombre de chemins indépendants[11] qui relient le nœud v_d au nœud v_a. Dans le souci de l'efficacité de la recherche, nous réduisons le graphe initial en éliminant de ce dernier tous les arcs reliant des nœuds qui n'interviennent pas dans les chemins de v_d à v_a. Nous nous appuyons encore une fois sur la matrice 'somme des chemins' *SIG*.

L'égalité $SIG[i,d]=0$ signifie qu'il n'existe aucun chemin permettant d'aller de v_d à v_i, de même l'égalité $SIG[a,i]=0$ signifie qu'il n'existe aucun chemin de v_i à v_a. Nous pouvons donc supprimer de la matrice d'antériorité A tous les arcs relatifs à ces nœuds v_i. Nous éliminons également les arcs liés aux prédécesseurs du nœud v_d ainsi qu'aux successeurs du nœud v_a qui par définition ne peuvent intervenir dans un chemin v_d-v_a. Enfin, nous éliminons les arcs reliant les prédécesseurs du nœud v_a qui ne sont pas accessibles depuis le nœud v_d et les successeurs du nœud v_d qui n'accèdent pas au nœud v_a. Ces suppressions, basées uniquement

[11] Deux chemins sont indépendants s'ils n'ont aucun arc en commun.

sur l'analyse de la matrice *SIG*, reviennent à annuler les lignes et les colonnes correspondant aux nœuds v_i concernés. L'espace de recherche se trouve ainsi considérablement restreint. Comme pour les circuits, la recherche des chemins s'effectue en profondeur dans un arbre qui admet le nœud v_d comme racine. Elle utilise deux procédures « Monter » et « Descendre » qui gèrent des pointeurs de visite. Le résultat de la recherche fournit les chemins et les vecteurs chemin associés contenus dans une structure de données *CH* et *TVH* similaire à celle qui contient les circuits (*CIR* et *TVC*, Cf. § *3.2.3*)

3.3. Exploitation de l'analyse.

Nous disposons à présent d'un ensemble d'outils d'analyse dédiés aux graphes GMT. Notre action porte sur l'interprétation des résultats d'analyse en termes d'architecture et de performance, sur les modifications que nous pouvons proposer et sur les conséquences induites sur le modèle Grafcet associé au modèle GMT analysé.

3.3.1. Architecture.

Qu'il soit issu de l'interprétation du cahier des charges ou de la traduction du modèle Grafcet, le graphe GMT d'une application peut présenter une complexité inacceptable vis-à-vis de la modularité exigée par une commande répartie. Il s'agit d'établir une partition de l'application qui adapte la taille des sous-ensembles de tâches tout en minimisant les liens entre ces sous-ensembles. Dans ce cadre, les isthmes et les points d'articulation présentent un intérêt particulier, mais la modification de l'architecture, par la duplication d'un point d'articulation ou par la suppression d'un isthme doit conserver les propriétés topologiques du graphe initial.

- *Modification par duplication d'un point d'articulation.*

Sur la *Figure 24a*, le nœud v_i représente le point d'articulation à dupliquer. La convergence sur le nœud v_i impose, sur le modèle Grafcet (*Figure 24b*), la création de deux étapes de synchronisation $q_{j\text{-}i}$ et $q_{r\text{-}i}$ en amont de l'étape q_i associée au nœud v_i.

L'étape q_i est activée dès que les deux étapes $q_{j\text{-}i}$ et $q_{r\text{-}i}$ sont actives. La réceptivité associée à la transition T_1 (en amont de q_i) étant toujours vraie, l'une des deux étapes $q_{j\text{-}i}$ ou $q_{r\text{-}i}$ est activée fugitivement (l'instabilité de cette étape a pour suite de ne pas émettre la sortie qui lui serait éventuellement associée). Le départ de séquences simultanées des étapes q_k et q_s respectivement associées aux nœuds v_k et v_s se produit dès la fin de la tâche associée à l'étape q_i par franchissement de la transition T_2.

Nous présentons sur la *Figure 24d* les conséquences de la modification d'architecture de la *Figure 24c*. L'activation de l'étape q_i est simultanée à la désactivation de l'étape $q_{j\text{-}i}$ et se

produit sur le franchissement de la transition T_3. Le franchissement de cette transition ne peut se faire que si l'étape q_{j-i} est active et que si la réceptivité Xq_{r-i}, est vraie, c'est-à-dire si l'étape q_{r-i} est aussi active (mécanisme repéré 1). L'activation de l'étape q_i provoque le franchissement de la transition T_5 dont la réceptivité devient vraie. Il s'ensuit la désactivation de l'étape q_{r-i} et l'activation de l'étape $q_{i\ bis}$ (mécanisme repéré 2). Enfin, la réceptivité associée à la transition T_4 devient vraie lorsque la tâche associée à l'étape q_i se termine. Cet événement provoque le franchissement de la transition T_4, la désactivation de l'étape q_i, l'activation de l'étape q_k, le franchissement de la transition T_6 dont la réceptivité est devenue vraie (mécanisme repéré 3) et finalement l'activation de l'étape q_s. Le chronogramme de la *Figure 25* décrit l'état des différentes variables internes au cours du franchissement des transitions.

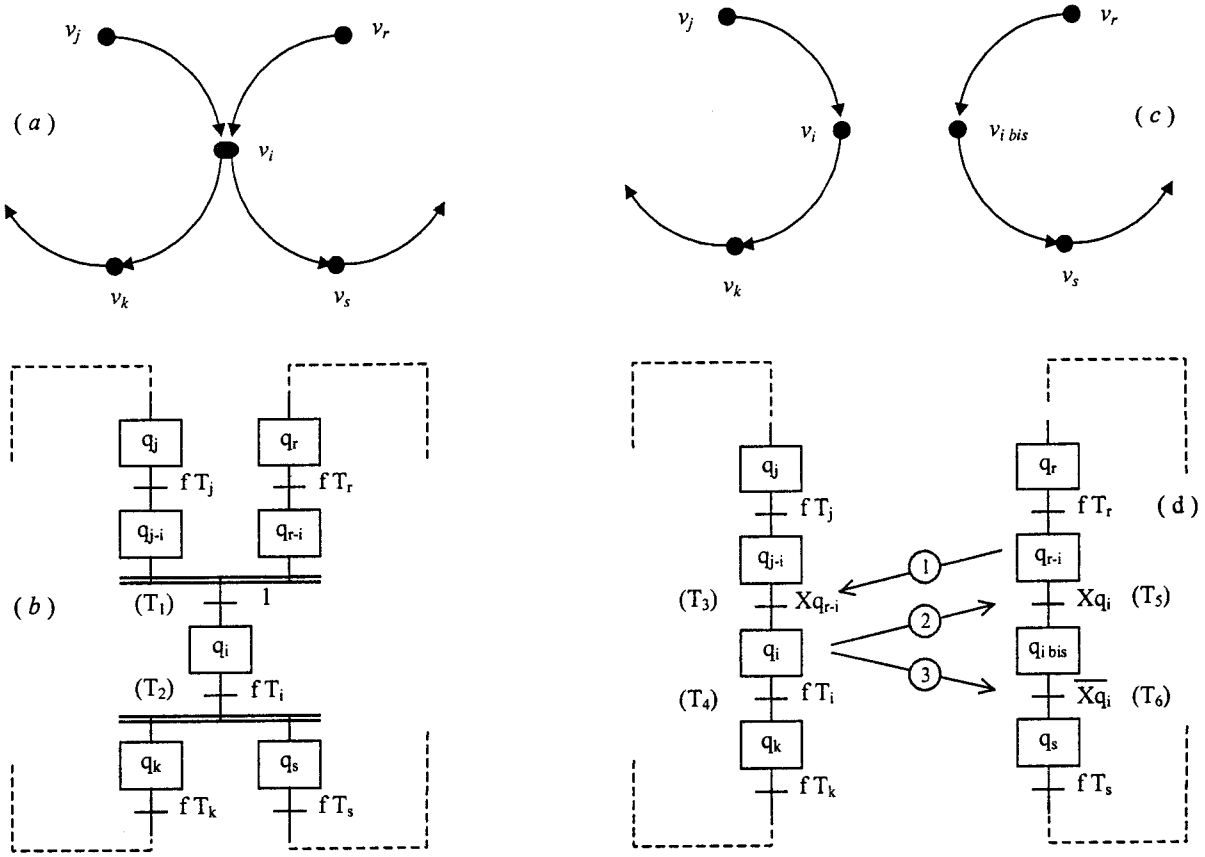


Figure 24 : Duplication d'un point d'articulation.

L'utilisation de l'état des variables internes (Xq_{j-i} et Xq_i) permet donc de respecter les mécanismes de synchronisation et d'activation simultanée de l'architecture initiale.

Conformément aux directives de DAVID, nous préférons créer une étape supplémentaire $q_{i\ bis}$ et utiliser les états *vrai* ou *faux* de la variable interne Xq_i, plutôt que le front descendant de cette variable.

Nous notons également qu'il existe une forme symétrique dans laquelle les rôles des transitions T_3 et T_4 sont respectivement inversés avec ceux des transitions T_5 et T_6.

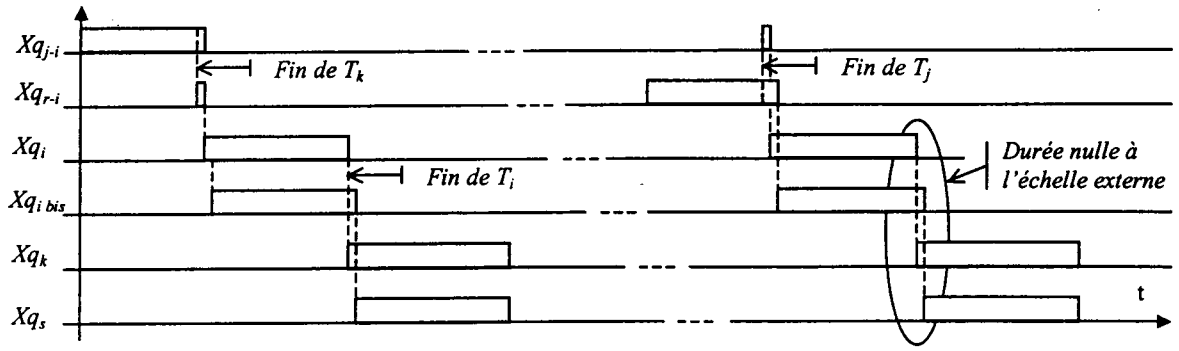


Figure 25 : Chronogramme de synchronisation et d'activations simultanées.

➢ *Modification par transformation d'un isthme.*

Sur la Figure 26*a*, le chemin v_i-v_s représente l'isthme que nous souhaitons couper en vue d'augmenter le nombre de connexité. Nous ne supprimons en fait que la fin de l'isthme, c'est-à-dire, l'arc v_u-v_s. Le départ simultané des séquences parallèles est assuré par le franchissement de la transition T_1. La convergence sur le nœud v_s impose sur le modèle Grafcet (*Figure 26b*), la création de deux étapes de synchronisation $q_{u\text{-}s}$ et $q_{r\text{-}s}$ en amont de l'étape q_s associée au nœud v_s. L'activation de l'étape q_s est issue du franchissement de la transition T_2.

Le départ de séquences parallèles autorisées par la transition T_1 n'est pas affecté par la suppression de l'arc v_u-v_s. Par contre, le respect de la convergence sur le nœud v_s nous impose de mémoriser le parcours complet de l'isthme par le marquage du nœud terminal $v_{u\text{-}s}$ (*Figure 26c*). Afin de rétablir le lien supprimé (l'arc v_u-v_s), nous utilisons une étape de mémorisation $q_{u\text{-}s}$ suivie d'une transition 'puits' (T_4)(*Figure 26d*). Cette solution qui n'est pas a priori unique, nous semble cependant d'un point de vue architecture, une des plus simples.

La transition T_3 est franchie après exécution de la dernière tâche de l'isthme. Elle active l'étape de mémorisation $q_{u\text{-}s}$, la réceptivité associée à la transition T_5 est alors vraie (mécanisme repéré 1). Si la transition T_5 est déjà valide (étape $q_{r\text{-}s}$ active) alors la transition T_5 est franchie, l'étape q_s est active et la réceptivité associée à la transition puits T_4 devient vraie, ce qui a pour effet de désactiver l'étape $q_{u\text{-}s}$ (mécanisme repéré 2): l'activité de l'étape $q_{u\text{-}s}$ est fugitive. Si au contraire l'étape $q_{r\text{-}s}$ n'est pas encore active, alors l'étape $q_{u\text{-}s}$ le restera jusqu'au franchissement de la transition T_6 : l'activité de l'étape $q_{r\text{-}s}$ sera fugitive.

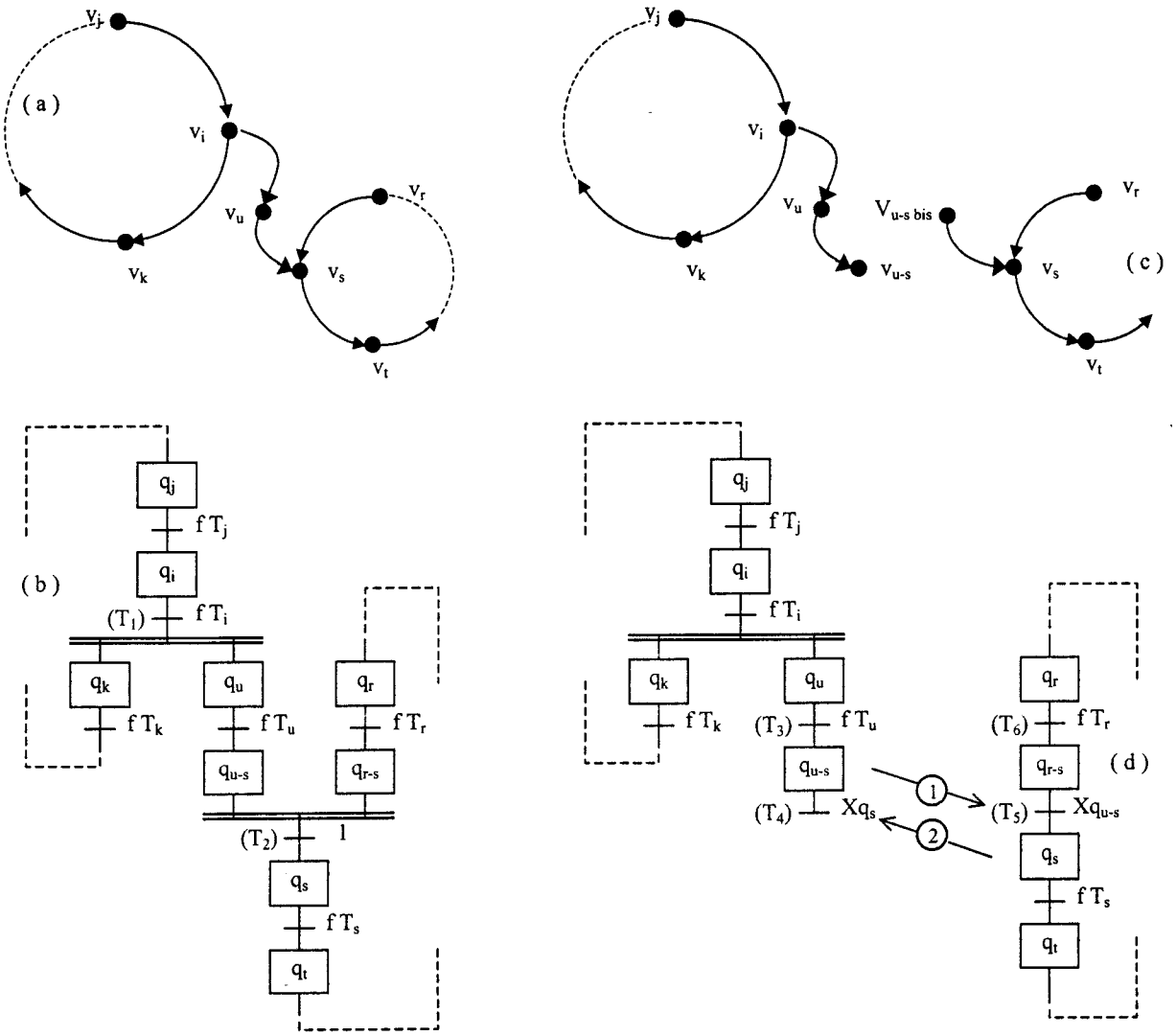


Figure 26 : Transformation d'un isthme.

Les états des variables internes lors du franchissement des transitions T_3, T_4, T_5 et T_6 sont représentés sur la *Figure 27.*

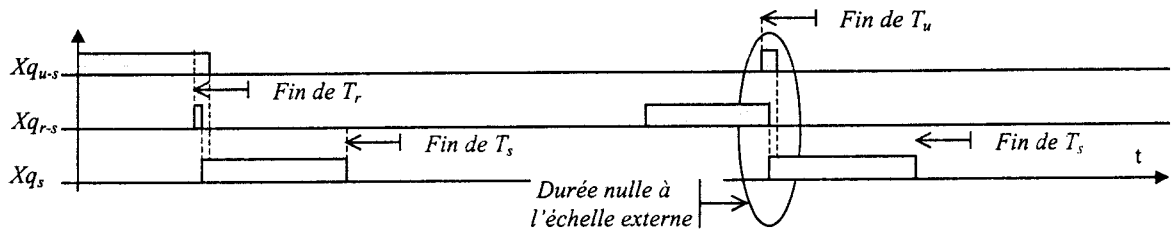


Figure 27 :Chronogramme de convergence associé à la fin d'un isthme.

3.3.2. Performances et propriétés.

➢ *Temps de cycle.*

Nous rappelons qu'un graphe GMT est un graphe non disjonctif (graphe d'événements). Après avoir dénombré les différents circuits de l'application, nous calculons pour chaque circuit, la somme des durées des tâches associées à chacun des nœuds de ce circuit. Le temps de cycle de l'application, noté *CT*, est la plus grande de ces sommes. Ce calcul n'est pas sans rappeler les opérateurs de l'algèbre MAX+ entrevus en première partie, paragraphe *2.3.4,* les graphes traités par cette algèbre sont eux aussi des graphes d'événements.

Pratiquement, les vecteurs circuits $\vec{Ci}$ sont connus par la table TVC : la coordonnée ci_j est égale à *un* si le nœud v_j appartient au circuit *Ci*, et à *zéro* dans le cas contraire. Le vecteur des durées des tâches, noté $\vec{D}$, est tel que la coordonnée d_j est égale à la durée de la tâche T_j associée au nœud v_j. Nous obtenons l'expression du temps de cycle suivante :

$$\boxed{CT = \max_i(\vec{Ci}.\vec{D})}$$

➢ *Temps d'accès au cycle le plus lent.*

En supposant que le marquage initial ait été réalisé, il est intéressant de calculer les temps Ta_{ij} d'accès de tous les nœuds initiaux v_i aux nœuds v_j du circuit le plus lent. Les vecteurs chemins $\overline{Hij}$ du nœud v_i au nœud v_j sont connus par la table *TVH*. La coordonnée hij_k est égale à un si le nœud v_k appartient au chemin v_i-v_j (sinon *0*). Nous obtenons l'expression du temps d'accès *Ta* le plus long qui est un minorant du temps transitoire.

$$\boxed{Ta = \max_{i,j}(\overline{H}_{ij}.\vec{D})} \quad \forall\ v_i \text{ nœud initial}, \forall\ v_j \in \text{circuit le plus lent.}$$

➢ *Cohérence des temps en amont et en aval d'un isthme.*

Lorsqu'un graphe comporte un isthme (Cf. *Figure 28*), nous sommes amenés à calculer deux temps de cycle associés respectivement aux parties du graphe situées en amont (C_1) et en aval (C_2) de l'isthme. Un isthme peut-être ennuyeux. Imaginons que les deux parties amont et aval du graphe correspondent à la commande de deux sous-machines, la sous-machine en amont fournissant par exemple des produits à la sous-machine en aval. Si le temps de cycle aval est plus grand que le temps de cycle amont, la machine en amont va rapidement saturer la machine en aval. Dans ce cas et pour pallier ce risque, nous remplaçons l'isthme par un circuit, en joignant par un arc supplémentaire l'arrivée de l'isthme à son origine. La sous-machine la plus rapide est alors cadencée par la sous-machine la plus lente, puisque le nœud v_i est devenu un nœud de synchronisation.

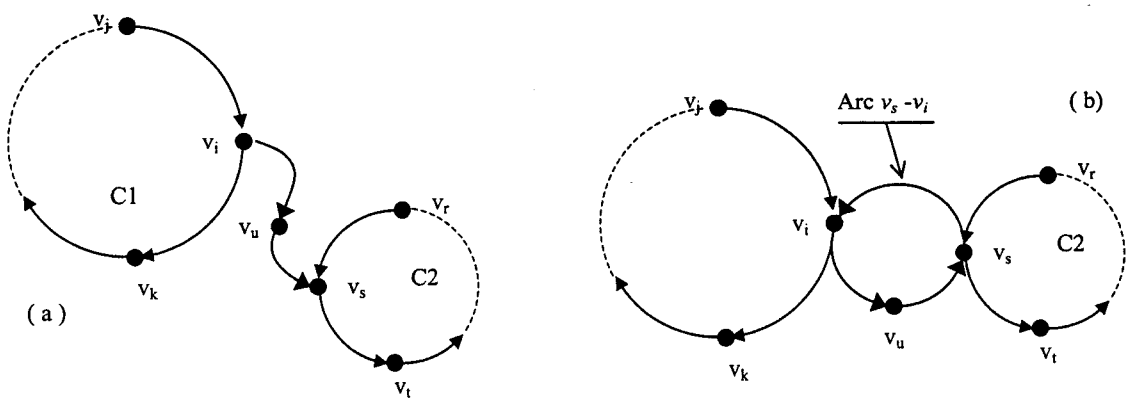


Figure 28 : Transformation d'un isthme en deux points d'articulation.

Cette vision macroscopique de l'application permet d'apprécier son bon dimensionnement, en particulier à travers l'équilibrage des temps de cycle des différentes parties de cette application.

- *Graphe vivant, graphe sauf.*

Nous rappelons qu'un graphe GMT est vivant si et seulement si le nombre de jetons de chaque circuit est positif. Un graphe GMT vivant est sauf si et seulement si chaque *sommet* du graphe appartient à un circuit dont le nombre de jetons est égal à *1*. (§ *2.2*). Nous avons également vu (§ *2.4.5.2*) que la passerelle GMT-Grafcet conservait cette propriété. Ceci signifie que si un modèle GMT est sauf, le marquage binaire du modèle Grafcet correspondant l'est également : toutes les étapes du modèle sont accessibles et aucune étape déjà active ne peut être réactivée (il n'y a pas d'absorption ou d'écrasement du marquage binaire). Cette première propriété garantit une évolution 'correcte' du modèle Grafcet.

Le second intérêt réside dans l'établissement du marquage initial du modèle Grafcet. La définition des étapes initiales d'un modèle Grafcet s'appuie généralement sur les conditions initiales de l'application : une étape est initiale parce qu'elle correspond à un état dans lequel l'application doit se trouver lors de sa mise en service. Mais ce marquage induit par les états initiaux peut être surabondant ou au contraire insuffisant, suivant le nombre et les endroits où l'on a placé ces étapes. La connaissance du marquage initial vivant et sauf du modèle GMT renseigne alors le concepteur qui peut transcrire ce marquage sur le modèle Grafcet associé ou le décliner sur ce modèle pour tenir compte d'impératifs fonctionnels.

Le marquage initial est défini par le vecteur $\overrightarrow{M_o}$ où la coordonnée $M_0(v_j)$ est à *1* si le nœud v_j est initial (*0* dans le cas contraire). Les circuits *Ci* sont connus par leurs vecteurs associés $\overrightarrow{C_i}$. Nous pouvons alors calculer pour chaque circuit le produit scalaire Vc_i tel que :

$$Vc_i = \overrightarrow{C_i} . \overrightarrow{M_o}$$

Si tous les produits *Vci* sont positifs alors le graphe GMT est vivant, mais il faut que tous les sommets du graphe appartiennent à des circuits et que ces produits soient égaux à *1* pour que le graphe soit sauf. Par définition, un graphe contenant des chemins 'source', des chemins 'puits' ou des isthmes ne peut donc pas être sauf. Nous retrouvons le problème posé par les isthmes au paragraphe précédent.

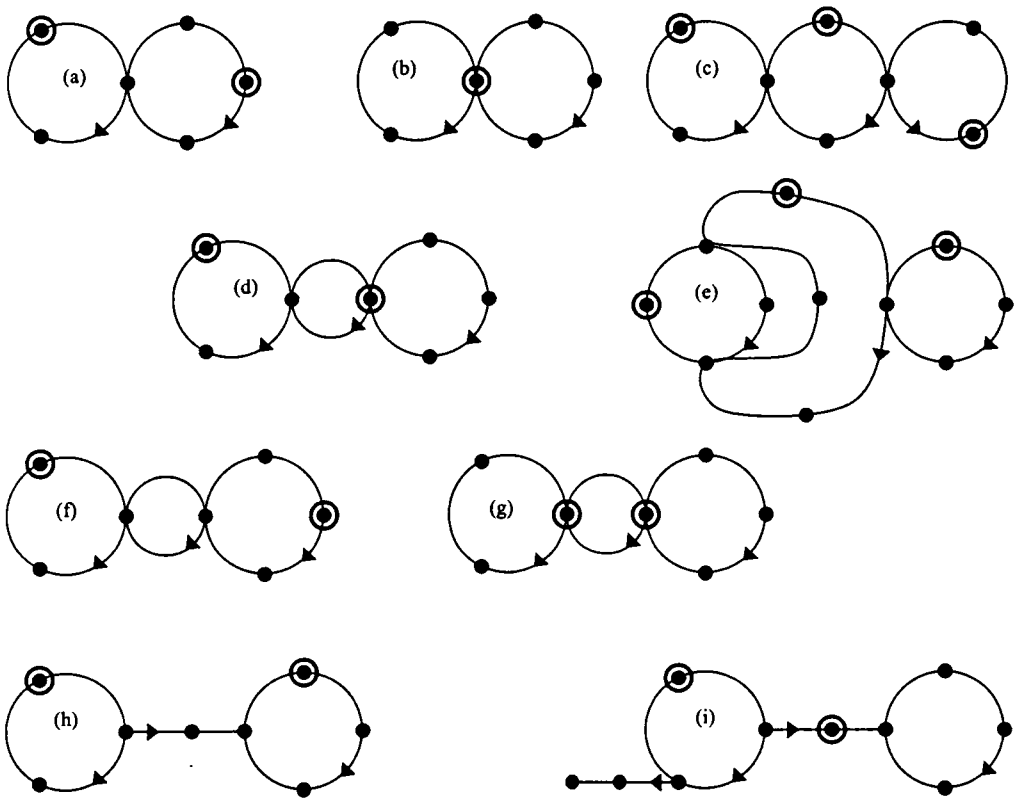


Figure 29 : Graphe sauf (a, b, c, d, e) et non saufs (f, g, h, i).

Pratiquement, le graphe d'une application peut être vu comme des circuits élémentaires assemblés au moyen de points d'articulation ou d'isthmes. Nous proposons ci-dessus quelques exemples d'établissement du marquage sauf d'un graphe GMT. Sur la *Figure 29*, les marquages (*a, b, c, d* et *e*) sont saufs, ce qui n'est pas le cas des marquages (*f et g*). Le marquage d'un point d'articulation initialise les circuits de l'articulation (cas *b* et *d*). Le graphe (*f*) n'est pas vivant et le graphe (*g*) n'est plus sauf. Le graphe (*h*) est vivant mais pas sauf. Enfin, le graphe(*i*) n'est ni vivant ni sauf.

3.4. DSM, une méthode supplémentaire.

Nous présentons à présent une méthode supplémentaire capable de travailler sur les modèles du formalisme GMT et dont l'objectif est de transformer des structures *séquentielles (SE Sequential Engineering)* en structures *parallèles (CE Concurrent Engineering)*. La méthode DSM (*Design Structure Matrix*) est issue des travaux de [DONALD V. & al., 65]. Elle permet de mettre à jour le couplage existant entre les entités d'un ensemble et éventuellement d'effectuer une partition de cet ensemble. Les entités pouvant être des tâches ou des ressources, la méthode a été récemment adaptée à des problèmes de re-engineering [SMITH RP. & al., 97], [TANG D. & al., 00] ou d'analyse de réseaux (VLAN) [KROMMENACKER N. & al., 99], (Web) [ROGERS JL. & al., 99].

La finalité de cette méthode est donc de paralléliser un ensemble de tâches en regroupant par niveaux des tâches simultanément exécutables vis à vis de leurs contraintes de précédence. Si cela est possible, la méthode ré-organise la matrice d'antériorité A sous forme triangulaire ou diagonale. Les conditions de ré-organisation font l'objet de 2 théorèmes [HARARY F., 62] (Cf. *Figure 30*):

- Théorème 1 : Une matrice carrée est irréductible si le graphe orienté associé est fortement connexe.

- Théorème 2 : Une matrice carrée est indécomposable si et seulement si le graphe associé est connexe.

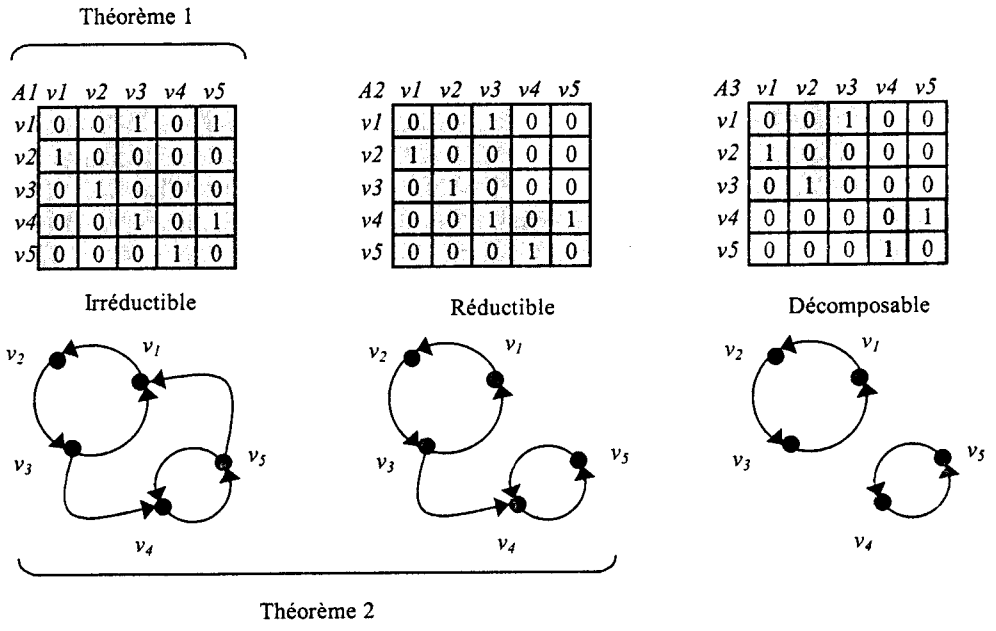


Figure 30 :Conditions de réduction selon HARARY.

Dans le cas d'une matrice réductible, la ré-organisation imposée à la matrice visualise les arcs dont la suppression peut rendre le graphe décomposable et réaliser ainsi une partition.

➢ *Présentation de la méthode.*

La méthode développée s'appuie partiellement sur les travaux exposés dans [TANG D. & al., 00] et [DONALD V. & al., 81]. Dans l'algorithme proposé, la recherche des niveaux est basée sur celle de TANG, mais la ré-organisation des tâches au sein de chaque niveau, est effectuée selon la technique de DONALD.

La matrice *A* (Cf. *Figure 32*) et la matrice 'somme des chemins' *SIG* constituent les données d'entrée de la méthode. Nous construisons une première matrice *PB* à partir de la matrice *SIG* : Si le terme SIG_{ij} est différent de zéro, alors PB_{ij} prend la valeur *un* (sinon *zéro*). Si PB_{ij} est différent de zéro, alors il existe un chemin du nœud v_j vers le nœud v_i et si PB_{ji} est différent de zéro, alors il existe un chemin du nœud v_i vers le nœud v_j. Nous construisons une seconde matrice *PP* (*Figure 31*) : le terme PP_{ij} de la matrice *PP* est égal au produit de PB_{ij} par PB_{ji}. Si PP_{ij} est différent de *zéro* alors les nœuds v_i et v_j sont fortement connectés. Les lignes identiques de la matrice *PP* sont ensuite fusionnées et les colonnes associées à ces lignes sont également fusionnées. Nous obtenons alors la matrice réduite *PR*, de dimension *(m, m)* où *m* représente le nombre de sous-ensembles ou listes au sein desquels les nœuds sont fortement connectés.

PP	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
v1	1	1	1	1	1	1	1	1	1	1	1	1	1
v2	0	1	1	1	1	1	1	1	1	1	1	1	1
v3	0	0	1	1	1	1	1	1	1	0	1	1	1
v4	0	0	0	1	1	0	0	0	0	0	0	0	0
v5	0	0	0	0	1	0	0	0	0	0	0	0	0
v6	0	0	0	1	1	1	1	1	1	0	1	1	1
v7	0	0	0	0	0	0							
v8	0	0	0	0	0	0	0	1	1	0	0	0	0
v9	0	0	0	0	0	0	0	0	1	0	0	0	0
v10	0	1	1	1	1	1	1	1	1	1	1	1	1
v11	0	0	0	0	0	0							
v12	0	0	0	0	0	0							
v13	0	0	0	0	0	0							

matrice de connexion *PP*

									v13
									v12
				v10					v11
PR	v1	v2	v3	v4	v5	v6	v7	v8	v9
v1	1	1	1	1	1	1	1	1	1
v10 v2	0	1	1	1	1	1	1	1	1
v3	0	0	1	1	1	1	1	1	1
v4	0	0	0	1	1	0	0	0	0
v5	0	0	0	0	1	0	0	0	0
v6	0	0	0	1	1	1	1	1	1
v13 v12 v11 v7									
v8	0	0	0	0	0	0	0	1	1
v9	0	0	0	0	0	0	0	0	1

matrice réduite *PR*

Figure 31 : Matrice de connexion et matrice réduite.

Le produit de cette matrice *PR*, par un vecteur « masque » de dimension m permet à la façon d'un peigne de sélectionner à chaque passage les ensembles de tâches simultanément exécutables. Toutes les composantes de ce vecteur sont initialisées à 1, puis nous annulons la ième composante de ce vecteur après chaque passage du peigne si le $i^{ème}$ élément vient d'être sélectionné par ce peigne. L'algorithme complet est présenté en annexe 1 (§ *3.3.*)

A	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12	v13
v1	0	0	0	0	0	0	0	0	0	0	0	0	0
v2	1	0	0	0	0	0	0	0	0	1	0	0	0
v3	0	1	0	0	0	0	0	0	0	0	0	0	0
v4	0	0	0	0	0	1	0	0	0	0	0	0	0
v5	0	0	1	1	0	0	0	0	0	0	0	0	0
v6	0	1	1	0	0	0	0	0	0	0	0	0	0
v7	0	0	1	0	0	1	0	0	0	0	1	0	1
v8	0	0	0	0	0	1	1	0	0	0	0	0	0
v9	0	0	0	0	0	0	1	1	0	0	0	0	0
v10	0	1	0	0	0	0	0	0	0	0	0	0	0
v11	0	0	1	0	0	0	1	0	0	0	0	0	0
v12	0	0	0	0	0	0	0	0	0	0	1	0	1
v13	0	0	0	0	0	0	0	0	0	0	0	1	0

AO	v1	v2	v10	v3	v6	v7	v11	v12	v13	v4	v8	v5	v9
v1	0	0	0	0	0	0	0	0	0	0	0	0	0
v2	1			0	0	0	0	0	0	0	0	0	0
v10	0			0	0	0	0	0	0	0	0	0	0
v3	0	1	0		0	0	0	0	0	0	0	0	0
v6	0	1	0	1	0	0	0	0	0	0	0	0	0
v7	0	0	0	1	1					0	0	0	0
v11	0	0	0	1	0					0	0	0	0
v12	0	0	0	0	0					0	0	0	0
v13	0	0	0	0	0					0	0	0	0
v4	0	0	0	0	1	0	0	0	0	0	0	0	0
v8	0	0	0	0	1	1	0	0	0	0	0	0	0
v5	0	0	0	1	0	0	0	0	0	1	0		
v9	0	0	0	0	0	1	0	0	0	0	1		

Figure 32 : Exemple d'application de la méthode DSM - d'après [TANG D. & al., 00]

La *Figure 32* représente une matrice de précédence initiale entre 13 tâches (à gauche) et le résultat de l'opération de ré-organisation (à droite). Les ensembles { 2, 10 } et {7, 11, 12, 13 } sont fortement connexes. Ces ensembles, les autres regroupements et les éléments isolés sont organisés de la gauche vers la droite dans le sens croissant du nombre des contraintes de précédence. En regroupant les nœuds fortement connectés, la méthode DSM offre au concepteur un graphe de haut niveau où les 'hyper' nœuds représentent des sous-ensembles fortement connexes du graphe initial (Cf. *Figure 33*).

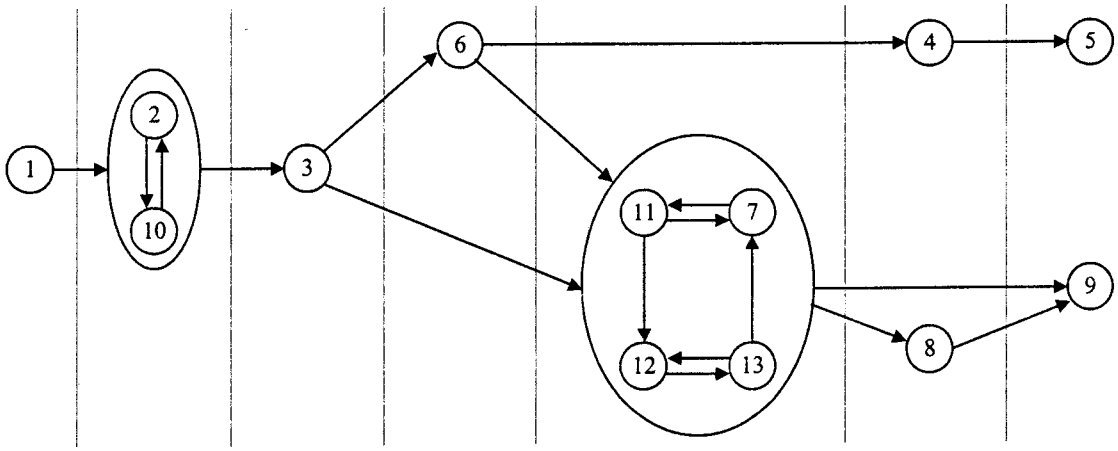


Figure 33 : Graphe associé à l'exemple d'application – [TANG D. & al., 00]

Dans un graphe GMT, une tâche est associée à chaque nœud. Ces tâches concernent la fabrication d'un produit : le graphe ré-organisé par la méthode DSM représente donc la gamme opératoire de ce produit où sont regroupées par niveau, les tâches qui doivent ou qui peuvent être simultanément exécutées. Sur l'exemple de la *Figure 33*, les tâches T_4 et T_8 (ou les tâches T_5 et T_9) peuvent être exécutées simultanément mais ce n'est pas une obligation

alors que par exemple les tâches T_2 et T_{10} ne peuvent être dissociées puisqu'elles sont fortement connectées.

Le graphe de haut niveau ainsi obtenu met en évidence les '*hyper*' nœuds auxquels seront associées des commandes localement dédiées, mais également les liens logiques existant entre ces commandes. Cette partition par niveau fait de la méthode DSM un outil précieux pour la conception de l'architecture d'une commande distribuée.

En outre, les tâches sont affectées à des ressources : il s'ensuit que ce graphe de haut niveau fournit également au concepteur des indications sur les liens physiques ou logiques existants entre ces ressources, indications utiles à la définition du réseau de transport et/ou d'informations.

3.5. Conclusion sur l'apport de la théorie des graphes.

Nous venons de définir un ensemble d'outils destiné à l'analyse du graphe GMT d'une application. Ces outils sont adaptés à l'évaluation des performances temporelles et à la vérification des propriétés '*graphe vivant*' et '*graphe sauf*'. La mise en évidence des composantes du graphe, par les outils traditionnels ou par la méthode DSM, donne au concepteur une vision globale favorisant la phase de conception préliminaire de l'application étudiée. Cette vue macroscopique guide le concepteur dans les opérations de répartition qui peuvent par exemple se faire autour d'un point d'articulation (ou autour d'un isthme en connaissance des risques encourus). La conformité des performances temporelles au cahier des charges du modèle GMT autorise alors le concepteur, par le biais de la passerelle GMT → Grafcet, à établir le modèle Grafcet de l'application.

En résumé, la théorie des graphes est un apport appréciable pour l'étude d'une application décrite dans le formalisme GMT. Nous pouvons juger ou non de la conformité de la solution étudiée au cahier des charges, mais nous n'avons jusqu'ici, aucune certitude sur l'optimalité de cette solution. Nous tentons d'y remédier au chapitre suivant, en présentant une méthode d'optimisation de l'agencement des tâches basée sur l'ordonnancement cyclique.

4. L'apport de l'ordonnancement cyclique.

4.1. Introduction.

Nous avons évoqué au paragraphe *3.2* de la première partie, la similitude entre les niveaux 0 et 1 de la classification CIM, puis la similitude entre la problématique posée par l'agencement des tâches d'un SAP et celle de l'ordonnancement cyclique d'un atelier. Nous précisons l'intérêt présenté par l'ordonnancement cyclique et les travaux déjà réalisés qui s'y rapportent.

4.1.1. Attentes fondées sur l'ordonnancement cyclique.

Il est certain qu'un SAP offre d'une façon générale des possibilités d'ordonnancement des tâches, donc de mise en œuvre des ressources, en nombre plus limité que celles d'un atelier. La flexibilité d'un SAP est plus réduite que celle d'un atelier parce que ses ressources sont davantage dédiées à des tâches spécifiques. Nous souhaitons cependant adapter les principes de l'ordonnancement cyclique à l'agencement d'un SAP afin d'atteindre plusieurs objectifs : concevoir le Grafcet de commande d'un SAP, re-concevoir le SAP à travers sa commande et prolonger au niveau du SAP (dans la mesure du possible) des efforts d'ordonnancement entrepris en amont, c'est à dire au niveau de l'atelier.

- *Concevoir le Grafcet de commande d'un SAP: une méthode supplémentaire.*

La conception d'un Grafcet de commande repose actuellement sur trois *'préceptes'* de méthodologie croissante et par ailleurs non exclusifs :

a) L'expérience et/ou l'intuition qui consiste à traduire directement le cahier de charges textuel en un modèle Grafcet, ce qui s'est fait dés l'introduction de ce formalisme en automatique séquentielle (donc au départ et sans expérience avec une bonne *'dose'* d'intuition).

b) La construction qui, après identification des problèmes *'type'* de l'application (synchronisation, partage d'une ressource, mémorisation ...etc.), assemble les solutions correspondantes (transition de convergence, étape ressource ou de mémorisation...)

c) La déduction, basée sur les antériorités qui existent au sein de la gamme de fabrication du produit et/ou qui sont imposées par le procédé (FROMENT, TAILLARD, MERLAUD).

La méthode d'ordonnancement d'un SAP, telle que nous l'entendons, s'appuie aussi sur les antériorités de ses tâches (point c), c'est à dire sur ses contraintes, mais pas forcément sur les solutions académiques (point b). Nous évoquons dans la première partie de ce mémoire (p. 59 *figure 34*), les interdits et obligations auxquels une commande doit répondre, vis à vis de son cahier des charges. Ce schéma nous a conduit à nous poser deux questions :

- Existe-t-il une solution qui soit à la fois respectueuse du cahier des charges et qui présente une productivité accrue (si ce n'est une productivité optimale) ? Si elle existe, cette solution est située quelque part dans l'espace des solutions compris entre - la possibilité de tout faire sauf ce qui est interdit – et – celle de ne faire que ce qui est obligatoire.
- Le second questionnement tient à l'interprétation du cahier des charges. Existe-t-il une organisation des tâches telle qu'on ne soit plus obligé d'interdire certaines situations ? Existe-t-il un agencement des tâches qui ne nous interdise plus de contourner certaines obligations ? C'est le second espoir que nous plaçons dans cette méthode : interpréter plus largement le cahier de charges tout en le respectant scrupuleusement.

➢ *Re-concevoir le Grafcet de commande d'un SAP.*

L'analyse du circuit critique du graphe de commande met en évidence les tâches, donc les ressources qui limitent l'application. La remise en cause de la définition des tâches de l'application porte sur les limites 'début' et 'fin' de tâches. Une tâche peut être fragmentée en deux nouvelles tâches, plus petites, mais qui feront globalement la même chose que la tâche initiale. Ce nouveau découpage permet de disposer d'informations 'à mi-chemin' qui autorisent par exemple la libération de la partie correspondante des ressources. L'ordonnancement de ce nouvel ensemble de tâches nous permet éventuellement d'augmenter la parallélisation du SAP. L'itération de cette démarche de re-conception conduit à l'instant t à une optimisation économiquement contrôlée de la commande, mais elle permet aussi au cours du temps, une adaptation de cette commande aux évolutions possibles du cahier des charges. La démarche inverse, qui vise à fusionner deux tâches successives, est moins fréquente, puisqu'elle relève d'une erreur d'analyse initiale.

➢ *Prolonger les actions d'ordonnancement de l'atelier.*

Au même titre que nous changeons l'ordonnancement de l'atelier lorsque nous changeons sur un horizon donné son plan de production, nous souhaitons changer la commande (ou adapter la commande) lorsque les produits à fabriquer par le SAP évoluent. L'adaptation à court terme répond par exemple à une évolution du ratio des produits que doit réaliser le SAP. La fabrication de variantes du ou des produits (une opération en plus ou une opération en moins) pour lequel le SAP est initialement conçu peut également entraîner la création d'un nouveau Grafcet de commande. A titre illustratif, un partage de ressource représente un interdit pour une tâche qui demanderait cette ressource, alors que celle-ci est momentanément occupée et les contraintes de précédence de la gamme 'produit' représentent des obligations. En figeant l'ordre d'allocation d'une ressource partagée pour un ratio de production donnée, puis un autre ordre pour un second ratio de production, nous supprimons le mécanisme de partage de ressource que nécessite la commande unique assurant les deux types de production: cette

solution impose deux Grafcets de commande distincts, mais chacun d'eux peut-être plus performant que le Grafcet unique (nous supprimons des interdits en éliminant la situation où les conflits pouvaient exister en particulier autour de la ressource critique).

4.1.2. Travaux relatifs à l'ordonnancement cyclique.

➢ *Positionnement de l'ordonnancement cyclique.*

Nous supposons disposer de ressources capables de réaliser les tâches de la fabrication d'un ensemble demandé de produits. L'ordonnancement consiste à déterminer le séquencement et les instants d'affectation des tâches aux ressources pour répondre à la demande de fabrication d'une quantité donnée de ces produits. Lorsque la demande est trop importante, c'est à dire lorsque le nombre de tâches induites devient trop grand, la résolution globale de l'ordonnancement pour laquelle on examine l'ensemble de ces tâches n'est plus possible. Il faut alors considérer des sous-ensembles identiques de tâches, puis procéder à l'ordonnancement de ces sous-ensembles : la réponse à la demande totale sera donnée par la répétition de cet ordonnancement qui est appelé '*ordonnancement cyclique*'.

Les méthodes d'ordonnancement cyclique actuellement développées concernent les ateliers de production flexible manufacturière (PFM), les ateliers de galvanoplastie (HSP[12]) et également certaines applications informatiques. Elles sont aussi assez nombreuses du fait de la diversité des critères que l'on cherche à optimiser : minimisation du temps total de la production (makespan), minimisation de l'encours ou encore, maximalisation du flux produit, minimisation des temps d'attente. Nous sommes dans notre contexte plus particulièrement intéressés par les méthodes d'ordonnancement cyclique des PFM, optimisant le makespan et les encours. Elles ont notamment fait (et font encore) l'objet de travaux réalisés par le groupe PFM du Laboratoire d'Automatique et d'Informatique Industrielle de LILLE (LAIL).

➢ *Présentation succincte des travaux du LAIL.*

Cette équipe a développé une première méthode qui cherche à déterminer un ordonnancement cyclique à temps de cycle optimal tout en minimisant l'en-cours. L'étape de l'ordonnancement suit une étape d'évaluation de performances qui détermine le temps de cycle optimal *CT* et une borne inférieure pour l'encours. Cette méthode consiste à placer progressivement les tâches sur les différentes ressources dans une fenêtre temporelle de longueur *CT*. La méthode initiale a évolué en fonction des politiques de placement[13] et de la considération de cet intervalle *CT*: intervalle fixe, intervalle relaxé, intervalle glissant et autorisation de

[12] Hoist Scheduling Problem.

[13] Placement 'au plus tôt' dans la gamme ou dans l'intervalle *CT*, 'au plus tard' dans l'intervalle *CT* ...

chevauchement de l'intervalle. Dans [KORBAA O., 98], l'auteur prend en compte les différentes politiques de placement et autorise les chevauchements d'intervalle. L'algorithme proposé est basé sur une recherche par faisceaux dont la profondeur est fixée par l'utilisateur : cette possibilité permet de régler la taille de la combinatoire (jusqu'à exploration totale si la profondeur est égale au nombre de tâches à ordonnancer). Cette méthode respecte le temps de cycle optimal *CT* et minimise les encours : elle considère que les produits sont associés à des ressources de transport (palettes par exemple) et elle limite le nombre d'encours présents (WIP[14]) dans le système en contrôlant le nombre de ressources de transport utilisées dans la fenêtre temporelle *CT* [KORBAA O. & al., 00]. Le parcours du faisceau est guidé par l'évaluation d'une fonction de coût[15] qui apprécie la qualité de la solution trouvée.

Très récemment, l'équipe PFM du LAIL vient de compléter son activité dans l'ordonnancement cyclique, en proposant deux approches qui autorisent l'ordonnancement des problèmes d'assemblage et/ou désassemblage. Ces travaux font actuellement l'objet des deux Thèses de LEE J. et de TROUILLET B.

➢ *Bilan.*

Notre préoccupation nous conduit généralement au traitement de gammes graphiquement non linéaires (cas d'assemblage ou de désassemblage). Or, les méthodes actuellement disponibles ont été initialement développées par l'équipe PFM du LAIL dans le cadre d'ordonnancement de gammes linéaires.

➢ *Action.*

Nous proposons de développer une méthode d'ordonnancement limitée à notre préoccupation. La complexité de ce type de problème nous a incité à introduire un aspect aléatoire dans la recherche d'un ordonnancement cyclique : parmi les métaheuristiques déjà développées pour les ordonnancements non cycliques, nous avons choisi de nous orienter vers une approche évolutive qui emprunte aux fourmis un des principes de leur comportement social. Ce choix ne signifie nullement que les approches de recherche locale (recuit simulé, recherche tabou...) ou que d'autres approches évolutives (algorithme génétique...) ne soient pas adaptées à cette problématique, mais il faut bien faire un choix. Ce choix est également guidé par les possibilités d'implémentation informatique de l'application (structure de données) et par les facultés d'adaptation de la méthode à cette représentation.

[14] Work In Process.

[15] Cette fonction prend en compte le coût des encours, les marges de gammes et les marges machines.

4.1.3. Champ d'application de la méthode.

Nous considérons les niveaux *zéro* et *un* de la classification CIM. Une machine automatisée ou un atelier comporte un ensemble de ressources utilisables par un ensemble de tâches relatives aux gammes d'un ensemble de produits. Nous précisons ci-après la nature des ressources disponibles et les notions de gamme et de linéarité d'une gamme de fabrication. Les degrés de liberté propres aux gammes sont fixés par les niveaux de décision décrits dans [KORBAA O., 98].

4.1.3.1. Nature des ressources.

Suivant la classification proposée dans [ESQUIROL P. & al., 99], nous ne considérons que les *ressources renouvelables et disjonctives* :

- une ressource est renouvelable si, après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible (par opposition aux ressources consommables).

- une ressource est disjonctive ou non partageable lorsqu'elle ne peut exécuter qu'une seule tâche à la fois (par opposition aux ressources cumulatives).

Nous supposons qu'à chaque tâche est allouée une et une seule ressource : il n'y a donc pas de flexibilité d'affectation. Mais une ressource peut séquentiellement être utilisée par plusieurs tâches.

4.1.3.2. Gamme et sous-gamme, production cyclique.

Une gamme représente l'ensemble des tâches exécutées sur un cycle. Elle peut comprendre plusieurs sous-gammes connexes. Une sous-gamme est indépendante des autres sous-gammes et les tâches d'une sous-gamme n'ont pas de lien d'antériorité avec les tâches d'une autre sous-gamme. Une sous-gamme utilise des ressources communes à une autre sous-gamme, mais elle peut aussi avoir ses propres ressources. Dans ce dernier cas '*d'indépendance totale*', les ordonnancements des sous-gammes peuvent être faits séparément les uns des autres. Les deux notions de gamme et de sous-gamme sont confondues lorsqu'une gamme ne compte qu'une sous-gamme.

La production associée à une gamme (éventuellement à un ensemble de sous-gammes) est appelée '*production cyclique*'. La production cyclique représente l'ensemble des produits finis élaborés par le système (SAP ou atelier) à chaque cycle.

4.1.3.3. Nature des gammes ou des sous-gammes, degrés de libertés.

Les tâches d'une gamme peuvent présenter des opérations des types suivants :

- La transformation : la tâche transforme à l'aide d'une ressource un produit d'entrée en un produit de sortie (usinage...apport d'une valeur ajoutée),

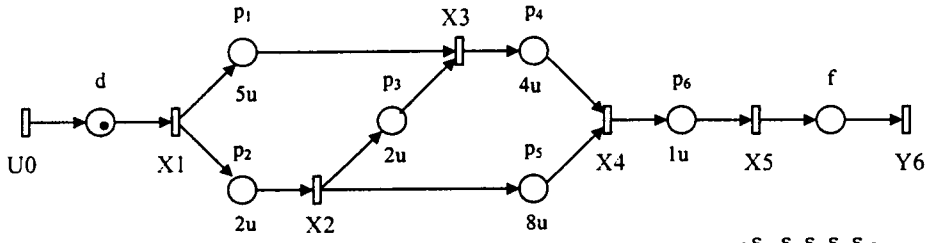
- L'assemblage : la tâche utilise une ressource pour constituer un produit de sortie à l'aide de plusieurs produits d'entrée,

- Le désassemblage : la tâche utilise une ressource pour constituer plusieurs produits de sortie à partir d'un seul produit d'entrée.

Toute autre transformation (x produits d'entrée, y produits de sortie $\langle x, y\rangle \neq \langle 1, 1\rangle$) peut se ramener à une composition des opérations de désassemblage et d'assemblage. La considération de ces types d'opérations apportées aux produits doit permettre de décrire un grand nombre de systèmes automatisés de production.

➢ *Linéarité d'une gamme.*

La représentation graphique d'une gamme comportant des opérations d'assemblage et/ou de désassemblage est *non-linéaire* : au sens de la théorie des graphes, elle n'est pas représentable par un *chemin,* mais cette notion de linéarité ne doit pas être confondue avec la linéarité des équations qui représentent le modèle dans l'algèbre des dioïdes. Nous donnons *Figure 34* un exemple d'une gamme représentée par un graphe non linéaire et qui admet cependant un modèle algébrique linéaire[16] sur $\mathbb{Z}_{max}$.



$$X(k) = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 2 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ 5 & 2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & 8 & 4 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & 1 & \varepsilon \end{pmatrix} X(k) \oplus \begin{pmatrix} e \\ \varepsilon \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix} U(k\text{-}1)$$

Figure 34 : Exemple de non-linéarité graphique et de linéarité du modèle algébrique.

➢ *Classes de décision.*

Nous adoptons la hiérarchisation des décisions proposée dans la thèse de KORBAA [KORBAA O., 98]. L'auteur limite les degrés de liberté d'un plan de production en effectuant un certain nombre de choix. Cette hiérarchisation repose sur 7 classes :

Classe D1 : choix des produits à fabriquer dans un cycle.

Classe D2 : choix des ratios de production.

Classe D3 : choix parmi les flexibilités de gamme (choix de l'ordre des opérations parmi plusieurs ordres possibles, choix pour une opération d'une ressource parmi plusieurs ressources distinctes et choix d'un procédé parmi plusieurs procédés possibles)

[16] Où $x_i(k)$ représente la date du $k^{ième}$ franchissement de la transition numéro *i*. [COHEN G., 95]

Classe D4 : choix d'une ressource particulière pour une opération parmi plusieurs ressources identiques.

Classe D5 : choix des dates d'affectation des opérations sur les ressources

Classe D6 : choix du nombre et de la répartition des encours.

Classe D7 : choix des régimes transitoires.

Nous supposons qu'à l'instant où nous calculons l'ordonnancement d'une gamme, les décisions de classes D1, D2, D3 et D4 sont déjà prises et que ces décisions font partie des *données de l'ordonnancement*. Si le produit fabriqué ou les ratios de fabrication évoluent ou plus rarement que sa gamme change (permutation d'opérations, changement de l'affectation tâche/ressource...), alors nous considérons un nouveau cahier des charges qui donne lieu à un nouvel ordonnancement.

L'ordonnancement calcule les dates d'affectation des opérations sur les ressources (Classe D5) à partir desquelles on détermine un *séquencement* d'affectation des opérations sur les ressources. Le nombre des encours et les régimes transitoires (Classes D6, D7) sont des *conséquences de l'ordonnancement*.

4.1.4. Grandes lignes de la méthode.

La méthode est basée sur l'utilisation d'un simulateur d'ordonnancement où les conflits d'allocation tâches-ressources sont arbitrés suivant des règles standards ou des règles construites à partir du comportement social des colonies de fourmis. Cette méthode comprend trois phases principales:

- Spécifications GMT et décyclage. paragraphe *4.2*
- Simulation et arbitrage. paragraphe *4.3*
- Recylage et extraction du graphe GMT. paragraphe *4.5*

Bien que notre but soit d'obtenir un ordonnancement cyclique, la première étape de la méthode commence effectivement par une opération de décyclage dans laquelle nous effectuons une modification des contraintes d'antériorités du problème initial. Ces contraintes modifiées seront rétablies dans la dernière phase.

Dans le même esprit et pour un bon entendement, nous avons donné quand nous l'avons jugé nécessaire, le RdP équivalent aux différents graphes GMT que nous utilisons. Nous n'utilisons les réseaux de PETRI que dans le but d'une meilleure compréhension.

Enfin, tout au long de cette présentation, nous avons essayé de dégager des principes généraux, puis de les appliquer pas à pas à un exemple d'application.

4.2. Spécification d'une application.

4.2.1. Etablissement d'une gamme GMT.

➢ *Principe de spécification d'une gamme GMT.*

Chaque produit possède une gamme opératoire unique à laquelle est associé un graphe de précédence du formalisme GMT : nous appelons ce graphe *une gamme GMT.* Une gamme GMT n'autorise aucun choix de procédés, l'affectation tâche/ressource est unique: il n'y a pas de flexibilité d'affectation. Elle commence par un nœud *début* et se termine par un nœud *fin.*
Ce graphe peut présenter des enchaînements séquentiels de tâches, des divergences et des convergences.

- Un enchaînement séquentiel est un tronçon de gamme linéaire où se succèdent logiquement les tâches à apporter au produit, conformément aux impératifs du procédé de fabrication.

- Une divergence est un départ de séquences parallèles. Un départ de séquences parallèles met en œuvre des ressources distinctes qui effectuent des tâches indépendantes sur des sous-produits distincts. Elle correspond par exemple à une opération de fragmentation d'un produit en plusieurs sous-produits.

- Une convergence représente la contribution de plusieurs sous-produits à l'élaboration d'un produit unique et dans ce cas, la tâche ne peut s'effectuer que si les tâches directement en amont sont terminées (exemple d'un assemblage).

Ces structures élémentaires sont présentées sur la figure ci-dessous.

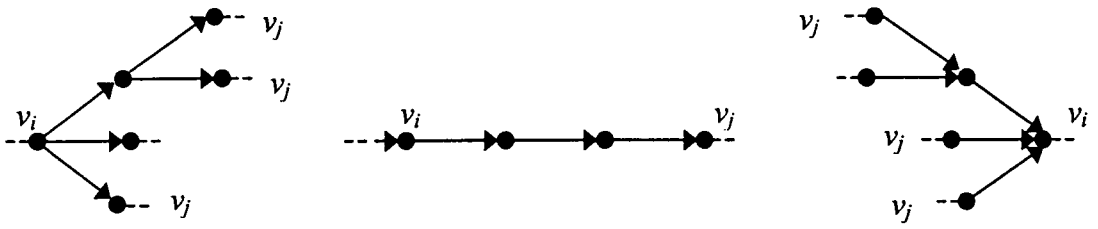
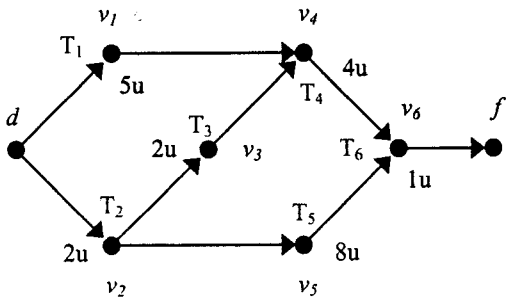


Figure 35 : Divergence, enchaînement et convergence d'une gamme GMT.

➢ *Application à un exemple.*

Les colonnes '*Entrée*' et '*Sortie*' de la *Figure 36* représentent les produits nécessaires aux tâches correspondantes pour élaborer les produits de sortie. La colonne '*Nature*' désigne le type d'opération alors que les colonnes '*Durée*' et '*Ressources*' représentent respectivement la durée et la ressource requises par chaque tâche.



Descriptif des tâches

Nature	Entrée	Nom	Sortie	Durée	Ressource
Tranformation	A	T1	A'	5	R1
Désassemblage	BC	T2	B C	2	R2
Tranformation	B	T3	B'	2	R2
Assemblage	A' B'	T4	A'B'	4	R1
Tranformation	C	T5	C'	8	R3
Assemblage	A'B' C'	T6	(A'B') C'	1	R2

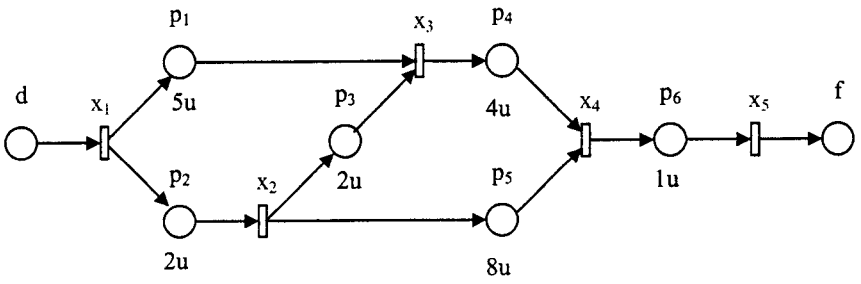


Figure 36 : Exemple de gamme GMT non linéaire.

Sur l'exemple de la *Figure 36,* la fin de la tâche T_2 autorise le lancement des tâches T_3 et T_5, la tâche T_4 ne peut être exécutée qu'après avoir terminé les tâches T_1 et T_3. La gamme GMT est un graphe qui peut être interprété comme un réseau de Petri P-Temporisé.

4.2.2. Transformée d'une gamme GMT en gamme décyclée.

Notre objectif est de générer un ordonnancement cyclique tel que l'engagement de la ressource la plus sollicitée soit maximum. Pour ce faire, nous forçons l'exécution périodique de toutes les tâches de l'application en les cadençant par l'exécution des tâches qui utilisent cette ressource. Ces actions de forçage sont consignées dans une transformée de la gamme GMT initiale, appelée *gamme décyclée.* Nous définissons au préalable la notion de ressource goulet et celle de tâches séquentiellement indissociables.

- *Définition de la ressource goulet.*

La table de description des tâches indique pour chaque ressource la durée d'exécution des tâches qu'elle exécute. On appelle '*ressource goulet Rg*', la ressource qui est la plus engagée ; Cette durée maximale d'engagement est notée *CT.*

$$CT = \max_{\forall Ressources\_Ri} \left(\sum_{\forall Gamme} durée(opération\_exécutée\_sur\_Ri) \right)$$

Lorsque la ressource *Rg* est saturée, le temps de cycle de l'application est optimal et a pour valeur *CT.* Nous considérons cette ressource unique. Dans le cas contraire, il est possible d'étudier autant d'ordonnancements que de ressources goulet existantes, mais dans chaque cas, seule cette ressource goulet serait considérée comme telle.

➢ *Tâches séquentiellement indissociables.*

Deux tâches T_i et T_j exécutées par la même ressource forment un groupe ($T_i \rightarrow T_j$) séquentiellement indissociable lorsque l'exécution de la tâche T_i ne peut être physiquement suivie que par l'exécution de la tâche T_j. La fusion de ces deux tâches en une tâche unique T_{ij} est possible, mais elle entraîne généralement la perte de l'information *'fin de la tâche T_i'*, ce qui peut être dommageable pour les performances temporelles de l'application. Supposons en effet qu'une autre tâche T_k succède également à la tâche T_i, alors le lancement de la tâche T_k ne peut plus avoir lieu à la fin de la tâche T_i mais uniquement à la fin de la tâche T_j. Dans les opérations de transformation décrite ci-après, la contrainte de précédence entre deux tâches séquentiellement indissociables ne sera jamais supprimée.

➢ *Principe d'établissement de la gamme décyclée.*

La gamme décyclée est une transformée de la gamme initiale dans laquelle nous ajoutons et supprimons des contraintes de précédence. L'ajout de contraintes (qui n'existent pas dans la gamme GMT initiale) permet d'une part de saturer la ressource goulet et d'autre part de synchroniser les autres ressources sur cette ressource goulet. La suppression des contraintes associées aux convergences permet de libérer la ressource goulet des attentes éventuelles imposées par les convergences. Toutes les contraintes supprimées seront ultérieurement rétablies. La méthode de décyclage présentée ci-après comporte quatre étapes.

Etape 1. Nous effectuons une première simulation d'une exécution unique de la gamme GMT initiale à l'aide du simulateur décrit au paragraphe *4.3* :

- La gamme représente l'ensemble de toutes les tâches nécessaires à la réalisation d'une production cyclique, y compris dans le cas où la production cyclique est fixée par un MPS[17].
- Le simulateur qui s'appuie sur l'algorithme des fourmis produit un ordonnancement optimisé au sens d'un makespan minimal (temps d'exécution total minimal). Le résultat de cette simulation est donné sous la forme d'un diagramme de GANTT initial duquel nous pouvons extraire pour l'exécution d'une production cyclique, la séquence de passage des tâches sur la ressource goulet. Cette séquence est appelée *chemin critique.*

Etape 2. On reproduit le chemin critique (séquence des tâches exécutées par le goulet)

[17] Minimal Part Set. Ensemble minimal des produits constituant la production cyclique : A titre d'exemple, MPS=2*produit A + 3*produit B + 1* produit C.

Ce chemin respecte les liens d'antériorité entre les tâches de chacune des sous-gammes de la gamme qui utilisent la ressource goulet *Rg*, bien que les tâches de différentes sous-gammes puissent être imbriquées. C'est ce chemin critique qui forcera la saturation de la ressource goulet.

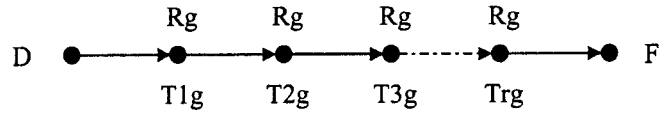


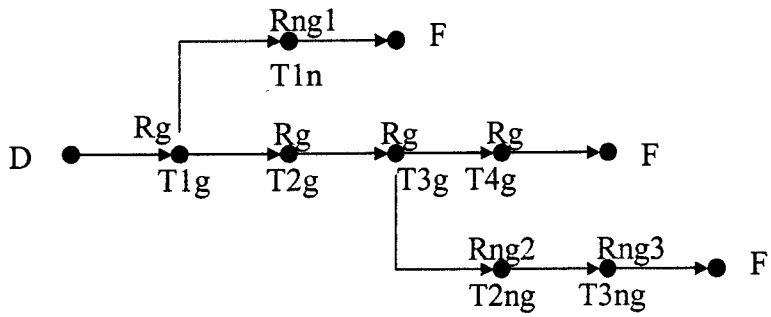
Figure 37 : Chemin critique de l'ordonnancement.

Etape 3. On cherche à rattacher au chemin critique les autres tâches de l'application :

Sous-étape 3.1. On se place sur le nœud *'départ'* du chemin critique (D).

Sous-étape 3.2. On cherche *'prochain nœud'* du chemin critique dont la tâche associée est l'origine d'un arc de la gamme GMT conduisant à un nœud non encore reproduit sur la gamme décyclée. On rattache cet arc et son nœud destination au chemin critique. On reproduit ensuite toute la partie de la gamme initiale accessible depuis ce nouveau nœud, exceptés les arcs dont la destination conduirait à un nœud du chemin critique.

Sous-étape 3.3. On ré-itère la *sous-étape 3.2*, jusqu'à ce que *'prochain nœud'* soit le *'nœud fin'* (F) du chemin critique ou que toutes les tâches soient reproduites.



Rg, Tig : Ressource goulet et tâches exécutées sur la ressource goulet.
Rngi, Ting : Ressources non goulet et tâches exécutées sur des ressources non goulet.

Figure 38 : Rattachement au chemin critique des tâches directement accessibles.

Les arcs allant des nœuds du chemin critique à des nœuds n'appartenant pas au chemin critique permettent le cadencement direct de ressources non goulet par la ressource goulet (cas des ressources Rng1 et Rng2 sur le schéma ci-dessus). La ressource Rng3 est à son tour cadencée par la ressource Rng2 (lien T2ng→T3ng) ; c'est un cadencement indirect par la ressource Rg.

Sous-étape 3.4. On ferme la gamme GMT initiale et on se place sur le nœud 'départ' du chemin critique.

Sous-étape 3.5. On cherche 'prochain nœud' du chemin critique ayant conduit à une reproduction partielle de la gamme GMT. On se place sur le premier *'nœud fin '* de la reproduction partielle de la gamme GMT.

Sous-étape 3.6. On cherche en passant par le lien de fermeture F→D la première tâche non reproduite dans la gamme décyclée. On rattache le nœud correspondant par un arc admettant comme origine le nœud précédant le nœud '*fin*' et comme destination le nœud associé à la tâche non reproduite. On reproduit ensuite toute la partie de la gamme initiale accessible depuis ce nouveau nœud, exceptés les arcs dont la destination conduirait à un nœud du chemin critique ou à des nœuds associés à des tâches déjà reproduites.

Sous-étape 3.7. S'il existe encore des tâches non reproduites dans la gamme décyclée, alors s'il existe encore un prochain '*nœud fin*' alors on se place sur le prochain '*nœud fin*' et on ré-itère la *sous-étape 3.6.* sinon on ré-itère la *sous-étape 3.5.*

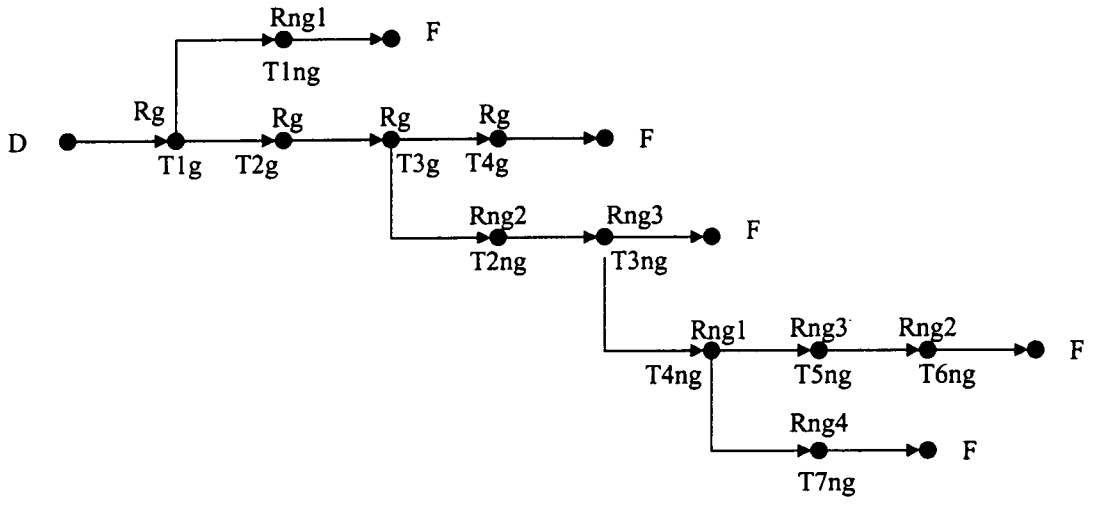


Figure 39 : Rattachement des tâches accessibles par le lien de fermeture.

A ce stade, seules les tâches n'utilisant pas la ressource goulet et appartenant à des sous-gammes indépendantes ou totalement indépendantes ne sont pas encore reproduites sur la gamme décyclée en cours de construction.

Etape 4 : On cherche à rattacher à la gamme décyclée les sous-gammes indépendantes. Les sous-gammes indépendantes sont obtenues en appliquant à ces sous-gammes la procédure décrite dans les étapes 2 & 3. On rattache la première tâche de ces sous-gammes au premier nœud du chemin critique. Enfin, le rattachement des sous-gammes totalement indépendantes n'est pas obligatoire, mais il permet néanmoins de synchroniser la production de ces sous-gammes sur la sous-gamme construite autour de la ressource goulet.

A ce stade, toutes les tâches de la gamme GMT sont reproduites et toutes les ressources sont cadencées directement ou indirectement par la ressource goulet.

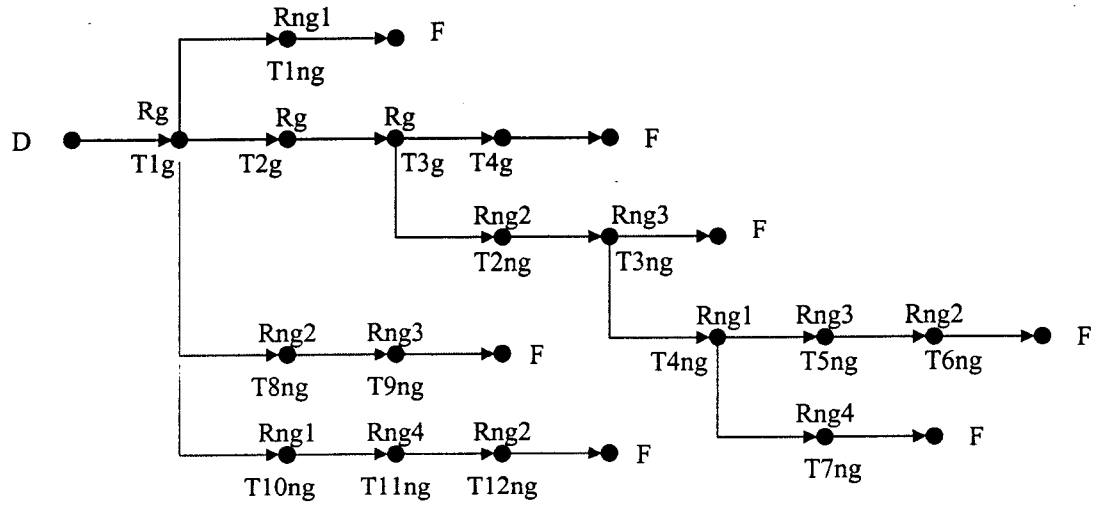


Figure 40 : Cadencement par rattachement des sous-gammes indépendantes.

Remarque 1 : Ajouts et suppressions de contraintes :

Les arcs empruntant le chemin de fermeture F→D ajoutés à la *sous-étape 3.6.* ou les arcs de rattachement des sous-gammes au premier nœud critique ajoutés à *l'étape 4* reviennent à *sur-contraindre* la gamme GMT initiale. Les ajouts du premier type (passant par le chemin de fermeture) sont nécessaires pour imposer le cadencement des tâches et éventuellement des ressources qui ne seraient pas accessibles depuis les nœuds du chemin critique. Les ajouts du second type permettent le cadencement des sous-gammes indépendantes par la ressource goulet : l'exécution indépendante de ces sous-gammes ne conduirait pas à une exécution cyclique d'ensemble. Si elles ne nuisent pas au mécanisme de cadencement, les contraintes ajoutées au problème initial peuvent être relâchées dans la phase d'extraction du graphe de commande.

Aucune convergence de la gamme GMT initiale n'est reproduite sur la gamme décyclée. Nous distinguons deux types de convergences : celles dont le nœud de convergence appartient au chemin critique et les autres. Dans le premier cas, leur maintien pourrait contraindre la ressource goulet à attendre une (ou plusieurs) tâche(s) exécutée(s) par une ressource non-goulet. Cette suppression de contraintes *(relâchement)* est une condition nécessaire pour arriver à la saturation de la ressource goulet. Dans tous les cas, ces suppressions introduisent des degrés de liberté supplémentaires permettant à l'algorithme d'effectuer sa recherche dans un espace de solutions accru. Ces contraintes supprimées du problème initial doivent être rétablies dans la phase d'extraction du graphe de commande.

Remarque 2 : Influence sur l'encours : en imposant aux autres ressources le rythme de la ressource goulet, nous imposons à l'ensemble des tâches des différentes gammes de la simulation qui utilisent ces ressources d'être également synchronisées par les tâches qui sont exécutées par le goulet. Il s'ensuit un état des encours non maîtrisé, mais stationnaire et dans le ratio des sous-produits nécessaires à l'ensemble des sous-gammes.

➢ *Application à l'exemple spécifié Figure 36.*

Le diagramme de GANTT initial nous indique que la ressource goulet est la ressource R1, pour une durée minimale *CT* de 9 unités de temps (d_1+d_4). L'application des règles de transformation à l'exemple précédent conduit à la gamme décyclée de la Figure 41.

- Le chemin critique est formé des deux tâches T_1 et T_4,

- La tâche T_1 cadence par le lien de fermeture F→D la tâche T_2 qui à son tour pousse les tâches T_3 et T_5. La tâche T_6 succède naturellement à la tâche T_5.

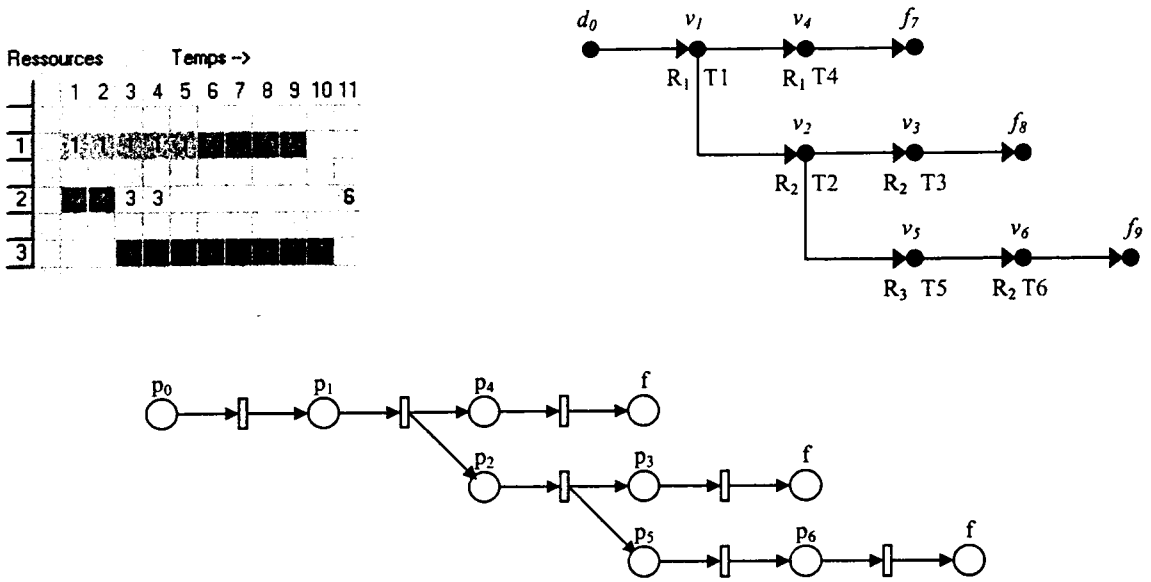


Figure 41 : Gamme décyclée et RdP associé.

Les contraintes de convergence de T_3→T_4 et T_4→T_6 n'apparaissent plus sur la gamme décyclée et nous avons ajouté la contrainte T_1→T_2 qui n'existe pas dans la gamme GMT initiale. Il faudra donc en fonctionnement cyclique et pour respecter la gamme GMT initiale, rétablir les contraintes supprimées et éventuellement relâcher la contrainte ajoutée (Cf. paragraphe *4.5*). Pour lever toute ambiguïté d'interprétation, nous donnons le réseau de Petri équivalent à la gamme décyclée.

4.3. Simulation.

4.3.1. Graphe de simulation.

➢ *Principe d'établissement.*

L'ordonnancement cyclique de la gamme GMT initiale est établi en exécutant *X* fois la gamme décyclée précédente. Cette exécution multiple repose sur l'exécution d'un nouveau graphe, appelé *'graphe de simulation'*, que nous obtenons en dupliquant *X fois* la gamme décyclée. La valeur initiale de *X* est fixée à 3 (en deçà, il est en effet difficile d'observer un phénomène cyclique). Puis elle est incrémentée jusqu'à ce que la cyclicité apparaisse. Les graphes dupliqués sont disposés en parallèle et reliés entre eux par deux nouveaux nœuds *départ* et *fin* marqués respectivement au début et à la fin de la simulation. Ce graphe est encore un graphe GMT et c'est à partir de ce graphe que nous effectuons enfin la simulation. Si on appelle *nbt* le nombre de tâches de la gamme décyclée, le nombre de tâches à réaliser sur l'horizon de production, noté *nth,* est tel que : $nth = (X * nbt) + 2$.

➢ *Application à l'exemple spécifié Figure 36.*

Nous présentons sur la *Figure 42* le graphe de simulation issu de trois duplications de la gamme décyclée. Son équivalent dans le formalisme RdP est représenté sur la *Figure 43.*

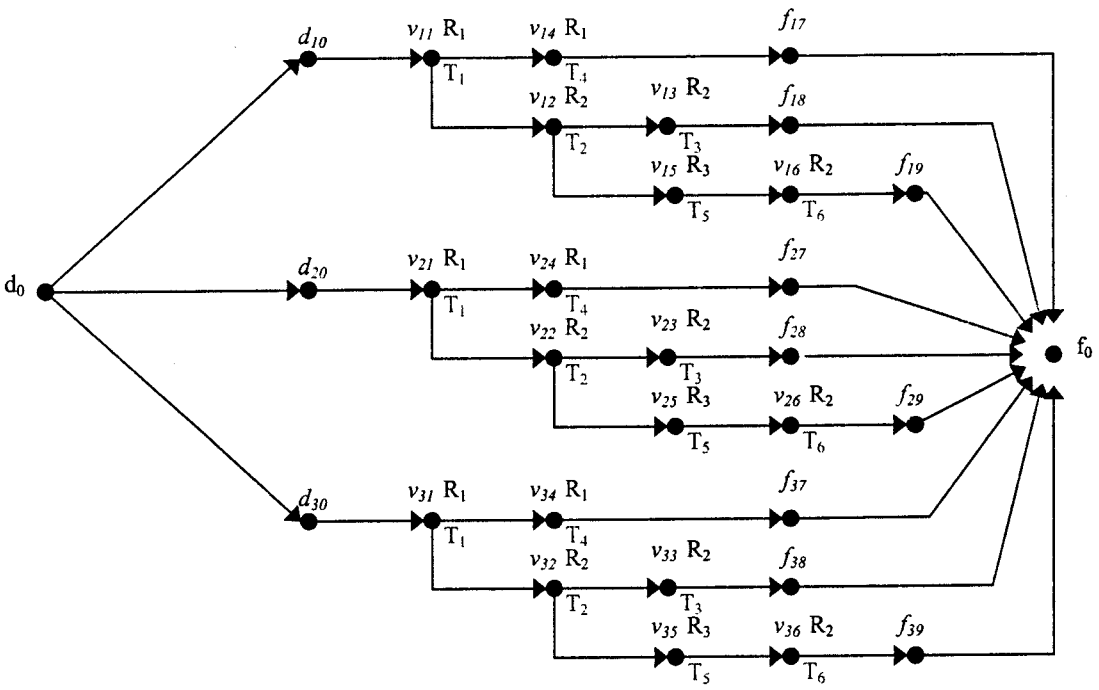


Figure 42 : Graphe GMT de simulation.

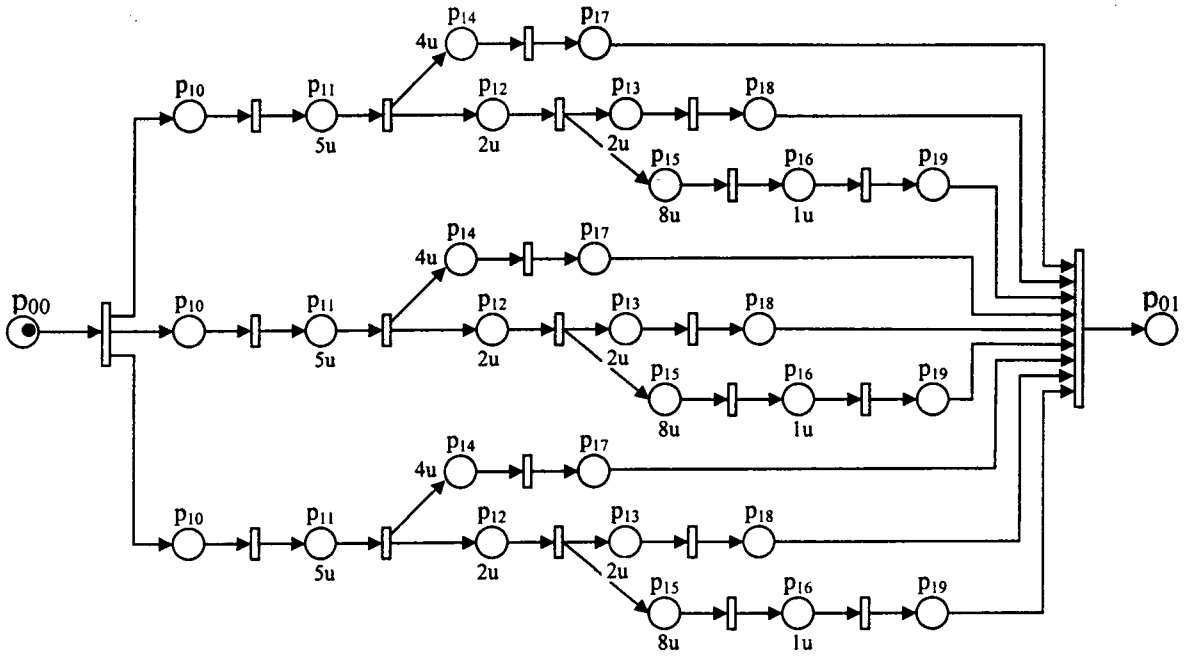


Figure 43 : RdP équivalent au graphe GMT de simulation.

4.3.2. Simulation.

La simulation consiste à faire évoluer le graphe de simulation, en respectant d'une part les règles de mise à feu des nœuds et d'autre part en évaluant les conditions associées à ces mises à feu.

Chacune de ces conditions est le produit logique de deux conditions élémentaires : l'élection et la sélection. La première porte sur la possibilité ou non d'attribuer la ressource à la tâche considérée et la seconde porte sur la possibilité ou non de mettre en service cette ressource. Dans les deux cas, nous sommes très souvent confrontés à des possibilités multiples d'attribution et/ou de mise en service : par exemple, plusieurs tâches peuvent demander simultanément la même ressource. Il importe alors de mettre en place des règles d'arbitrage permettant de gérer ces conflits (Cf. paragraphe *4.3.3*).

Nous justifions la deuxième condition élémentaire par le fait que le nombre *Xr* de ressources simultanément en service influe sur le nombre d'encours. Nous proposons au paragraphe 4.4.3, une définition de l'encours dans le cas des gammes d'assemblage et/ou de désassemblage, puis une technique d'évaluation de ce nombre à partir du résultat de simulation (diagramme de GANTT résultant). Lorsqu'on fait croître progressivement *Xr*, le temps de cycle diminue pour finalement prendre la valeur de saturation de la ressource goulet *CT*. A ce stade, l'augmentation de *Xr* n'apporte plus rien en termes de productivité, si ce n'est une augmentation néfaste du nombre des encours. En utilisant une méthode d'ordonnancement basée sur la simulation, le nombre des encours apparaît donc comme une *conséquence* de cet ordonnancement mais nous pouvons agir indirectement et de manière

qualitative sur ce nombre en limitant le nombre de ressources simultanément en service *Xr*. C'est pourquoi, en plus des conflits d'allocation tâches-ressources, nous gardons la possibilité d'arbitrer les conflits inter-ressources de mise en service, de façon à ce que le nombre de ces ressources effectivement et simultanément en service ne dépasse pas le nombre *Xr* choisi a priori (sélection, deuxième arbitrage).

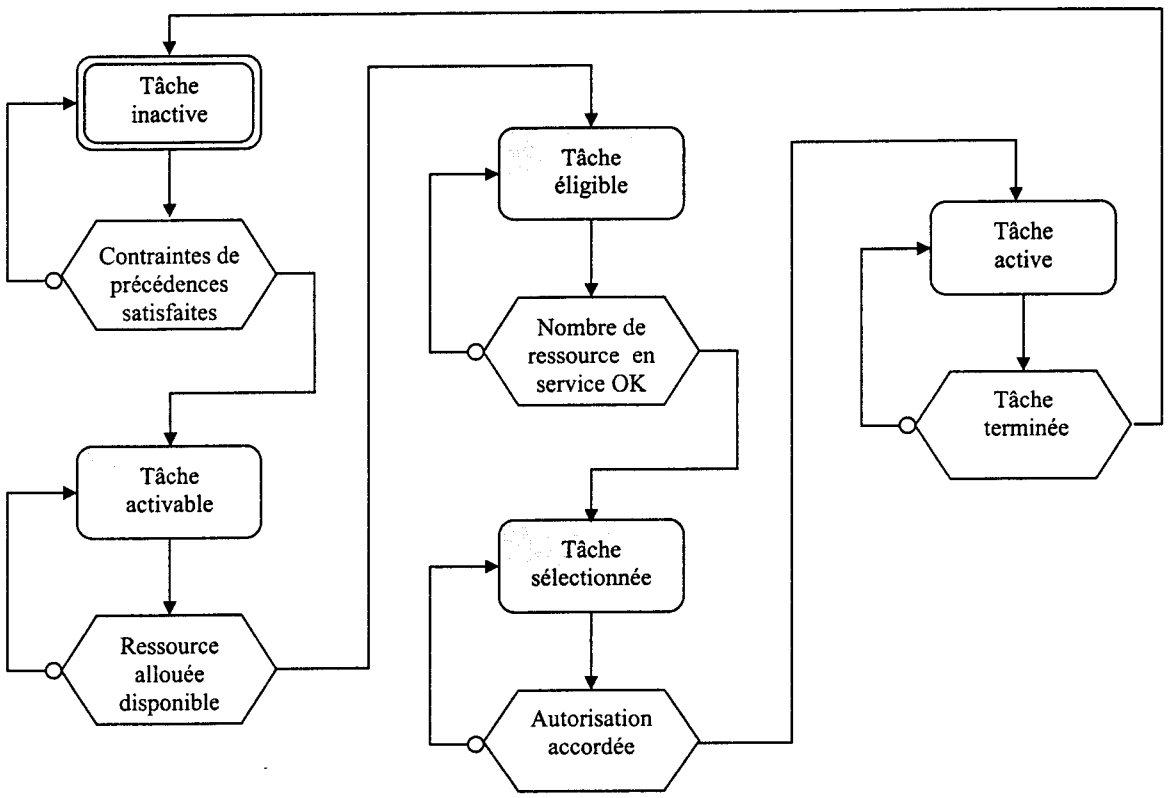


Figure 44 : Différents états d'une tâche au sein du simulateur.

L'implémentation de notre ordonnancement nous a conduit à déterminer à chaque instant, l'état dans lequel chacune des tâches peut se trouver. Les états de ces tâches sont au nombre de cinq:

- Etat inactif : Les contraintes de précédence étant non satisfaites la tâche n'est pas réalisée : le nœud associé ne peut pas être activé.

- Etat activable: La (ou les) tâche(s) antérieure(s) est (sont) terminée(s) : le nœud associé peut être activé.

- Etat éligible : La ressource allouée à la tâche activable est disponible : le nœud associé peut être activé.

- Etat sélectionné: L'occupation de la ressource par la tâche élue n'entraîne pas un dépassement du nombre *Xr* toléré des ressources simultanément en service : le nœud associé peut être activé.

- Etat actif: La tâche sélectionnée est en exécution sur la ressource allouée tant que le temps écoulé depuis son début est inférieur à sa durée : le nœud associé est activé.

Nous résumons sur la *Figure 44* les différents états et les conditions de changement d'état de chaque tâche. Le rôle du simulateur consiste donc à déterminer à chaque instant dans quel état doit se trouver chaque tâche.

Enfin, l'arbitrage des conflits fondé sur les colonies de fourmis (Cf. paragraphe *4.3.3.2*) nous a contraint à choisir une technique de simulation non pas basée sur un échéancier, mais sur la discrétisation du temps (Cf. BEL Chapitre 6 [LOPEZ P. & al., 01]). La connaissance à ces instants discrétisés de l'activité des tâches et de l'occupation des ressources impliquées nous conduisent alors au diagramme de GANTT de l'ordonnancement (Cf. paragraphe *4.4*).
L'algorithme du simulateur est fourni au chapitre *3* de l'annexe *2* relative à toutes les procédures spécifiques à l'ordonnancement.

4.3.3. Arbitrage des conflits.

Nous avons choisi de régler les deux types de conflits (élection et sélection) de deux manières :

- en utilisant deux familles de règles standards dédiées à l'élection et à la sélection,
- en utilisant un algorithme basé sur le comportement social des colonies de fourmis.

4.3.3.1. Arbitrage des conflits basé sur des règles standards.

L'arbitrage des conflits d'allocation tâches-ressources peut se faire suivant une règle de type FIFO, SPT ou LPT, respectivement First In First Out, Shortest Processing Time, Longest Processing Time. La règle FIFO choisit parmi plusieurs candidates, la tâche qui a formulé la demande de sa ressource depuis le plus longtemps. Les règles LPT et SPT allouent la ressource respectivement à la tâche de durée la plus grande ou la plus courte.
L'arbitrage entre ressources peut se faire suivant une règle de type LWK' ou GWL, respectivement Least Work - règle favorisant la ressource qui permet de traiter la tâche qui sera portée au produit le plus avancé dans son traitement ou Greatest Work Load - règle favorisant la ressource la plus chargée. LWK' est une règle dérivée de la règle LWKR (Least Work Remaining) qui affecte la plus courte tâche parmi celles qui restent à exécuter jusqu'à la fin de la gamme. Les règles standard de l'ordonnancement qui font référence à la date de livraison (EDD[18]) ne sont pas applicables en ordonnancement cyclique.

[18] Earliest Due Date - plus petite date d'échéance.

4.3.3.2. Arbitrage des conflits basé sur les fourmis.

Les principes des algorithmes de recherche basée sur le comportement des insectes sociaux [BONABEAU E. & al., 94] et plus particulièrement sur celui des colonies de fourmis [DORIGO M. & al., 91] sont récents. Lors de la réalisation d'une tâche spécifique par une colonie d'insectes, il a été observé récemment que la coordination ne dépendait pas des ouvriers mais plutôt de l'avancement de la tâche. Tout insecte, en travaillant, modifie la forme de la stimulation qui déclenche son comportement et provoque ainsi l'apparition d'une nouvelle stimulation qui déclenchera d'autres réactions chez lui ou chez l'un de ses congénères. Cette approche est classée au chapitre *3* [LOPEZ P. & al., 01] dans les approches évolutives. Le processus cyclique qui est à la base des méthodes évolutives comprend une phase de coopération et une phase d'adaptation individuelle.

Dans la phase de coopération, on examine chaque solution individuelle (parcours de la fourmi) dans le but de mettre à jour la *mémoire collective*. Dans la phase d'adaptation, une méthode constructive utilise de manière répétée cette mémoire collective pour générer de nouvelles solutions admissibles (déplacement de la fourmi). L'algorithme de la fourmi construit les solutions de façon probabiliste en misant sur l'expérience acquise.

Afin de communiquer avec leurs congénères, les fourmis déposent sur le sol de la phéromone (substance olfactive qui s'évapore avec le temps). Imaginons une colonie de fourmis à la recherche de nourriture et l'une d'elles ayant fait une bonne découverte : sur le chemin du retour menant au nid, elle en informe les autres en déposant de la phéromone sur le sol : cette trace pourra les guider vers l'endroit convoité. A leur retour, elles déposent également de la phéromone et renforcent le marquage qui va du nid à la source de nourriture. Ainsi, au fil du temps, la piste s'optimise et ce d'autant plus que la phéromone des pistes peu empruntées s'évapore. La colonie converge vers une solution optimale alors que chaque fourmi n'a qu'une information locale [LOPEZ P. & al., 01].

L'algorithme général proposé par les auteurs du chapitre référencé ci-dessus, procède à chaque instant au choix de la tâche à effectuer et au choix de la ressource à affecter à cette tâche. Nous n'avons pas à considérer ce dernier choix puisque dans notre contexte, les ressources sont affectées aux tâches de façon définitive. Par contre, nous avons vu au paragraphe *4.3.2* que nous devions faire un double choix : l'élection d'une tâche parmi les tâches éligibles et la sélection d'une ressource parmi les ressources sur lesquelles sont affectées les tâches élues. Ces choix se basent sur l'expérience que le système acquiert au fil du temps en retenant les solutions qui ont conduit pour le moment, aux meilleurs ordonnancements. L'occurrence de ces choix constitue la notion d'attrait : attrait des tâches et

attrait des ressources pour l'ordonnancement. L'originalité de la méthode proposée repose sur le fait que nous considérons que ces attraits dépendent de *l'instant de la simulation où nous effectuons ces choix*.

➢ *Principe de l'algorithme.*

La structure de données essentielle sur laquelle s'appuie cet algorithme est composée de deux tables à trois dimensions, notée *Phéro[i, u, t]* et *Copie-Phéro[i, u, t]* , où *i* représente le numéro de la tâche, *u* le numéro de la ressource considérée et *t* l'instant. Les variables *Phéro[i, u, t]* représentent l'attrait de la tâche *i* sur la ressource *u* à l'instant *t* pour l'ordonnancement. Lorsque *i* vaut *zéro*, *Phéro[0, u, t]* représente l'attrait de la ressource *u* à l'instant *t*. (Cf. *Figure 45*). Les variables *Copie-Phéro[i, u, t]*, initialement nulles sont mises à jour par les fourmis.

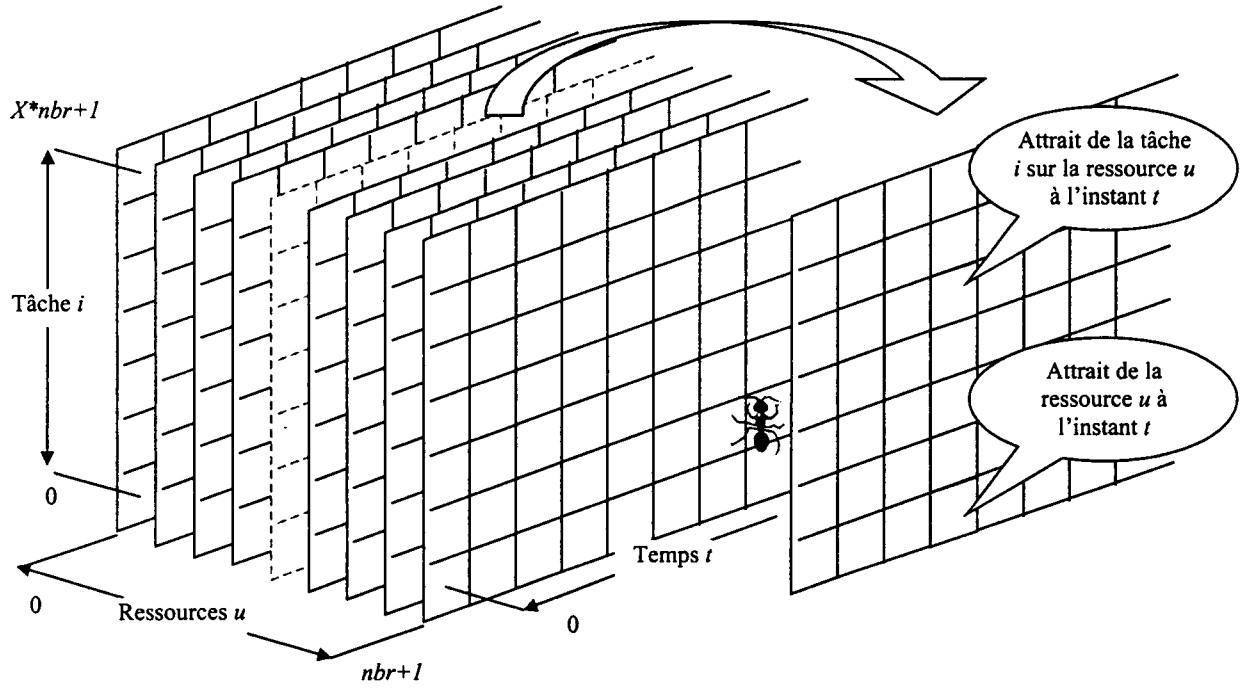


Figure 45 : Structure de données de l'algorithme d'arbitrage selon les fourmis.

Nous examinons la situation courante de l'algorithme. La $i^{ème}$ fourmi de la $j^{ième}$ colonie (ou génération) va, à l'instant *t*, procéder à deux choix :

- Il s'agit pour chaque ressource *u*, de choisir une tâche parmi les tâches activables demandant la ressource disponible *u* à l'instant *t*.

- Il s'agit aussi, si le nombre de ressources auxquelles on vient d'attribuer des tâches est supérieur au nombre toléré de ressources simultanément mises en service, de choisir les tâches qui impliquent les meilleures ressources à l'instant *t* pour l'ordonnancement.

Ces choix sont faits par deux tirages aléatoires distincts, suivant le principe de la roulette, qui se basent sur les probabilités établies par la colonie précédente (*j-1*), respectivement *Phéro[i,*

u, t] et *Phéro[0, u, t]*. Ces choix étant faits, nous les mémorisons dans la matrice *Copie-Phéro* en incrémentant les variables correspondantes :

Copie-Phéro[i, u, t] := Copie-Phéro[i, u, t] +1 et

Copie-Phéro[0, u, t] := Copie-Phéro[0, u, t] +1

A la fin du parcours de la $i^{ème}$ fourmi, nous évaluons la qualité de son ordonnancement par le temps total de la production correspondante (makespan). Ce temps ne mesure pas réellement la qualité de l'ordonnancement cyclique parce qu'il intègre le temps des régimes transitoires de début et de fin de production, mais nous pouvons atténuer cette anormalité en agrandissant l'horizon de production (nombre de duplications de la gamme GMT). Si ce temps est le plus court de tous les temps réalisés par les fourmis précédentes de cette même colonie, alors nous mémorisons le diagramme de GANTT de son ordonnancement.

Après le passage de dernière fourmi de $j^{ième}$ colonie, nous normalisons la matrice *Copie_Phéro.* Les variables *Copie-Phéro[i, u, t]* représentent alors les probabilités sur lesquelles se baseront les fourmis de la colonie suivante. Ces nouvelles probabilités sont ou non bruitées et prennent en compte ou non l'héritage des colonies antérieures. Le bruitage consiste à polluer dans un intervalle maîtrisé les probabilités de la matrice *Copie-Phéro* normalisée. Ce mécanisme nous permet de sortir d'un éventuel optimum local.

$\forall$ *i, u, t, Copie-Phéro[i, u, t] := Copie-Phéro[i, u, t]* (1+(Bruit-random(2*Bruit))*

A l'inverse l'héritage renforce la robustesse de la recherche en prenant en considération la colonie qui vient de passer mais également la précédente : avec $\alpha \in]0, 1[$

$\forall$ *i, u, t, Copie-Phéro[i, u, t] := (1-*α*) * Copie-Phéro[i, u, t]+* α** Phéro[i, u, t]*

Le processus s'arrête après un nombre *M* de cycles arbitrairement fixés. Initialement, nous fixons les probabilités du cycle zéro[19] de façon équiprobable.

Notons enfin, qu'en référence aux politiques de placement exposées par KORBAA [KORBAA O., 98] , placement au plus tôt, placement au plus tard, nous avons mis en place la possibilité aléatoire de ne pas affecter à un instant donné une tâche sur une ressource. Il est en effet possible que la non-affectation d'une tâche sur une ressource à un instant donné autorise un meilleur ordonnancement, non pas sur le critère temporel, mais du point de vue du nombre des encours.

[19] C'est sur cette matrice équiprobable que se baseront les tirages aléatoires relatifs aux conflits arbitrés par la première colonie de fourmis.

4.4. Résultats de la simulation.

4.4.1. Diagramme de GANTT exploitable.

Le diagramme de GANTT cyclique représentant l'ordonnancement arbitré par les fourmis est un résultat brut, obtenu à partir d'une gamme sur-contrainte et décyclée. Son exploitation peut être respectivement précédée par deux aménagements relatifs au relâchement éventuel de sur-contraintes et à l'exécution réelle[20] de la gamme GMT initiale. Il semble a priori préférable de procéder au relâchement des contraintes imposées par la gamme décyclée avant de s'intéresser à l'exécution réelle de la gamme GMT : on retrouve ainsi plus rapidement le problème de départ.

- *Relâchement des sur-contraintes.*

Les sur-contraintes imposées dans l'opération de décyclage sont de deux types : elles permettent d'une part de saturer la ressource goulet et d'autre part de synchroniser les autres ressources sur cette ressource goulet. Si nous souhaitons conserver la saturation de la ressource goulet, seules les contraintes du second type peuvent être relâchées. Ce relâchement correspond sur le diagramme de GANTT à un décalage à gauche des tâches non exécutées par la ressource goulet, dans la limite où sont conservés les contraintes de précédence de la gamme GMT initiale et les mécanismes de synchronisation par la ressource goulet.

- *Exécution réelle de la gamme GMT initiale.*

La première tâche ordonnancée (première tâche du chemin critique) n'a aucune raison d'être une des premières tâches de la gamme GMT initiale. Il s'ensuit que le résultat brut de la simulation (diagramme de GANTT) peut ne pas correspondre à un ordonnancement réalisable : il conduirait dans ce cas, à exécuter une tâche sans avoir réalisé les tâches qui la précédent. Pour pallier ce phénomène, nous procédons à une ré-organisation du diagramme de GANTT qui réalise dans les faits une affectation tâche/produit.

Cette opération est donc nécessaire dès lors que les premières tâches de la gamme décyclée ne sont pas les premières tâches de la gamme GMT initiale. La ré-organisation du diagramme de GANTT est décrite par la procédure ci-dessous :

Etape 1 : On recherche à partir de l'origine des temps du diagramme de GANTT issu de la simulation la première tâche correspondant à la première tâche (ou à la première des premières tâches) de la gamme GMT initiale. La date de début de cette tâche devient la nouvelle origine des temps : toutes les tâches antérieures à cette date ou toutes les tâches qui ont commencé avant cette date sont ignorées.

[20] Enchaînement complet et ordonné des tâches menant du produit brut au produit fini.

Etape 2 : On re-numérote le diagramme de GANTT en reconstruisant sur celui-ci la première gamme GMT initiale. Cette reproduction respecte TOUTES les contraintes imposées par la gamme GMT initiale : les enchaînements, les départs de séquences parallèles et les convergences, y compris celles qui avaient été relâchées dans la phase de décyclage.

Etape 3 : On recherche la première tâche de la prochaine gamme GMT initiale et on reconstruit cette gamme par une re-numérotation identique à celle de l'étape 2.

Etape 4 : On itère l'étape 3 un nombre de fois supérieur au nombre d'encours estimé.

Le diagramme de GANTT issu des éventuelles opérations de re-numérotation et de relâchement est appelé *diagramme de GANTT exploitable* et c'est à partir de ce diagramme que nous pouvons mettre en évidence le régime cyclique, les régimes transitoires et l'encours.

➢ *Application à l'exemple spécifié Figure 36.*

Le diagramme de GANTT de la *Figure 46* représente le résultat brut de l'exécution de quatre gammes décyclées, l'ordonnancement étant arbitré suivant le principe de colonies de fourmis.

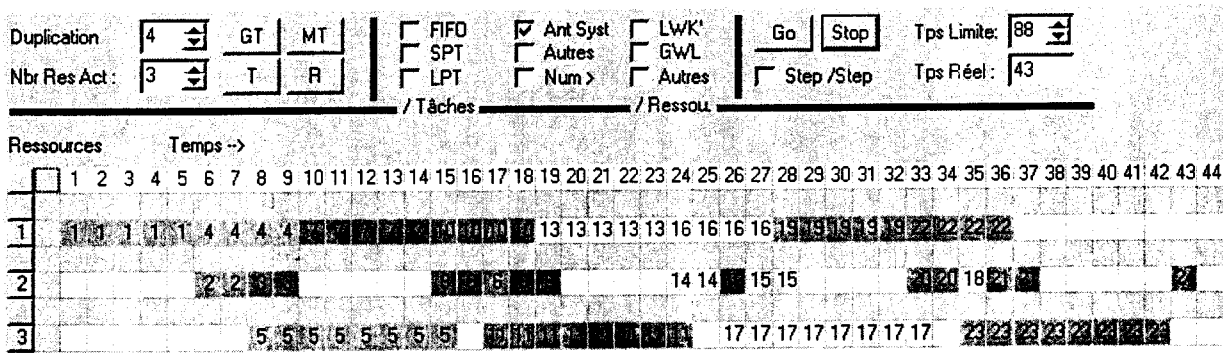


Figure 46 : Résultats bruts d'ordonnancement cyclique.

La numérotation des tâches est à considérer modulo *6* (*6* étant ici égal au nombre de tâches de cette application : la tâche n°*11* correspond à une tâche de type n°*5* du deuxième produit...etc). Une date[21] correspond au début ou à la fin d'un segment de l'échelle du temps. La contrainte $T_1 \rightarrow T_2$ imposée par la gamme décyclée pour cadencer la ressource R_2 par la ressource R_1 peut être relâchée. Ce relâchement se traduit par un décalage à gauche de cinq unités de temps de toutes les tâches exécutées par la ressource R_2. Cette ressource reste

[21] Nous établissons la relation qui existe entre une date et un segment et nous notons cette relation comme suit : la date t=0 correspond au début du segment 1, on la note [1, la date t=1 correspond au début du segment 2, noté [2, qui est aussi la fin du segment 1, noté 1]. D'une façon générale, la date t=i est notée i] ou [i+1. L'intervalle [i , j] correspond à l'intervalle de temps compris entre les dates (t=i-1) et (t=j) incluses. L'intervalle [i,j[représente l'intervalle de temps compris entre la date (i-1) incluse et la date j exclue...etc. [i] représente l'intervalle de temps compris entre les dates (t=i-1) et (t=i) incluses.

cadencée par la ressource goulet, mais par l'intermédiaire de la nouvelle contrainte $T_4 \rightarrow T_2$ qui correspond au lancement simultané et initial des tâches T_1 et T_2. Ce décalage sur la ressource R_2 entraîne un décalage identique sur la ressource R_3 (conservation de la contrainte $T_2 \rightarrow T_5$). Le relâchement de la contrainte $T_1 \rightarrow T_2$ conduit au diagramme de GANTT de la *Figure 47*. Le temps total d'exécution est réduit de *5* unités de temps et cet ordonnancement est physiquement réalisable.

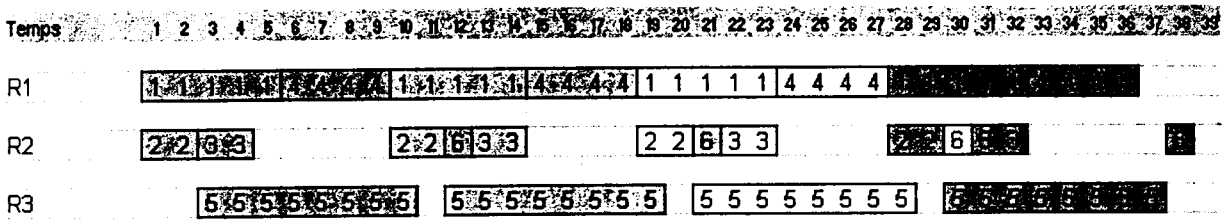


Figure 47 : Diagramme de GANTT exploitable.

De façon fortuite, la première tâche de la gamme GMT initiale (T_1) est également la première tâche de la gamme décyclée : l'ordonnancement établi par le simulateur est donc physiquement réalisable et ne nécessite pas d'opération de ré-organisation.

4.4.2. Régime permanent et régimes transitoires.

➢ *Principe de recherche.*

La découverte du régime permanent revient à déterminer la fenêtre temporelle où il est observable. Nous nous plaçons au milieu du diagramme de GANTT, puis nous explorons ce diagramme simultanément vers la droite et vers la gauche de façon à trouver respectivement le premier début ou la première fin de la première tâche. Ces deux points constituent les extrémités de la fenêtre d'observation et nous notons sa longueur *CT*. Si la ressource goulet est saturée, ce temps de cycle correspond à la somme de la durée des tâches qui lui sont affectées. Cette fenêtre d'observation est glissante, elle admet les chevauchements de tâches et sa longueur peut être relaxée en limitant le nombre de ressource simultanément actives.

La fin du régime transitoire d'entrée correspond au début du régime cyclique. Il suffit donc de faire glisser la fenêtre temporelle vers la gauche jusqu'à ce que le contenu de cette fenêtre ne soit plus cyclique. Le début du régime transitoire de sortie correspond à la fin de la dernière exécution cyclique.

En comparaison aux travaux décrits dans [KORBAA O., 98], les résultats de simulation permettent donc d'obtenir simultanément le régime cyclique et les régimes transitoires d'entrée et de sortie sans aucun calcul supplémentaire.

➢ *Application à l'exemple spécifié Figure 36.*

- La fenêtre temporelle CT.

Nous nous plaçons à la date *19]*, puis nous cherchons à la droite et à la gauche de cette date le début ou la fin d'une première tâche de type T_1 : le début des deux tâches T_1 qui encadrent cette date se trouve respectivement aux dates *18]* et *27]*. La fenêtre *CT* a donc une durée de *9* unités de temps, correspondant à la saturation de la ressource goulet.

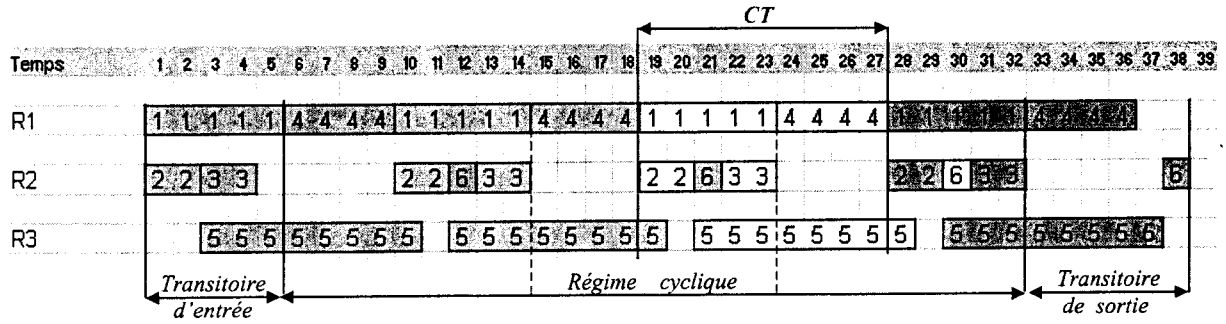


Figure 48 :Régime cyclique et régimes transitoires.

- Le régime cyclique.

Pour obtenir le début du régime cyclique, nous déplaçons la fenêtre *CT* sur la gauche tant que son contenu reste cyclique, c'est-à-dire dans notre cas, jusqu'à la date *5]*. Au-delà, le contenu n'est plus cyclique puisque la ressource R_2 est inactive sur le segment *5* alors qu'elle exécute une tâche T_3 sur le segment *14*.

Pour obtenir la fin du régime cyclique, nous reportons à partir du début du régime cyclique le contenu la première fenêtre *CT* tant que le contenu de la fenêtre reportée reste cyclique : la fin de la dernière fenêtre définit alors la fin du régime cyclique. Sur l'exemple considéré, nous pouvons reporter trois exemplaires, ce qui fixe la fin du régime cyclique à la date *32]*.

- Les régimes transitoires.

Les régimes transitoires d'entrée et de sortie sont déduits à partir du régime cyclique. Le régime transitoire d'entrée correspond à l'exécution partielle de la première production cyclique, alors que le régime transitoire de sortie correspond à l'exécution partielle de la dernière production cyclique.

4.4.3. Encours.

Le traitement de gammes non linéaires (au sens de la linéarité graphique) nous a conduit à compléter le vocabulaire existant et lié à la notion d'encours : l'encours, tel que nous allons le définir, est un facteur déterminant dans l'élaboration du graphe de commande. Nous montrons que le nombre d'encours représente le nombre minimal et nécessaire d'exécutions parallèles de gammes GMT initiales que doit conduire le graphe de commande pour assurer une exécution cyclique et saturée de l'application.

➢ *Définitions relatives à l'encours.*

Pour une gamme GMT initiale donnée et devant être exécutée de manière cyclique, nous proposons les définitions suivantes :

- Un '*produit*' est un produit brut d'entrée ou un produit fini de sortie du SAP.

- Le '*produit cyclique*' est l'ensemble des produits finis qui correspondent à la production de l'exécution d'une gamme GMT initiale (qui peut être elle-même un ensemble de sous-gammes exécutées dans un cycle).

- Un '*élément*' est un constituant momentané des produits finis du *produit cyclique.* Un élément n'existe réellement qu'à l'intérieur du SAP et il définit une classe d'équivalence. On note : c_e le nombre de classes d'équivalence de type *élément.*

ye_i le nombre d'éléments de la classe d'équivalence i, $i \in [1..c_e]$ nécessaire pour constituer la production cyclique.

$xe_i(t)$, le nombre d'éléments de la classe d'équivalence i, $i \in [1..c_e]$, réellement présents dans le SAP à l'instant t (sur les ressources ou sur les stocks inter-machine). $xe_i(t)$ est appelé *encours élémentaires*. Les éléments présents sur une ressource sont comptabilisés[22] dans l'état où ils se trouveront à la fin de l'exécution de la tâche qui occupe cette ressource, à l'exception des tâches qui élaborent un produit fini : dans ce cas, nous considérons l'état des éléments à leur entrée sur ces ressources.

- Un '*atome*' est un constituant indivisible. S'il n'est pas sécable, un atome peut être un produit brut, un élément ou un produit fini ou encore sa propre transformée (la transformation n'est ni un assemblage, ni un désassemblage). Un atome définit une classe d'équivalence. Il est l'unité commune qui permet de quantifier un produit ou un élément. On note :

c_a le nombre de classes d'équivalence de type *atome.*

$xa_i(t)$, le nombre d'atomes de la classe d'équivalence de type i, $i \in [1..c_a]$, à l'instant t, il est appelé *encours atomique.*

A l'instant t, l'*encours instantané Xe(t),* est la différence de tout ce qui est entré dans le système et de ce qui n'en est pas encore sorti, ce bilan étant dressé sans perte ni profit : en exprimant à cet instant la quantité entrée dans le SAP et celle qui en est sortie, en fonctions du nombre d'atomes, nous définissons par différence et atome par atome, les c_a encours atomiques $xa_i(t)$.

---

[22] Nous connaissons pour chaque tâche la nature et le nombre des éléments d'entrée et de sortie. Au début de l'activité de chaque tâche (date fournie par le diagramme de GANTT exploitable), nous incrémentons du nombre correspondant l'encours élémentaire de sortie et nous décrémentons de la quantité correspondante l'encours élémentaire d'entrée, excepté toute fois pour les tâches terminales où la décrémentation se fait en fin de tâche.

L'expression des c_e encours élémentaires $xe_i(t)$ en *fonction du nombre d'atomes* conduit aux mêmes encours atomiques $xa_i(t)$. $Xe(t)$ est plus grand des quotients $\lceil x_{ei}(t) / y_{ei} \rceil$ où $\lceil x \rceil$ représente la première valeur entière supérieure ou égale à x.

$$Xe(t) = max\ \forall i \in [1,ce]\ (\lceil xei(t) / yei \rceil)$$

$Xe(t)$ représente le nombre de produits cycliques en cours de fabrication présents dans le SAP à l'instant t. Finalement, nous définissons $Xe = \max_{\forall t}(Xe(t))$ comme l'encours global appelé *encours*. L'encours est le plus grand des encours instantanés quel que soit l'instant t d'une fenêtre *CT*.

➢ *Positionnement du problème - Macrogamme.*

L'examen d'un diagramme de GANTT tel que celui représenté sur la *Figure 47* montre qu'il faut considérer l'exécution simultanée d'un ensemble minimal de gammes GMT initiales, pour pallier l'impossibilité physique d'un élément (au sens de la définition donnée ci-dessus) à se trouver simultanément sur plusieurs ressources distinctes. Par exemple, sur le segment *21* de la *Figure 47* les ressources R_1 et R_3 exécutent les tâches T_1 et T_5, alors que la ressource R_2 exécute la tâche T_6. Deux éléments (A'B') et C' sont en cours d'assemblage sur cette ressource R_2 alors que l'élément C' est en cours de fabrication sur la ressource R_3 : il ne peut s'agir que d'éléments relatifs à des gammes distinctes. Le problème consiste donc à déterminer, tout au long d'une fenêtre CT, quel est le nombre de gammes qu'il faut considérer pour qu'un élément ne soit jamais simultanément sur plusieurs ressources.

Cet ensemble minimal de gammes GMT initiales est tel que son fonctionnement devient autonome et qu'il peut être répété cycliquement. Cette juxtaposition éventuellement imbriquée de gammes GMT initiales est appelée *macrogamme*.

➢ *Evaluation de l'encours suffisant.*

Posons n_e, le nombre de gammes GMT initiales qui constituent la macrogamme et définissons $Ye_i = n_e * ye_i$ le nombres d'éléments de la classe d'équivalence i, $i \in [1..c_e]$ nécessaires pour constituer la production cyclique de la macrogamme.

La solution du problème consiste à faire en sorte que, quelle que soit la classe d'équivalence d'éléments de type i et quel que soit l'instant t de la fenêtre *CT*, les éléments de cette classe ne soient au plus que sur une seule ressource. Quel soit i et t, il s'agit donc de rendre l'encours élémentaire $xe_i(t)$ inférieur ou égal à la '*capacité*[23]' correspondante Ye_i de la macrogamme. Les encours élémentaires $xe_i(t)$ étant imposés par l'ordonnancement du diagramme de

[23] *Capacité* de la macrogamme à absorber les encours élémentaires de l'ordonnancement.

GANTT cyclique, nous sommes condamnés à augmenter n_e, c'est-à-dire la taille de la macrogamme, jusqu'à ce que, quelle que soit la classe d'équivalence et quel que soit l'instant, les Ye_i soient supérieurs ou égaux[24] aux encours élémentaires $xe_i(t)$. Ce qui peut s'écrire,

$\forall i, \forall t, Ye_i \geq xe_i(t) \Rightarrow \forall i, \forall t, n_e \times ye_i \geq xe_i(t) \Rightarrow \forall i, \forall t, n_e \geq xe_i(t) \div ye_i$, avec $n_e \in \mathbb{N}$

Or, d'après la définition de Xe donnée ci-dessus, il s'ensuit que : $\boxed{n_e = Xe}$

Cette égalité est d'importance, car d'un point de vue de la conduite de l'application, elle signifie que la gestion d'une macrogamme comportant n_e gammes GMT initiales, permet également de gérer Xe encours.

➢ *Application à l'exemple spécifié Figure 36.*

Par application des définitions précédentes :

- Le *produit cyclique* est représenté par l'unique produit { (A'B')C' },
- Les *produits* bruts sont représentés par l'ensemble { A, BC },
- Les classes d'équivalence de type *élément* sont {A', B, B', C, C', (A'B')}, (c_e=6),

Les classes d'équivalence de type *atome* sont {A, B, C}, A et A' appartiennent à la classe A, de même (B etB') et (C et C') appartiennent respectivement aux classes d'équivalence de type atome B et C, (c_a=3).

Nous examinons l'encours sur la *Figure 49*. Sur les lignes *12* à *14*, nous établissons un premier bilan atomique par soustraction des produits de sortie aux produits d'entrée. Nous comptabilisons ensuite (ligne *17* à *22*), les six encours élémentaires où les nombres en italique représentent les encours élémentaires inter-machine. Nous donnons ci-dessous quelques exemples de comptage de ces éléments :

Segment 1 et 2 :

Ligne 17 : un élément A' est encours de fabrication sur la ressource R_1,

Lignes 18 et 20 : un élément B et un élément C sont en cours de fabrication sur la ressource R_2,

Segment 3 et 4 :

Ligne 17 : un élément A' est encours de fabrication sur la ressource R_1,

Ligne 19 : B' est en cours de fabrication sur la ressource R_2, B n'est plus comptabilisé,

Ligne 21 : C' est en cours de fabrication sur la ressource R_3, C n'est plus comptabilisé,

Segment 5 :

[24] L'inversion du raisonnement aurait du conduire à un comparateur du type « strictement supérieur », mais nous admettrons l'égalité, même si elle conduit à une situation tangente

Ligne 19 : la fabrication de B' est terminée, B' existe tant qu'il n'est pas consommé par la ressource R_1 : B' est sur un stock inter-machine, il est noté en italique,

.....,

Segment 12 :

Ligne 17 : un élément A' est en cours de fabrication sur la ressource R_1,

Ligne 18 : la fabrication de B est terminée, B existe tant qu'il n'est pas consommé par la ressource R_2 : B est sur un stock inter-machine,

Ligne 21 : un élément C' est en cours de fabrication sur la ressource R_3, un second élément est utilisé dans l'assemblage du produit final par la ressource R_2,

Ligne 22 : un élément (A'B') est également utilisé dans l'assemblage du produit final par la ressource R_2.

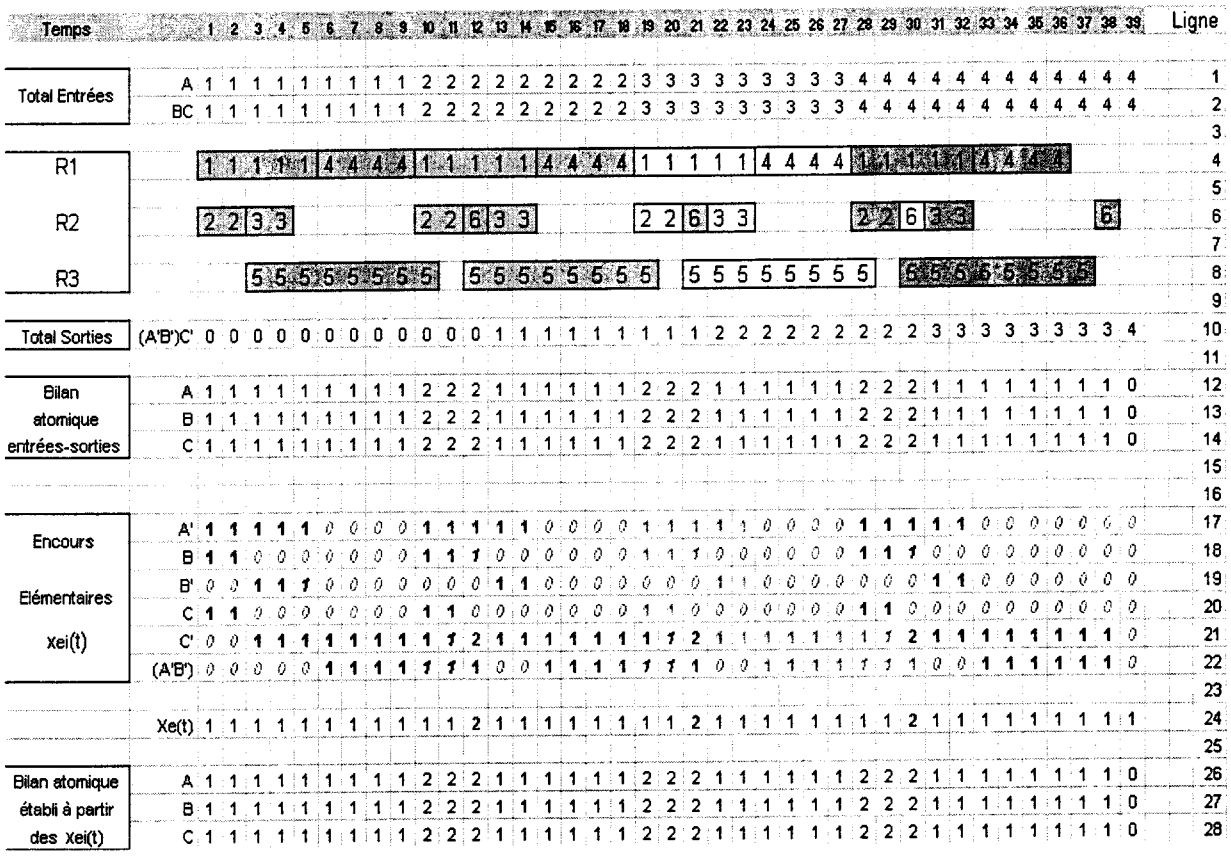


Figure 49 : Evaluation des encours élémentaires et de l'encours global.

Nous déduisons des encours élémentaires (lignes *17* à *22*) l'encours instantané (ligne *24*), puis l'encours global *Xe* qui a pour valeur ***2***. A titre de vérification, nous établissons à partir des encours élémentaires un second bilan atomique (lignes *26* à *28*) qui conduit au même résultat que le bilan établi à partir des entrées-sorties (lignes *12* à *14*).

4.4.4. Macrogamme d'une application.

La définition de la macrogamme constitue la dernière étape qui précède la phase d'extraction du graphe de commande. La macrogamme représente l'ensemble minimal de gammes GMT initiales qui autorise un fonctionnement autonome et cyclique. L'ordonnancement des tâches de toutes les gammes GMT initiales est déduit du diagramme de GANTT exploitable.

➢ *Principe de définition.*

Nous avons montré que l'encours *Xe* évalué précédemment correspond à n_e, nombre minimal et suffisant de gammes GMT initiales qui composent la macrogamme. L'origine de la macrogamme est définie par la première tâche de la première gamme GMT initiale qui appartient au régime cyclique. Afin de distinguer les tâches de même type des différentes gammes GMT initiales, nous adoptons à nouveau une numérotation des tâches telle que $Tki = (k-1)*nbt+Ti$ avec $k \in [1..n_e]$, où *Tki* représente le numéro de $i^{ème}$ tâche de la $k^{ième}$ gamme GMT initiale. La fermeture cyclique de la macrogamme est réalisée en remplaçant les dernières tâches du régime transitoire situées après l'origine de la macrogamme par les dernières tâches de la dernière gamme GMT initiale de cette macrogamme. Ainsi, la durée d'exécution *CF* de la macrogamme est égale au produit du temps de cycle *CT* par l'encours *Xe* ($CF = Xe*CT$).

➢ *Application à l'exemple spécifié Figure 36.*

L'encours a été déterminé au paragraphe *4.4.3* et a pour valeur *2* : l'origine de la macrogamme est située à la date *9]*. Nous procédons à la re-numérotation des tâches des deux premières gammes GMT initiales situées après cette origine (Cf. *Figure 50*) et la fermeture cyclique a pour effet de replacer la tâche T_{12} (tâche de type T_6 de la deuxième gamme GMT) entre les tâches T_2 et T_3 de la première gamme GMT. De même, le segment *28* qui représente la fin de l'exécution de la tâche T_{11} par la ressource R_3 peut être replacé sur le segment *10*.

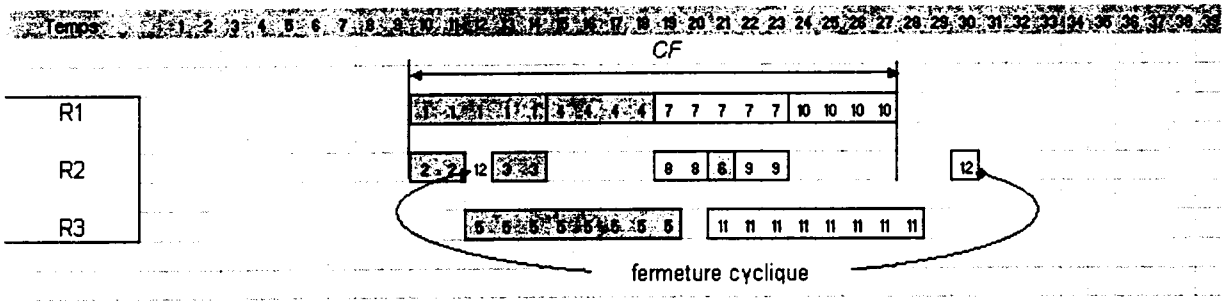


Figure 50 : Macrogamme de l'application.

4.5. Extraction du graphe et du Grafcet de commande.

Le graphe de commande est chargé de conduire l'exécution d'une macrogamme, donc de conduire l'exécution parallèle d'un nombre minimal mais suffisant de gammes GMT initiales. La préoccupation est double : il s'agit à la fois de gérer l'exécution de chacune des n_e gammes GMT initiales, mais aussi de partager l'utilisation des *nbr* ressources de l'application entre les tâches de ces n_e gammes initiales. Nous proposons deux méthodes d'extraction qui conduisent à l'élaboration d'un graphe unique et connexe ou à l'élaboration de $(nbr+n_e)$ graphes distincts associés aux *nbr* ressources et aux n_e gammes initiales. Dans les deux cas, l'extraction du graphe de commande s'appuie sur la macrogamme précédemment établie : cette dernière contient les informations relatives à la gamme GMT initiale et les informations relatives à l'ordonnancement cyclique. Nous présentons à présent ces deux méthodes d'extraction.

4.5.1. Extraction d'un graphe de commande connexe.

Le nombre de tâches de la macrogamme (noté *n*) est égal au produit du nombre *nbt* de tâches d'une gamme GMT initiale par le nombre de gammes GMT initiales qui composent cette macrogamme, soit $n = n_e*nbt$. Le principe est de consigner dans une matrice d'antériorité *A* de dimension (*n, n*) les deux types de liens qui représentent le graphe de commande :

- les liens d'antériorité propres à chaque gamme GMT initiale (enchaînement séquentiel, convergence et divergence qui existent entre les tâches d'une même gamme GMT initiale),

- les liens propres à l'utilisation et/ou au partage des ressources entre les différentes tâches des différentes gammes GMT initiales (chemin critique, séquence de passage des tâches sur les ressources non-goulet et cadencement de ces ressources par la ressource goulet).

Ces deux *familles* de liens peuvent cependant présenter des redondances. Une contrainte d'antériorité directe imposée par la gamme initiale peut être respectée indirectement par un enchaînement de contraintes issues de l'ordonnancement. A l'inverse, une contrainte directe imposée par l'ordonnancement peut également être respectée par un enchaînement de contraintes issues de la gamme initiale et/ou de l'ordonnancement.

Afin de limiter la complexité du graphe de commande, complexité directement liée au nombre de contraintes consignées dans la matrice, nous supprimons les contraintes directes qui se trouvent être respectées par un enchaînement de deux contraintes indirectes. Le nombre de contraintes indirectes peut être supérieur à deux si aucun des nœuds constituant ce chemin équivalent n'est une convergence ou une divergence.

➢ *Etablissement du graphe, construction de la matrice d'antériorité A.*

- Liens relatifs à la gamme GMT initiale.

- Enchaînement séquentiel.

S'il existe dans la gamme GMT un enchaînement séquentiel $T_i \rightarrow T_j$, alors le nœud v_i est suivi dans le graphe GMT de commande par le nœud v_j : $a_{ji} := 1$

- Convergence.

S'il existe dans la gamme GMT une convergence des tâches T_j sur la tâche T_i, alors le nœud v_i est précédé dans le graphe GMT de commande par les nœuds v_j : $a_{ij} := 1$

- Divergence.

S'il existe dans la gamme GMT une divergence à partir de la tâche T_i vers les tâches T_j, alors le nœud v_i est suivi dans le graphe GMT de commande par les nœuds v_j : $a_{ji} := 1$

- Liens relatifs à l'ordonnancement.

- Etablissement du chemin critique – fermeture cyclique.

On enchaîne dans l'ordre donné par la macrogamme la succession des tâches de la ressource critique, puis on procède à la fermeture : la dernière tâche de l'intervalle d'étude exécutée par la ressource goulet est suivie par la première tâche de cet intervalle exécutée par le goulet. Cet enchaînement est consigné dans la matrice comme un enchaînement séquentiel.

- Séquencement des ressources non-goulet par la ressource goulet.

Le séquencement des ressources non-goulet par la ressource goulet est identique à une divergence, il est donc consigné dans la matrice de la même façon.

- Suppression des redondances (limitée au rang *2*).

S'il existe une contrainte relative à la gamme GMT de type $T_i \rightarrow T_j$ (respectivement relative à l'ordonnancement) et s'il existe un enchaînement de contraintes relatives à l'ordonnancement (respectivement relatives à la gamme GMT) telles que $T_i \rightarrow T_k$ et $T_k \rightarrow T_j$, alors nous supprimons la contrainte redondante $T_i \rightarrow T_j$. Pratiquement, nous élevons la matrice précédemment constituée au carré (pour les enchaînements de deux contraintes directes) : si $(a_{ji} \neq 0) \wedge (a_{ji}^2 = \sum_{k=1}^{k=n} a_{jk} * a_{ki} \neq 0)$ alors il existe un enchaînement de rang *2* équivalent à une contrainte directe de $T_i \rightarrow T_j$ qui peut être supprimée.

- Etat initial et prise en compte du régime transitoire.

Nous n'avons pas réussi à dégager un mécanisme général permettant de prendre en compte le régime transitoire, c'est à dire la non-exécution initiale des tâches appartenant à la (ou aux) dernière(s) gamme(s) GMT initiale(s) de la macrogamme. Cependant, un saut d'étape conditionné par un booléen témoin du régime transitoire, semble généralement convenir.

➢ *Application à l'exemple spécifié Figure 36.*

Nous consignons dans les matrices *A1* et *A2* les deux types de contraintes relatives aux antériorités de gammes et aux antériorités de l'ordonnancement. (Cf. *Figure 51*)

Les deux groupes cerclés de la matrice *A1* représentent les contraintes des deux gammes GMT initiales de la macrogamme.

Les groupes 'Ri' de la matrice *A2* représentent la séquence d'allocation des ressources à chacune des tâches des deux gammes initiales. Par contre, l'allocation de la ressource R_3 à la tâche T_5 ou à la tâche T_{11} ne nécessite pas d'être consignée, car cette ressource est directement cadencée par la tâche T_2 et respectivement T_8 (ce cadencement est également imposé par une contrainte de la gamme GMT initiale). Les termes '*Cij*' de cette même matrice correspondent aux cadencement des ressources R_j par les ressources R_i.

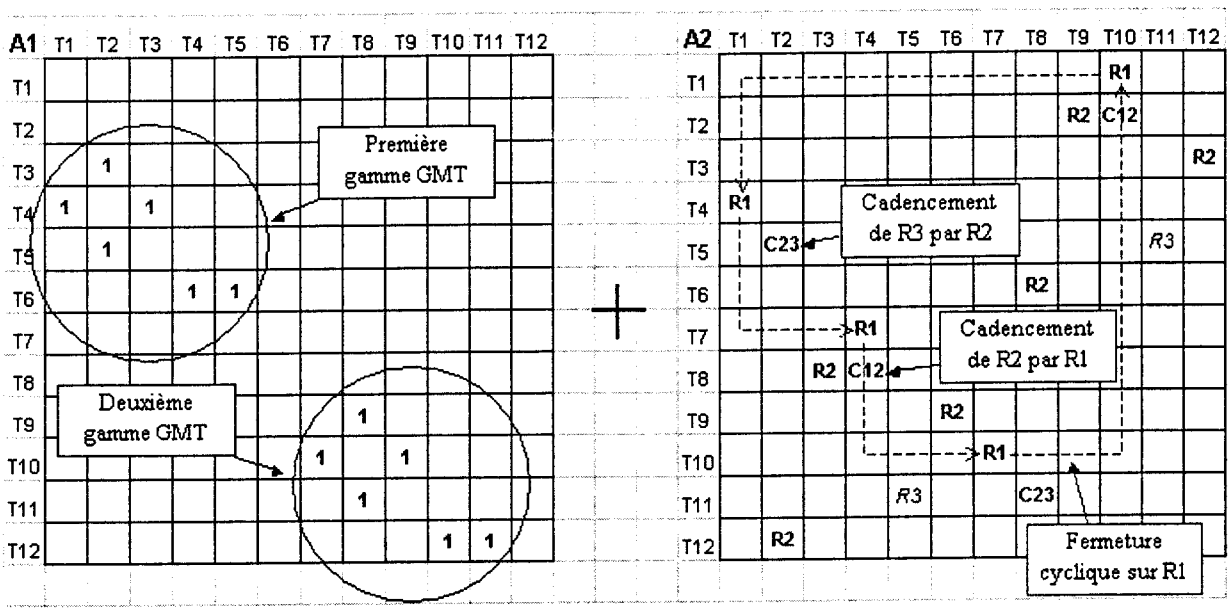


Figure 51 : Liens relatifs aux gammes (A1) et liens relatifs à l'ordonnancement (A2).

Nous effectuons la somme des deux matrices *A1* et *A2* (à gauche sur la *Figure 52*) : certaines contraintes sont présentes dans les deux matrices. Puis nous élevons la matrice 'somme' au carré afin de mettre en évidence les liens supprimables (à droite sur la *Figure 52*). Si un élément a_{ij} de la matrice A est différent de zéro et si le terme correspondant a^2_{ij} de la matrice A^2 est également différent de zéro alors le terme a_{ij} peut être supprimé de la matrice A. Ainsi, le lien direct $T_2 \rightarrow T_3$, issu de la gamme GMT initiale est remplacé par l'enchaînement des deux liens $T_2 \rightarrow T_{12}$ et $T_{12} \rightarrow T_3$ issus de l'ordonnancement. De la même manière, nous supprimons les liens $T_9 \rightarrow T_2$, $T_3 \rightarrow T_8$, $T_4 \rightarrow T_6$, $T_8 \rightarrow T_9$ et $T_{10} \rightarrow T_{12}$.

A	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
T1	0	0	0	0	0	0	0	0	0	1	0	0
T2	0	0	0	0	0	0	0	0	1	1	0	0
T3	0	1	0	0	0	0	0	0	0	0	0	1
T4	1	0	1	0	0	0	0	0	0	0	0	0
T5	0	1	0	0	0	0	0	0	0	0	0	0
T6	0	0	0	1	1	0	0	1	0	0	0	0
T7	0	0	0	1	0	0	0	0	0	0	0	0
T8	0	0	1	1	0	0	0	0	0	0	0	0
T9	0	0	0	0	0	1	0	1	0	0	0	0
T10	0	0	0	0	0	0	1	0	1	0	0	0
T11	0	0	0	0	0	0	0	1	0	0	0	0
T12	0	1	0	0	0	0	0	0	0	1	1	0

A^2	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
T1	0	0	0	0	0	0	1	0	1	0	0	0
T2	0	0	0	0	0	1	1	1	1	0	0	0
T3	0	1	0	0	0	0	0	0	1	2	1	0
T4	0	1	0	0	0	0	0	0	0	1	0	1
T5	0	0	0	0	0	0	0	0	1	1	0	0
T6	1	1	2	1	0	0	0	0	0	0	0	0
T7	1	0	1	0	0	0	0	0	0	0	0	0
T8	1	1	1	0	0	0	0	0	0	0	0	1
T9	0	0	1	2	1	0	0	1	0	0	0	0
T10	0	0	0	1	0	1	0	1	0	0	0	0
T11	0	0	1	1	0	0	0	0	0	0	0	0
T12	0	0	0	0	0	0	1	1	2	1	0	0

Figure 52 : Matrice d'antériorité A et son élévation au carré.

La matrice d'antériorité et le graphe GMT de commande sont donnés sur la *Figure 53*. Le graphe est étudié à l'aide des outils développés au chapitre *3.3.2*. Ces outils nous permettent de vérifier les propriétés de célérité et de vivacité et d'analyser la structure du graphe (Cf. *Figure 54*). Aucun des circuits ne présente un temps de cycle supérieur à *18 unités de temps*. Ce graphe comporte 1 circuit de rang 4 associé à la ressource goulet de durée *18 unités de temps,* 2 circuits de rang *5 (17 unités de temps)*, 2 circuits de rang *6* et 1 circuit de rang *8*. Ces trois derniers circuits d'une durée égale au circuit critique (*18 ut*) sont le résultat de la superposition des contraintes de gammes aux contraintes d'ordonnancement.

A	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
T1										1		
T2										1		
T3												1
T4	1		1									
T5		1										
T6					1			1				
T7				1								
T8				1								
T9						1						
T10							1		1			
T11								1				
T12		1									1	

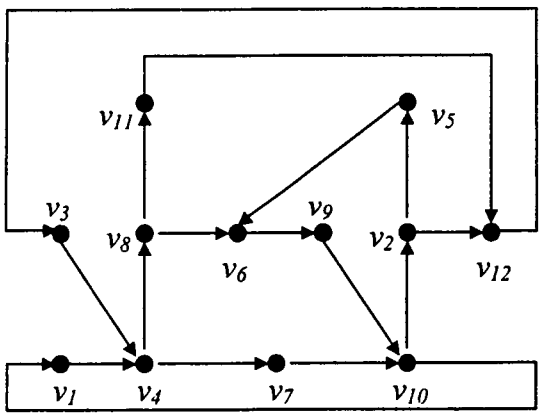


Figure 53 : Matrice et graphe connexe GMT de commande.

Le graphe est vivant pour le marquage initial du couple de nœuds v_{10} et v_{11} (ou pour le couple v_4 *et* v_5) : il est également sauf puisque tous les sommets appartiennent à des circuits. Ce

graphe est planaire et de connexité '*un*'. Il ne comporte ni puits, ni source, ni point d'articulation, ni isthme.

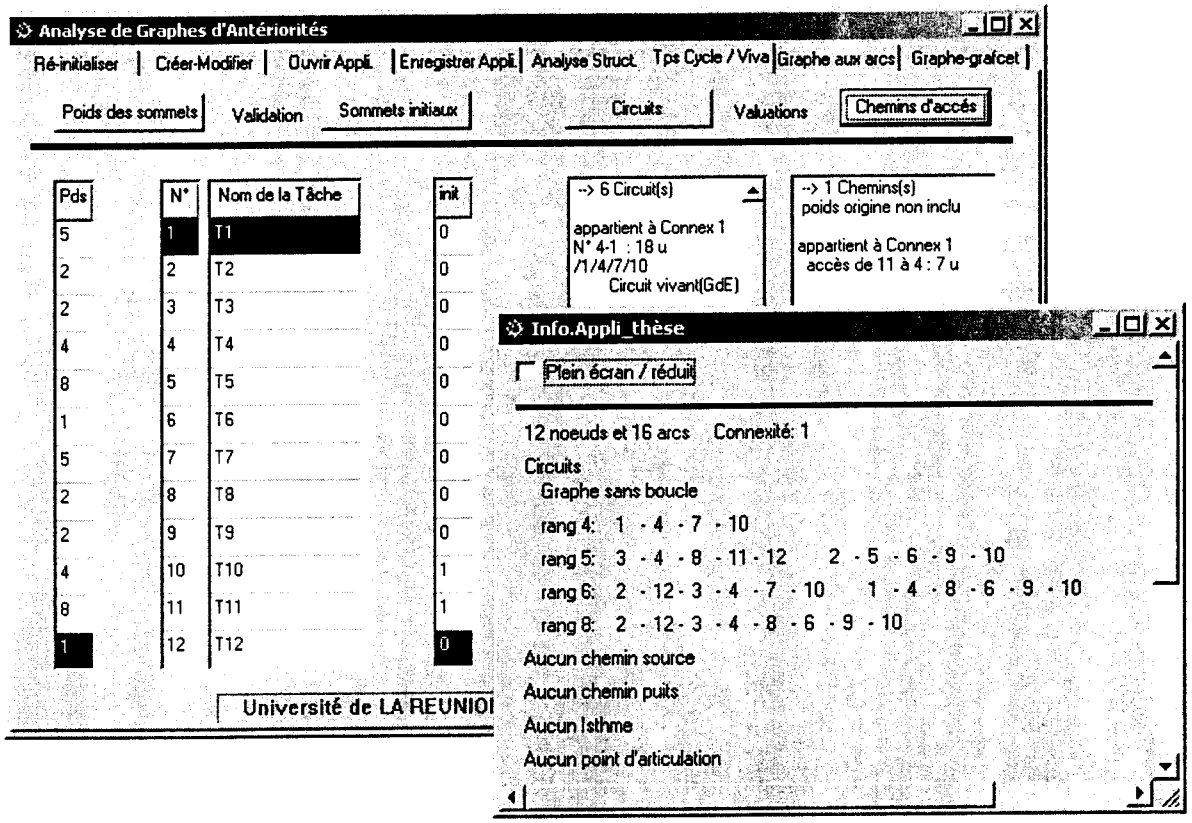


Figure 54 : Résultats d'analyse du graphe de commande.

➢ *Obtention du Grafcet de commande.*

La passerelle GMT→Grafcet définie au paragraphe *2.4* fournit l'architecture du Grafcet de commande de l'application (Cf. *Figure 55*) à l'exception toutefois de la structure de saut associée à l'étape (*12*). Le circuit (*1*), (*1-4*), (*4*), (*7*), (*7-10*), (*10*) représente l'activité de la ressource goulet R_1.

Les étapes initiales sont déduites du marquage initial du graphe GMT (nœuds v_{10} et v_{11}) :

- Le marquage du nœud v_{10} est remplacé par l'initialisation des étapes (*1*) et (*2*), dont les nœuds respectifs sur le graphe GMT lui succèdent directement. Cette modification conserve la propriété de vivacité tout en étant conforme à la réalité de l'exécution de l'application.

- Le marquage du nœud v_{11} est remplacé par l'initialisation de l'étape de synchronisation (*11-12*). Là encore, la propriété de vivacité est conservée puisque ce nœud appartient au même circuit. Le booléen (noté *trans* pour *transitoire*) permet l'exécution ou non de la tâche T_6 associée à l'étape 12 suivant que l'application se trouve en régime cyclique ou en régime transitoire de départ. Ce booléen est par exemple associé à une mémoire initialement mise à un, puis remise à zéro dés la première activité de l'étape (*10*) ou (*11*).

Ce Grafcet comporte *17* transitions et *16* étapes. Nous représentons sur la *Figure 56* un Grafcet de commande établi de façon traditionnelle comportant *8* transitions et *10* étapes.

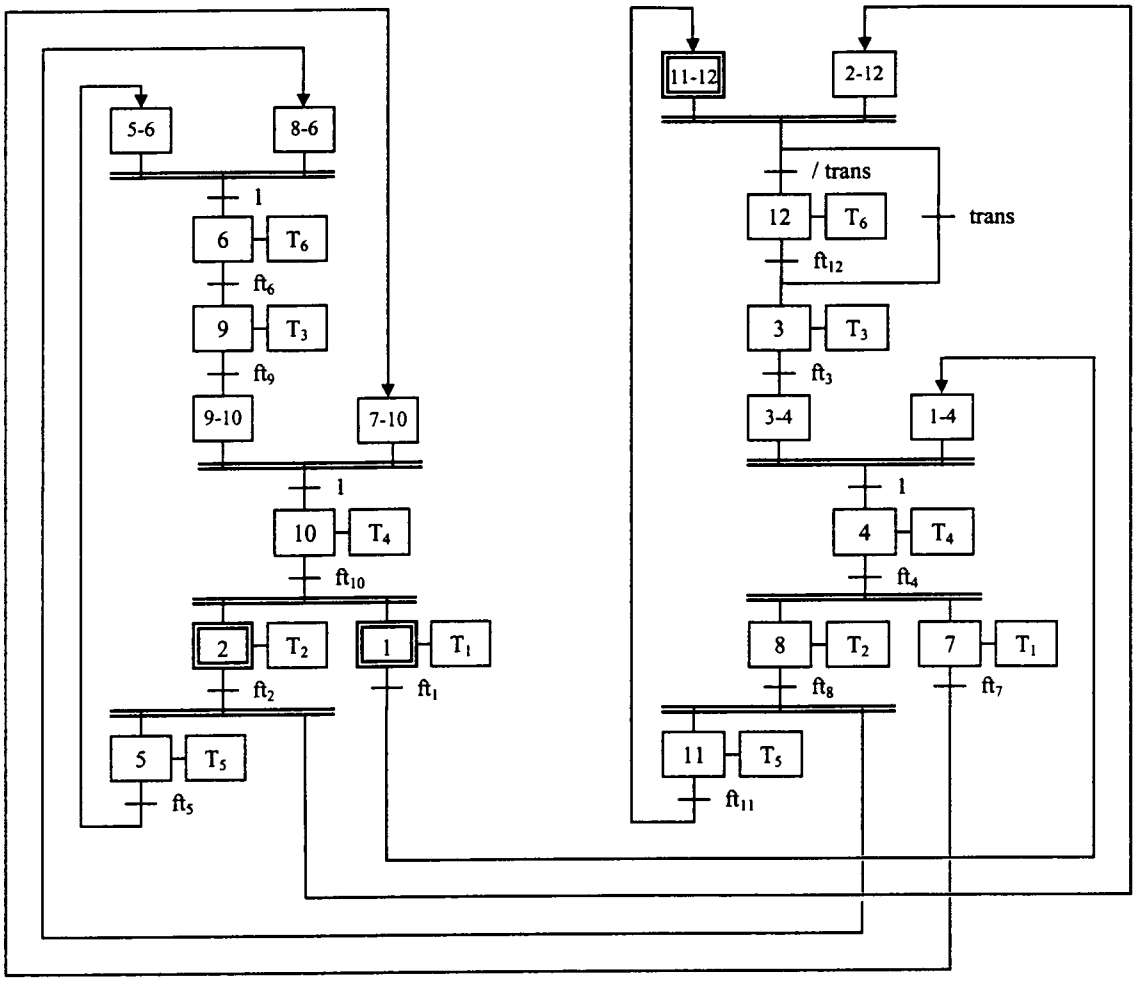


Figure 55 : Grafcet connexe de commande de l'application.

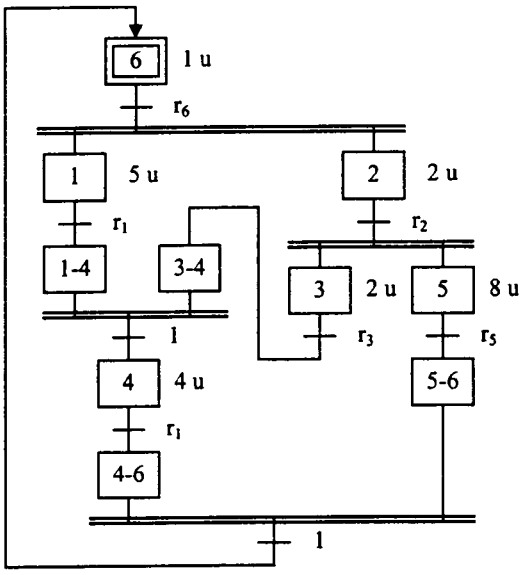


Figure 56 : Grafcet de commande traditionnel.

Hors aléa, le suivi d'exécution du Grafcet de commande de la *Figure 57* est conforme au diagramme de GANTT exploitable de la *Figure 47.*

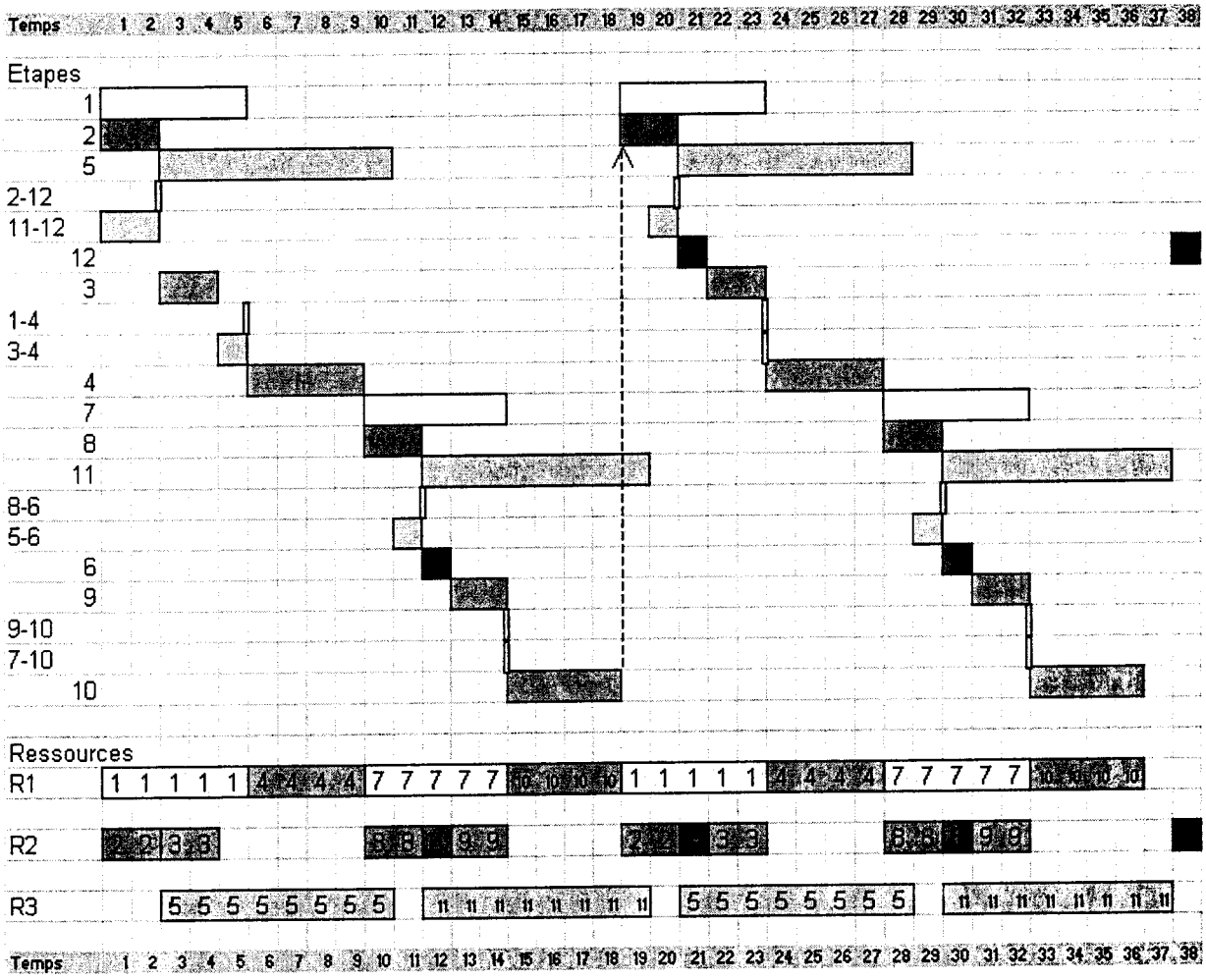


Figure 57 : Suivi d'exploitation du Grafcet de commande connexe.

➢ *Augmentation de la productivité.*

La production cyclique (2 produits finis) est effectivement obtenue au bout de 18 unités de temps alors que la solution traditionnelle (Cf. *Figure 56*) fournit un produit fini toutes les 11 unités de temps. Notre ordonnancement permet sur cet exemple d'augmenter la productivité d'environ 22%.

$$\frac{(2/18)-(1/11)}{(1/11)} = 22\%$$

Si le nombre d'étapes ou de transitions mesurait la complexité du Grafcet de commande, le gain de productivité ainsi obtenu se ferait aux dépends d'une complexité doublée.

4.5.2. Extraction d'un graphe de commande réparti.

Le point de départ de cette deuxième méthode est également le diagramme de la macrogamme. Le graphe de commande peut être représenté par la superposition des *nbr* graphes de gestion des ressources sur les n_e graphes de gestion des gammes GMT initiales de la macrogamme. Le graphe résultant du croisement des deux familles de graphes (Cf. *Figure 58*) devient rapidement très dense et le Grafcet de commande issu de sa transformation par la passerelle GMT-Grafcet devient lui aussi rapidement complexe. C'est pourquoi nous proposons une seconde méthode d'extraction fournissant un ensemble de n_e Grafcets propres à chaque gamme GMT initiale dont les exécutions parallèles sont conditionnées par l'activité des étapes de *nbr* Grafcets gérant la disponibilité des ressources.

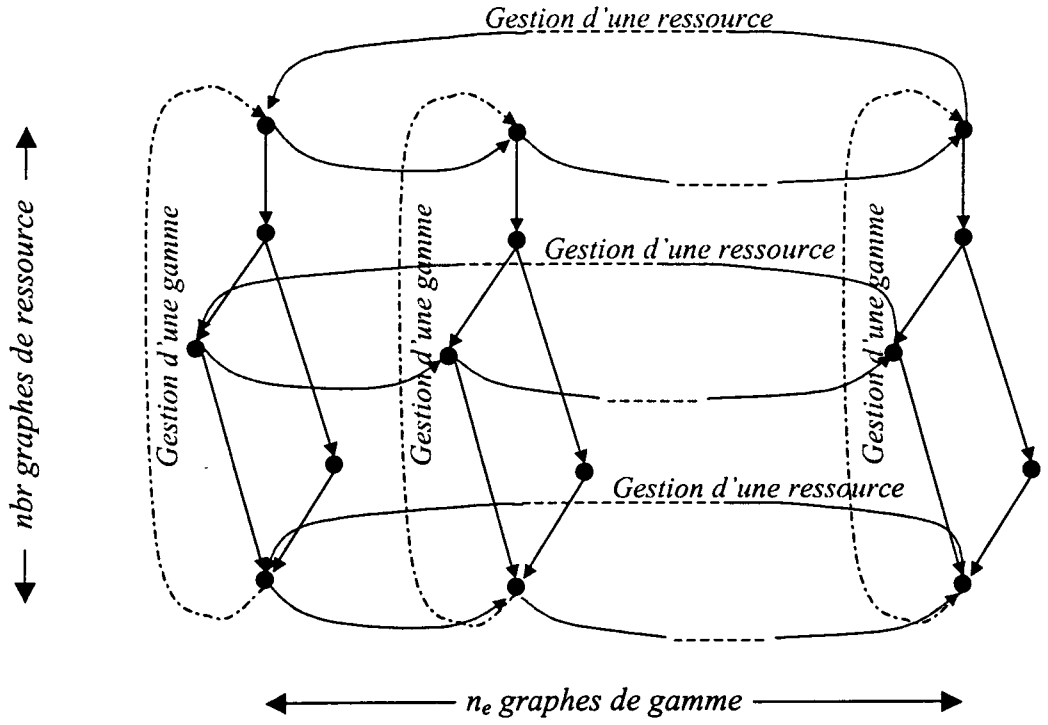


Figure 58 : Croisement des graphes de gamme et des graphes de ressource.

➢ *Principe.*

Chaque ressource est partagée entre les différentes tâches qu'elle doit exécuter en suivant la séquence de passage de ces tâches donnée par l'ordonnancement. Chaque graphe associé à une ressource est linéaire (au sens graphique de la définition) et il comporte autant de sommets que la ressource exécute de tâches sur une macrogamme. Sa structure équivalente dans le formalisme GRAFCET est immédiate et linéaire.

La structure du Grafcet d'une gamme GMT initiale reproduit les contraintes d'antériorité existant au sein de cette gamme initiale pour laquelle nous fusionnons les nœuds de '*début*' et de '*fin*'. Elle peut alors être obtenue à l'aide de la passerelle GMT→Grafcet.

Nous rétablissons ensuite les contraintes existant entre les deux familles de graphes par un mécanisme de type *hand shake.*

- L'exécution d'une tâche d'un Grafcet associé à une gamme est conditionnée par l'état de l'étape correspondante du Grafcet associé à la gestion de la ressource exécutant cette tâche.

- L'évolution d'un Grafcet associé à une ressource est à son tour conditionnée par la fin d'activité de l'étape du Grafcet associée à la gamme qui vient d'utiliser cette ressource.

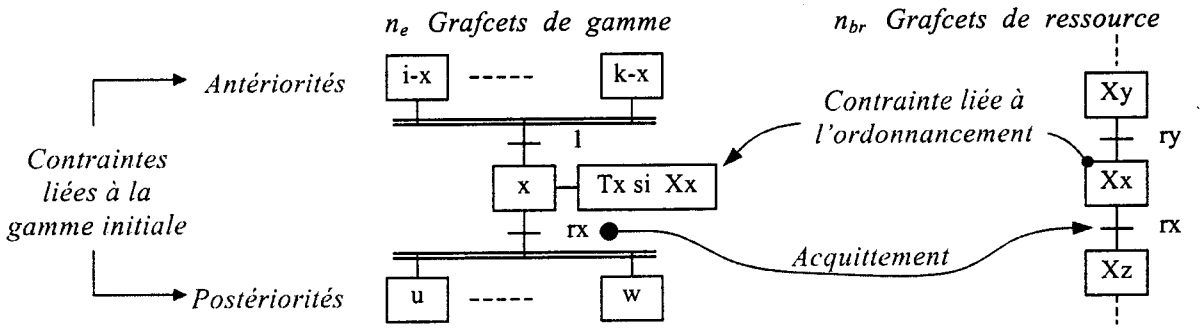


Figure 59 : Contraintes de gamme et d'ordonnancement.

Les étapes initiales des Grafcets de gamme correspondent à la (aux) première(s) tâche(s) exécutée(s) dans la gamme(s) GMT initiale(s) et les étapes initiales des Grafcets de gestion des ressources sont définies par la macrogamme. Si la vivacité peut être vérifiée à l'aide des outils développés au chapitre 3.3.2, il n'en est pas ainsi pour la célérité puisque nous conditionnons l'exécution des tâches associées aux étapes.

Enfin, la prise en compte du régime transitoire de début, c'est-à-dire la non-exécution pendant le régime transitoire des tâches appartenant à la (ou aux) dernière(s) gamme(s) GMT initiale(s) de la macrogamme peut être réalisée à l'aide d'un booléen témoin de l'existence de ce régime transitoire de début. Un mécanisme identique permet de gérer le régime transitoire de fin.

➢ *Application à l'exemple spécifié Figure 36.*

Le Grafcet de commande de notre application comporte cinq Grafcets connexes : deux Grafcets associés aux deux gammes GMT de la macrogamme et trois Grafcets associés aux trois ressources. Ils représentent au total *34* étapes et *31* transitions. Les étapes *(00)* et *(13)* correspondent à la fusion des nœuds '*début*' et '*fin*' des deux gammes initiales de la macrogamme.

La réceptivité *m* (en aval de l'étape *(00)*) symbolise l'ordre de marche alors que la réceptivité *X103.m* (en aval de l'étape*(13)*) participe au mécanisme de prise en compte du régime transitoire de départ : initialement, les étapes *(13)* et *(201)* sont actives. Lorsque la réceptivité *r2* est vraie, le franchissement de la transition située en aval de l'étape *(201)* provoque

l'activité fugitive de l'étape (*202*) (réceptivité *X13* vraie). L'activité de l'étape (*203*) autorise l'exécution de la tâche T_3. Lorsque la tâche T_4 (relativement aux étapes (*4*) et (*102*)) est terminée, l'activation de l'étape (*103*) provoque la désactivation de l'étape(*13*) et l'activation des étapes (*7*) et (*8*). En régime permanent, l'activité de l'étape (*13*) est fugitive.

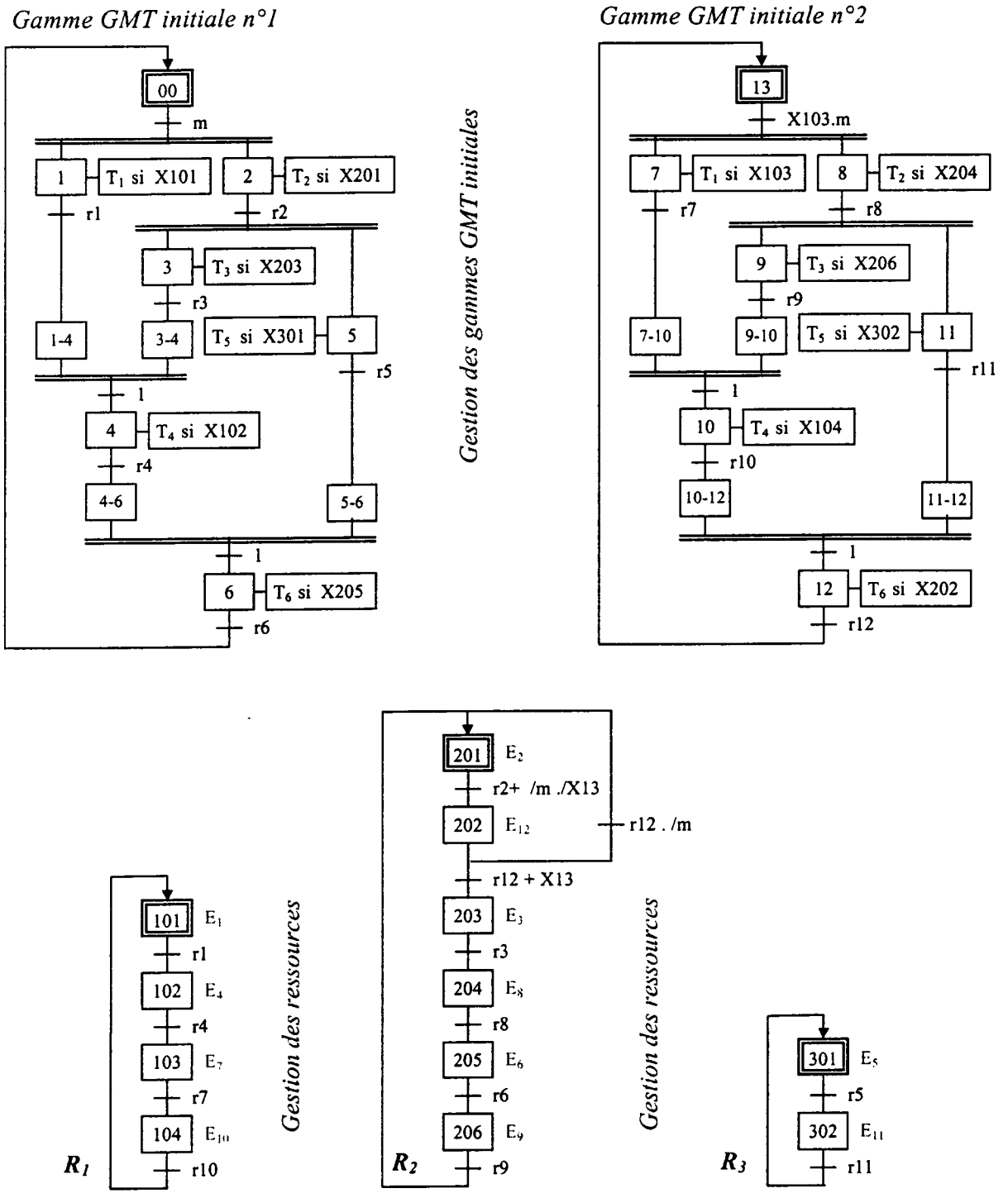


Figure 60 : Grafcets de commande.

Le régime transitoire de fin est pris en compte lorsque la variable *m* n'est plus vraie. Si la variable *m* devient fausse avant la fin de la tâche T_6 associée aux étapes (*6*) et (*205*), alors le Grafcet de la gamme n°1 se trouve bloqué dans son état initial (étape (*00*)) et l'étape (*206*) s'active. La tâche T_7 (étapes (*7*) et (*103*)) continue son exécution et la tâche T_9 (étapes (*9*) et

(*206*) peut s'exécuter. La fin d'exécution de ces deux tâches entraîne l'exécution de la tâche T_4 associée à l'étape (*10*) et parallèlement, la réceptivité *r9* et l'absence simultanée de *m* et de *X13* provoquent l'activité fugitive de l'étape (*201*) puis l'activité de l'étape (*202*). A la fin de l'exécution de la dernière tâche T_6 (relative aux étapes (*12*) et (*202*)), le Grafcet de gestion de la ressource R_2 revient dans son état initial.

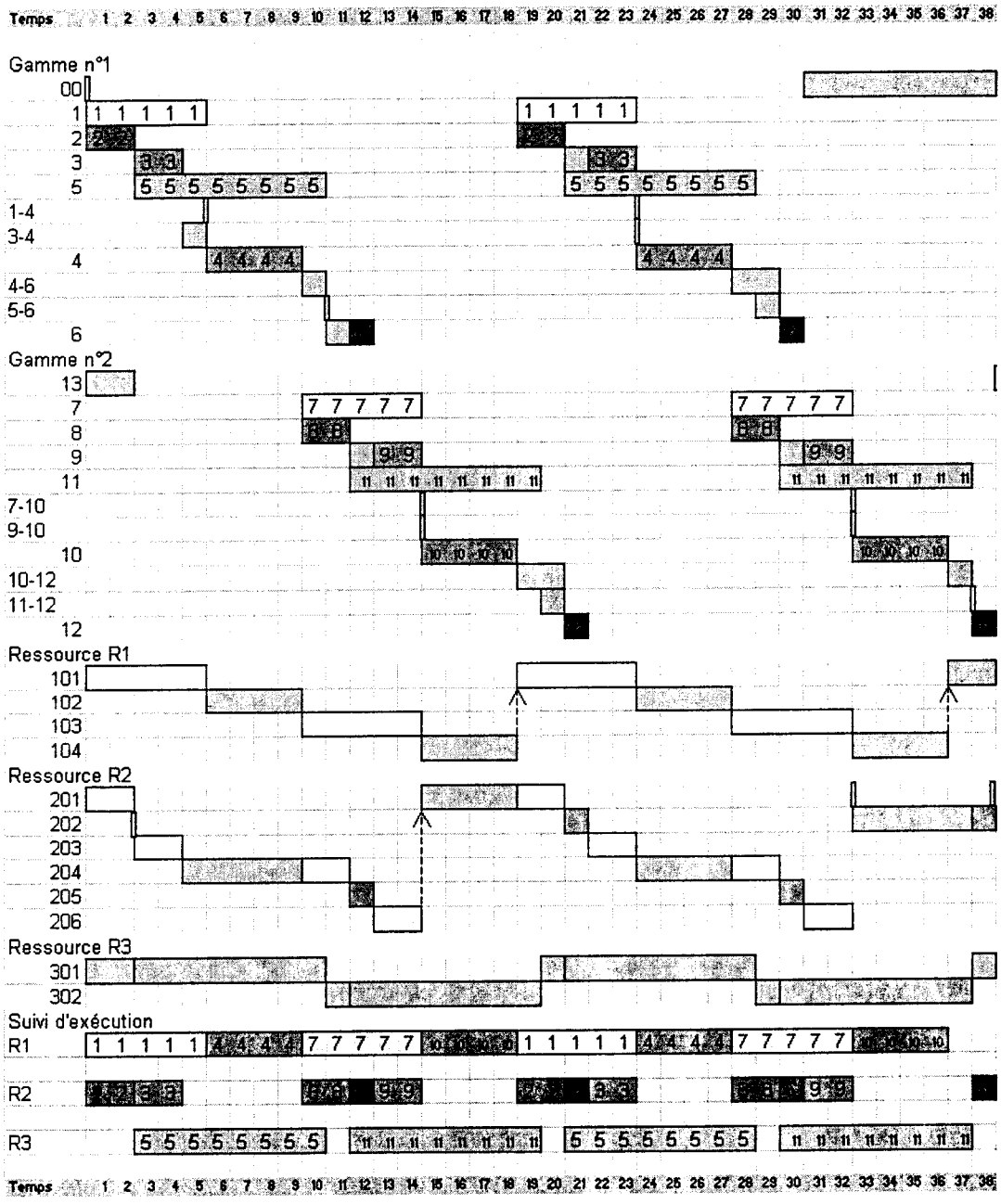


Figure 61 : Suivi d'exécution des Grafcets commande.

Le suivi d'exécution des cinq Grafcets de la *Figure 61* est conforme au diagramme de GANTT exploitable de la *Figure 47* et le gain de productivité est identique à la solution connexe exposée précédemment.

4.5.3. Conclusion sur les deux méthodes d'extraction.

Les deux méthodes d'extraction que nous venons d'étudier s'appuient sur la macrogamme et respectent le même diagramme de GANTT exploitable. Les Grafcets de commande obtenus permettent donc tous les deux de saturer la machine goulet avec un même encours, ce qui d'un point de vue de l'utilisateur rend ces méthodes équivalentes et de productivité optimale.
Cependant, la complexité du graphe unique croît avec le nombre d'encours minimal *Xe*. Dès lors et bien qu'elle entraîne a priori un nombre supérieur d'étapes/transitions, la solution répartie apparaît favorable et mieux structurée, en particulier du point de vue de la mise en œuvre et de la maintenance.
En contrepartie, les régimes transitoires de début et de fin de production semblent plus faciles à prendre en compte avec un graphe unique, parce que l'évolution d'un graphe connexe est plus commode à suivre que l'évolution de graphes multiples et interdépendants.
Notons finalement que ces deux méthodes d'extraction ne sont pas exclusives : en effet, il est toujours possible dans la solution connexe de respecter des contraintes d'antériorités non pas par l'implémentation dans le graphe d'un arc représentatif de cette contrainte, mais par l'ajout dans le Grafcet d'une étape d'attente désactivée par la condition de synchronisation.

4.6. Conclusion sur l'ordonnancement cyclique.

Nous venons de présenter une méthode d'ordonnancement cyclique basée sur la simulation et adaptée au traitement de gammes opératoires non linéaires. Le processus est résumé sur la *Figure 62* et il conduit finalement à l'obtention du Grafcet de commande du SAP qui assure une production optimale tout en respectant la gamme opératoire initiale.
L'arbitrage des conflits inhérents aux opérations d'ordonnancement est géré par un algorithme fondé sur le comportement des colonies de fourmis. Le résultat est donné par un diagramme de GANTT qui propose un ordonnancement saturant la machine goulet et qui fournit simultanément le régime cyclique et les régimes transitoires de début et de fin de production. Le traitement de gammes non linéaires (au sens de leur représentation graphique) nous a conduit à étendre la définition actuelle de l'encours à partir de laquelle nous introduisons la notion de macrogamme : cette dernière représente l'ensemble minimal de tâches qui autorise un fonctionnement autonome et cyclique de l'application. Enfin, deux techniques d'extraction exploitant la macrogamme permettent d'établir un modèle GRAFCET de commande du SAP qui sature sa ressource goulet.

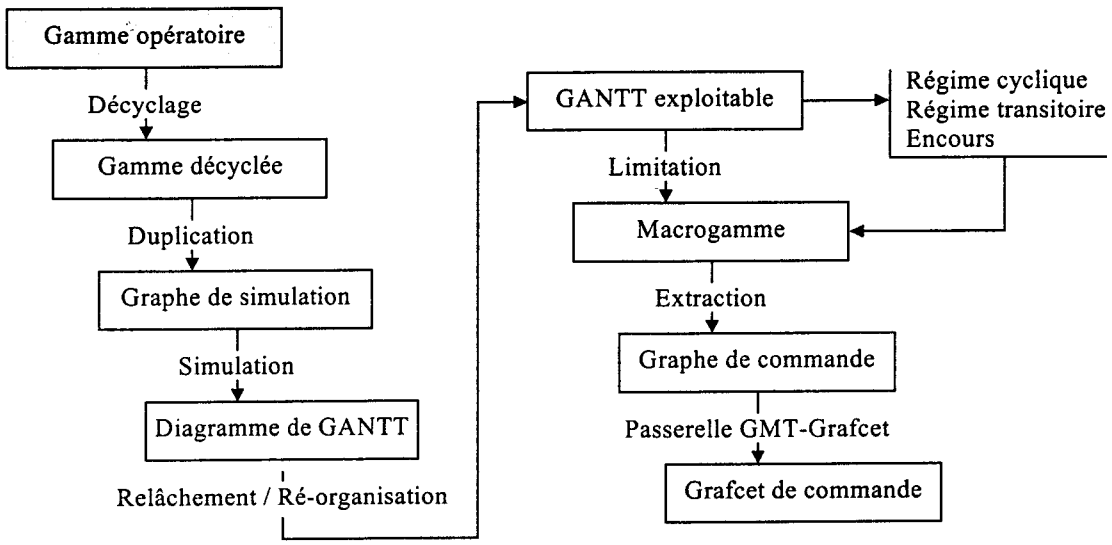


Figure 62 : Processus d'obtention du Grafcet de commande.

En conclusion et pour une gamme opératoire donnée, il est possible d'établir le Grafcet de commande d'un SAP tel que sa productivité soit optimale[25], puisque l'une de ses ressources peut toujours être saturée : le SAP ne peut pas '*mieux faire*' sans l'apport de modifications technologiques.

L'optimisation de la flexibilité est indirecte : elle est liée au fait que toute modification de gamme à laquelle un système flexible doit répondre, est considérée comme une nouvelle gamme et qu'il est possible à l'aide de notre méthode de re-concevoir le graphe de commande pour qu'il sature à nouveau une des ressources du SAP chargé de son exécution.

L'objectif fixé à la fin du chapitre *3* - § *3.5* est donc atteint : nous disposons à présent d'une méthode de conception des graphes de commande d'un SAP basée sur l'ordonnancement cyclique et nous pouvons vérifier les premières propriétés de célérité et de vivacité à l'aide des outils issus de la théorie des graphes.

Nous proposons dans le chapitre suivant, d'étendre l'application de notre méthode à deux exemples complémentaires.

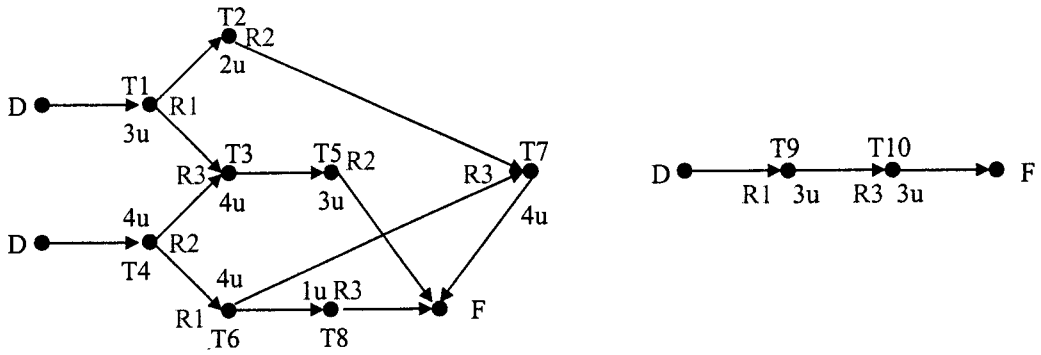
[25] L'optimalité au sens de l'encours n'est cependant pas garantie. L'encours n'est minimal que pour la macrogamme et il n'est optimal que s'il l'est pour l'ordonnancement global. Or nous avons vu que l'encours n'est qu'une conséquence de notre méthode d'ordonnancement et non pas un critère d'optimisation.

5. Exemples d'application complémentaires.

L'objectif est de montrer les capacités de la méthode à concevoir et à optimiser la conception d'un Grafcet de commande, d'une part pour une gamme opératoire plus complexe (exemple 1) et d'autre part pour une gamme opératoire linéaire (exemple 2).

5.1. Exemple 1.

L'exemple présenté sur la *Figure 63* est composé de deux gammes indépendantes, il comporte une dizaine de tâches et possède trois ressources. Le travail de chaque tâche, la durée et la ressource utilisée sont décrits dans le tableau de la *Figure 63*. A partir de trois produits bruts (*AB*, *CDE* et *F*), les deux gammes permettent la fabrication de quatre produits finis (*(BC)'*, *A'D*, *E'* et *F''*).



Descriptif des tâches.						Durée.	Ressource
Type de tâche	Entrées		N°T	Sorties			
Désassemblage	AB		T1	A	B	3	R1
Transformation	A		T2	A'		2	R2
Assemblage	B	C	T3	BC		4	R3
Désassemblage	CDE		T4	C	DE	4	R2
Transformation	BC		T5	(BC)'		3	R2
Désassemblage	DE		T6	D	E	4	R1
Assemblage	A'	D	T7	A'D		4	R3
Transformation	E		T8	E'		1	R3
Transformation	F		T9	F'		3	R1
Transformation	F'		T10	F''		3	R3

Figure 63 : Gamme opératoire de l'exemple 1.

Bien que la complexité de cet exemple reste abordable, l'établissement direct du Grafcet de commande n'est pas une opération aisée. Il nécessite la superposition des trois mécanismes de partage de ressources et des deux ensembles de contraintes d'antériorité liées aux gammes. Cette solution traditionnelle présente un temps de cycle de *16 unités de temps*, correspondant au circuit T_4, T_3, T_8, T_7, T_{10}.

Le processus exposé au paragraphe *4.2.2* nous permet d'établir la gamme décyclée à partir de la simulation d'une gamme GMT initiale unique. La ressource R_3 apparaît comme la ressource critique. Nous représentons la gamme décyclée sur la *Figure 64.*

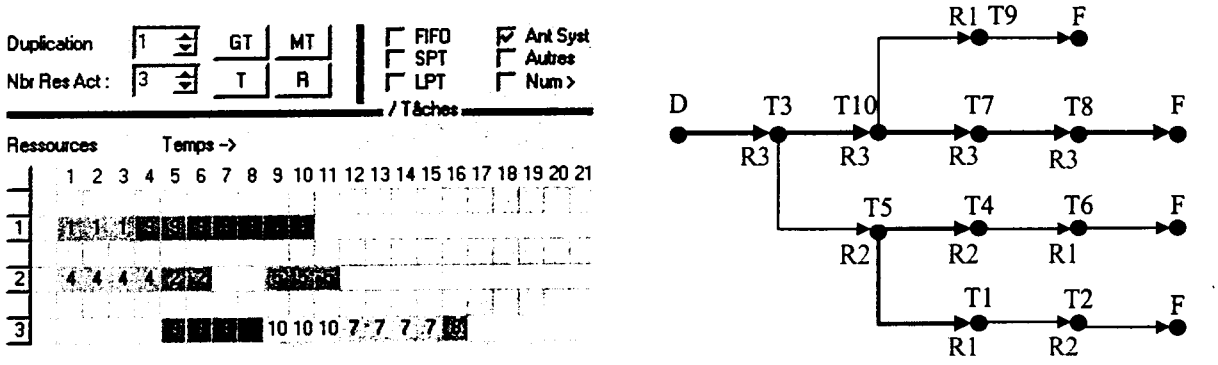


Figure 64 : Gamme décyclée.

Après duplication de la gamme décyclée, nous obtenons par simulation le diagramme de GANTT brut de la *Figure 65 (a)* : conformément à la procédure exposée au paragraphe *4.4.1,* nous en déduisons le diagramme exploitable de la *Figure 65 (b)* .

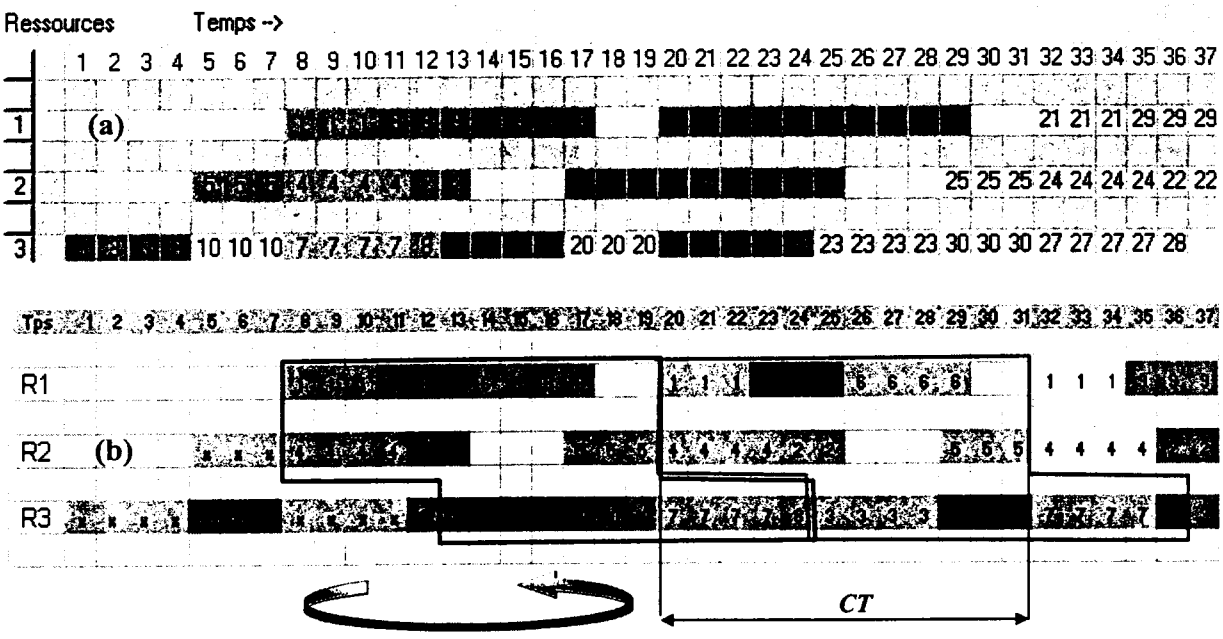


Figure 65 : Résultat de la simulation et diagramme de GANTT exploitable.

Quel que soit t appartenant à l'intervalle *CT*, *Xe(t)* est toujours égal à *un* : l'encours (global) *Xe* est donc égal à *un.* Il s'ensuit que la macrogamme ne comporte qu'une seule gamme GMT initiale et elle est représentée dans l'un ou l'autre des polygones de la *Figure 65 (b).* La simplicité de la macrogamme nous amène à définir le graphe de commande par un unique Grafcet connexe pour lequel nous établissons la matrice d'antériorité de la *Figure 66.*

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
T1					S	1				
T2	A			2						
T3	A			A				R3		
T4					2					
T5		2	SA							
T6				A					1	
T7		A				A				R3
T8						A	R3			
T9	1									
T10			R3						A	

Nous reportons la fermeture cyclique sur le goulet (marquée par *R3*), les fermetures cycliques sur les ressources 1 et 2 (marquées respect. par *1* et *2*), puis TOUTES les antériorités de la gamme initiale (marquées par la lettre *A*).

Les synchronisations des ressources sont notées par la lettre *S*. T3→T5 est une synchronisation naturelle issue de la gamme initiale alors que la synchronisation T5→T1 est issue d'un arc empruntant le lien de fermeture F→D

Figure 66 : Matrice d'antériorité de l'exemple 1.

La passerelle GMT-Grafcet fournit finalement le Grafcet de commande de cette application.

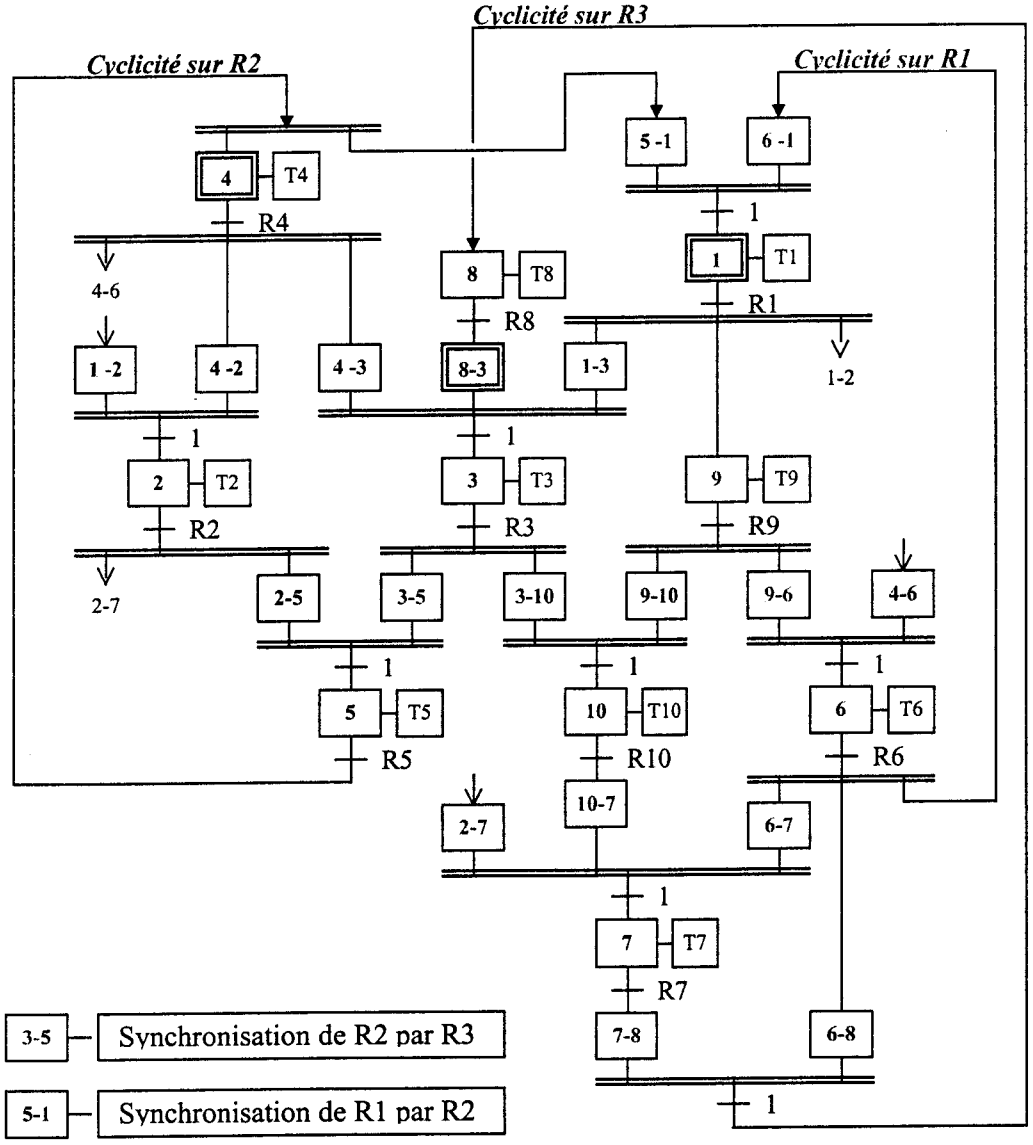


Figure 67 : Grafcet de commande connexe de l'exemple 1.

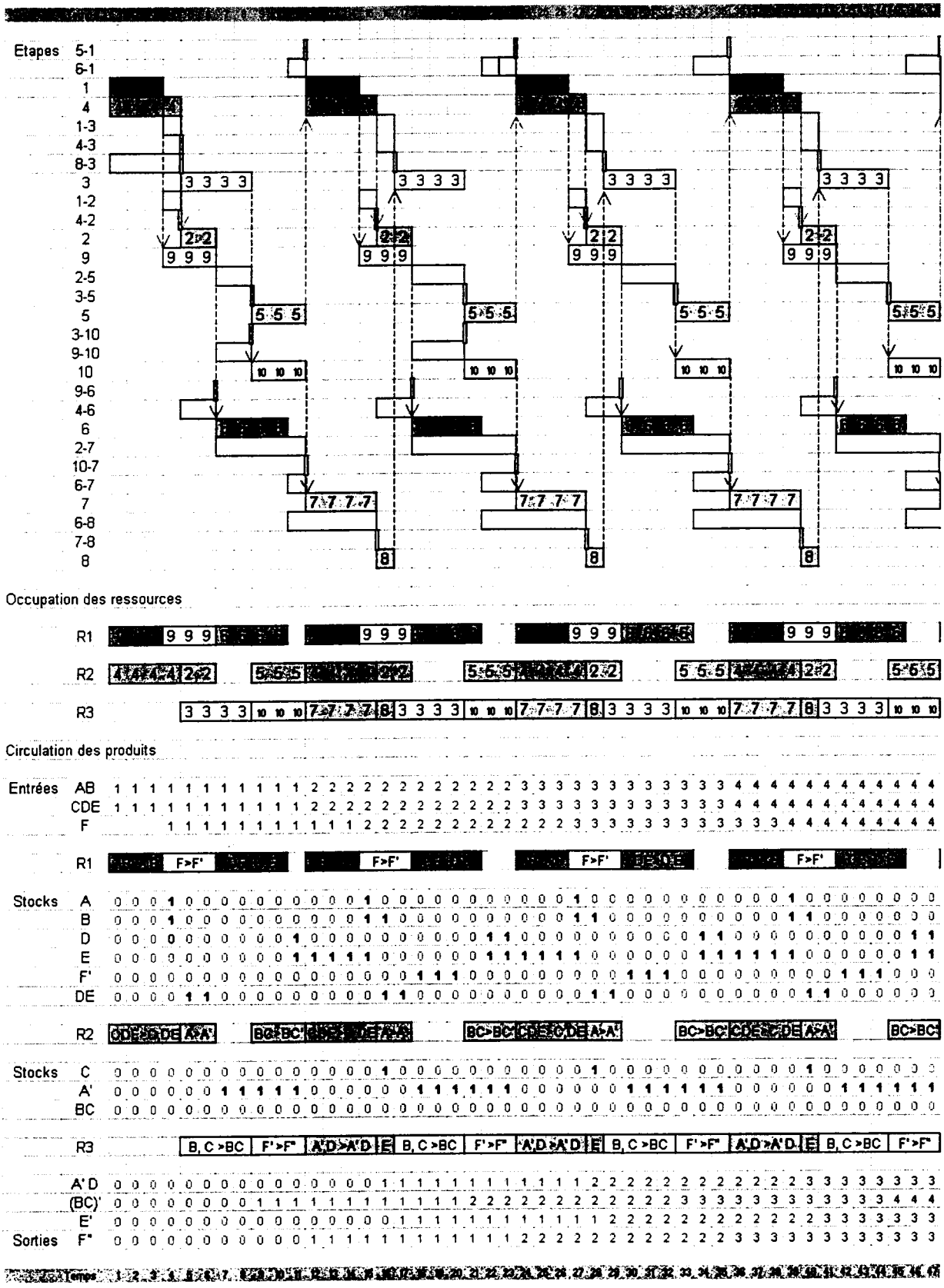


Figure 68 : Suivi d'exécution du Grafcet de commande relatif à l'exemple 1.

Le suivi d'exécution de la *Figure 68* montre la conformité du Grafcet de commande à l'ordonnancement exploitable précédemment établi. Nous pouvons vérifier, au sens de notre définition, que l'encours est bien d'une unité. Le gain de productivité est de 25% (*16u→12u*).

5.2. Exemple 2.

Le second exemple est donné sur la *Figure 69*. Il comporte quatre tâches et utilise trois ressources : il existe un seul produit fini qui résulte du traitement successif par les quatre tâches T_1, T_2, T_3 et T_4 d'un unique produit brut.

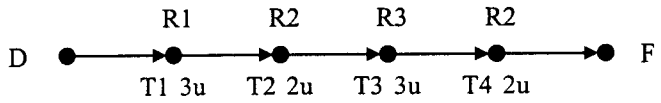


Figure 69 : Gamme opératoire de l'exemple 2.

Le Grafcet de commande traditionnel est immédiat : il est constitué d'une séquence linéaire de quatre étapes associées respectivement aux quatre tâches. Son temps de cycle est égal à la somme des durées des quatre tâches, soit *10 unités de temps*.

Nous appliquons la méthode de conception à ce deuxième exemple. Nous établissons la gamme décyclée à partir de la simulation d'une gamme GMT initiale unique. R_2 est la ressource critique. (*Figure 70.*)

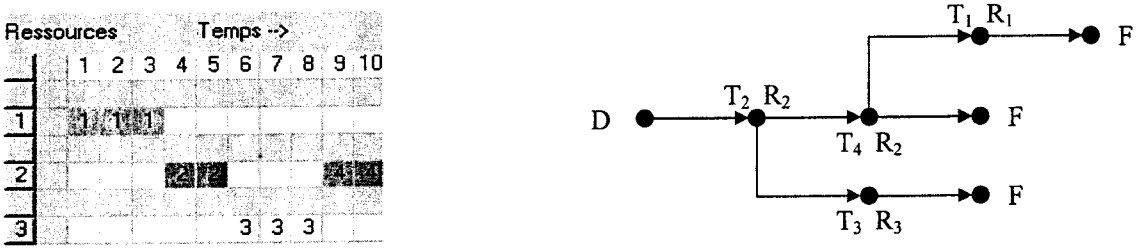


Figure 70 : Gamme décyclée de l'exemple 2.

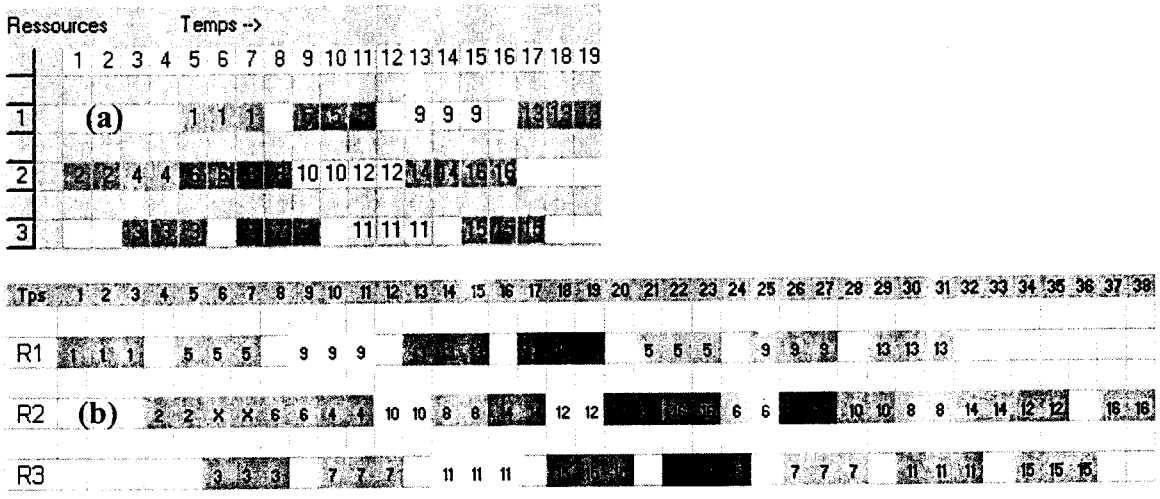


Figure 71 : Résultat de simulation (a) et Diagramme de GANTT exploitable (b).

La simulation fournit le résultat de la *Figure 71(a)* : conformément à la procédure exposée au paragraphe *4.4.1,* nous déduisons le diagramme de GANTT exploitable de la *Figure 71(b).* Le

début de la tâche T_2 (tâche n°*10* sur la *Figure 71(a)*) peut être relâché sur le diagramme de GANTT exploitable d'une unité, ce qui entraîne un décalage identique de la tâche T_3 (tâche n°*11* sur la *Figure 71(a)*). Quel que soit l'instant *t* du régime permanent, l'encours *Xe(t)* est inférieur ou égal à trois, ce qui signifie que la macrogamme comporte *3* gammes GMT initiales. Le fonctionnement de cet ensemble minimal de gammes GMT initiales est autonome et peut se reproduire cycliquement. A l'image de la *Figure 72*, la tâche T_{12} vient s'insérer entre les deux tâches T_2 et T_6, permettant ainsi la saturation cyclique de la ressource R_2.

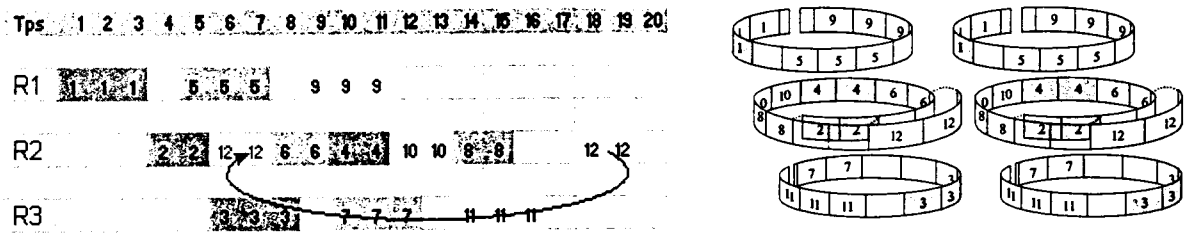


Figure 72 : Macrogamme relative à l'exemple 2.

Nous choisissons vis à vis du nombre d'encours (égal ici à *3*), d'établir une commande répartie composée de trois Grafcets associés aux trois gammes GMT initiales et de trois autres Grafcets associés aux trois ressources.

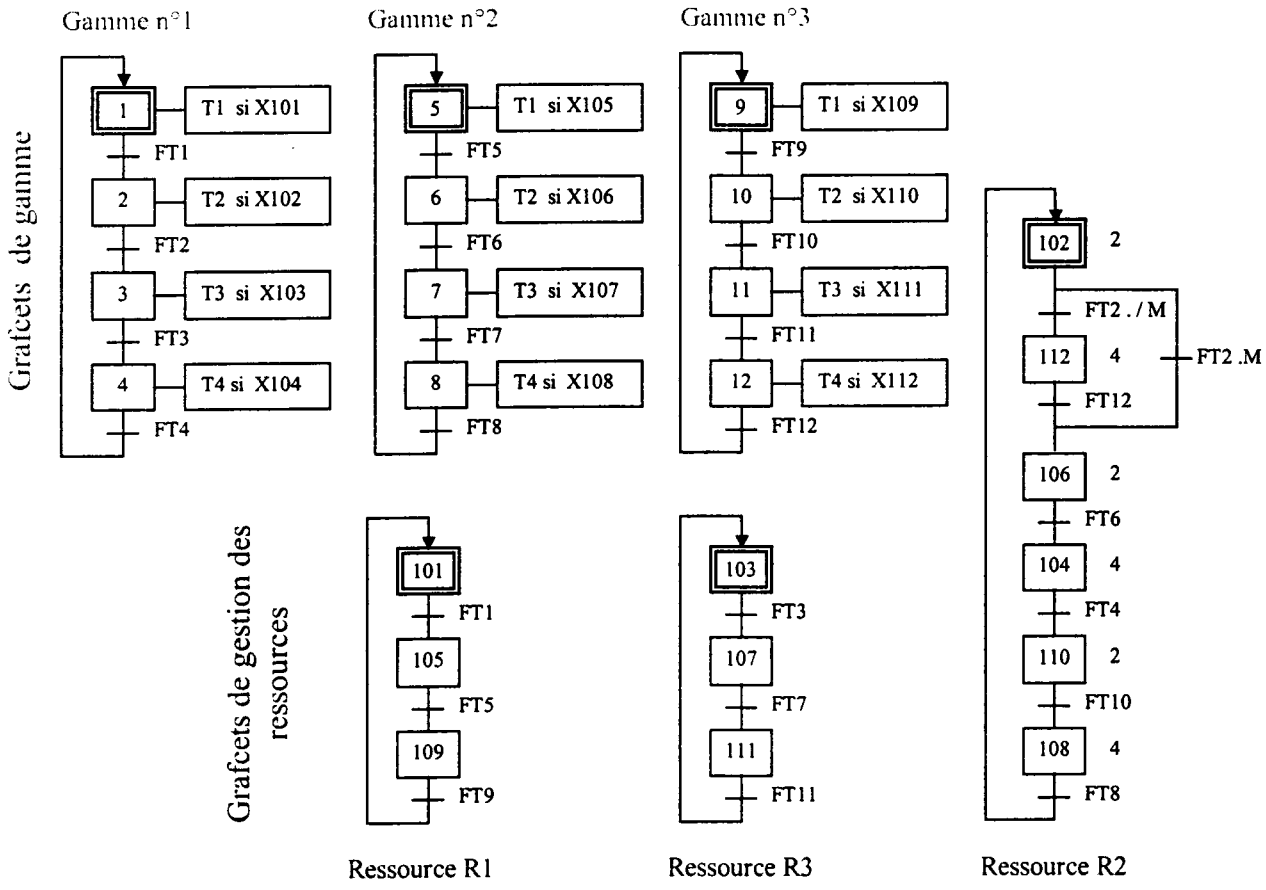


Figure 73 : Grafcets de commande répartie relative à l'exemple 2.

La variable notée *M* correspond à l'état d'une mémoire mise à *un* à l'initialisation et remise à *zéro* à la fin de la première exécution de la tâche T_5. Cet artifice permet la non-exécution de la première tâche T_{12} du régime transitoire.

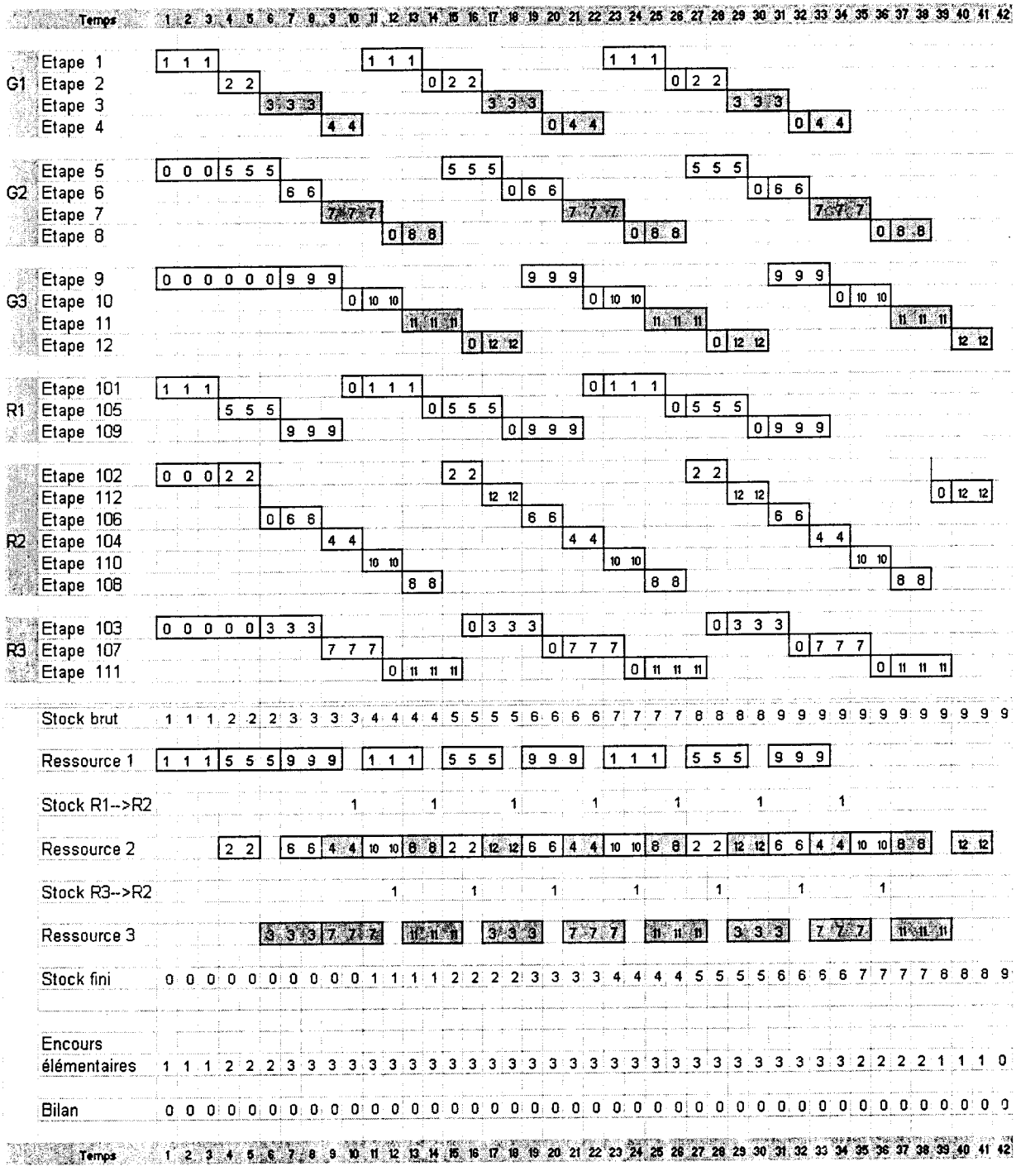


Figure 74 : Suivi d'exécution relatif à l'exemple 2.

Le transitoire du début de l'exécution est légèrement différent du transitoire prévu par l'ordonnancement du diagramme de GANTT exploitable. L'explication réside dans la non-exécution de la première tâche T_{12} et dans la propagation de cette non-exécution. Le transitoire de fin d'exécution diffère également des prévisions. Cette fois, l'écart est lié à la

non-exécution de la dernière tâche T_2. Le suivi d'exécution montre également l'évolution de l'encours élémentaire : ce dernier croît pendant le régime transitoire de début, se stabilise en régime cyclique, puis décroît pendant le régime transitoire de fin. Le régime cyclique est identique au résultat de simulation et la production cyclique de *3* produits finis est obtenue toutes les *12 unités de temps*. Le gain de productivité est appréciable :

$$\frac{(3/12)-(1/10)}{(1/10)} = 150\%$$

5.3. Constats.

La généralité d'une méthode ne peut pas se baser sur son application à des exemples. Dans ce paragraphe, nous avons simplement voulu montrer l'aide que peut apporter la méthode dans la phase de conception du graphe de commande d'un SAP, que les ingrédients d'un problème complexe soient ou non réunis.

Le premier exemple traite simultanément plusieurs produits bruts, assemble et désassemble des produits élémentaires pour constituer plusieurs produits finis. Il organise le partage des ressources entre différentes tâches, y compris lorsque celles-ci appartiennent à des gammes disjointes. A l'inverse, le second exemple est une simple suite de quatre tâches exécutées séquentiellement par trois ressources.

Dans les deux cas, le simulateur parvient à établir un enchaînement imbriqué des tâches qui sature une des ressources. L'exploitation du résultat de simulation fournit la macrogamme, à partir de laquelle nous construisons un Grafcet de commande de productivité optimale. Mais nous constatons à ce niveau que la prise en compte des transitoires par le Grafcet de commande n'est pas systématique et nécessite un traitement '*cas par cas*'.

Nous constatons également dans le deuxième exemple que l'introduction d'encours dévoile la possibilité de contourner certaines contraintes du cahier des charges a priori impératives. A première vue, le cahier des charges induit un Grafcet composé d'un seul circuit de *4* étapes.

Enfin, nous remarquons que plus le graphe de la gamme GMT initiale est linéaire, plus il induit un nombre d'encours important. Ce qui revient à dire que plus la gamme GMT de la production paraît simple, plus son graphe de commande est complexe ! Un graphe purement linéaire représente un enchaînement strictement séquentiel des opérations, ce qui est aux antipodes de ce qu'on cherche à faire dans une opération de parallélisation. Une gamme linéaire n'offre aucune opportunité d'imbrication naturelle qui favorise le parallélisme d'où l'*appétit* en encours de ce type de gamme.

6. Conclusion.

L'application étant décrite par son cahier des charges, notre démarche autorise :

- L'exploration approfondie de l'espace des solutions admissibles : en se basant sur des tirages aléatoires, le simulateur recherche et détermine un séquencement imbriqué et saturant des tâches, dont l'établissement *manuel* devient très difficile dès lors que le nombre de ces tâches s'accroît,
- L'interprétation étendue mais néanmoins respectueuse du cahier des charges : par exemple, la gestion du nombre minimal des encours de la macrogamme permet de mettre à jour des solutions cachées par la considération spontanée d'un encours unique,
- L'élaboration d'un Grafcet connexe ou réparti de commande,
- La vérification des propriétés de célérité et de vivacité des solutions retenues.

En résumé, la méthode de conception basée sur l'ordonnancement cyclique et les outils d'analyse des graphes concourent à l'étude d'une application dans l'esprit du cycle de vie SPIRALE. Nous sommes à présent en mesure de concevoir (ou de re-concevoir) le graphe de commande d'un système automatisé de production pour que sa productivité soit optimale. Les outils de la théorie des graphes développés autour du formalisme GMT nous permettent alors de juger des propriétés du modèle obtenu. En amont, ces outils et en particulier la technique DSM, permettent de procéder à une analyse macroscopique favorable à la conception d'une architecture distribuée.

Par ailleurs, la méthode d'ordonnancement cyclique, méthode que nous avons initialement développée dans le cadre de la conception de la commande des SAP et qui traite au sens graphique des gammes non-linéaires, peut également s'appliquer à l'ordonnancement d'un atelier.

Dans la troisième partie de ce mémoire, nous proposons de confirmer les aptitudes de notre approche à optimiser la productivité d'une application de type MPS. La réponse de l'application à diverses combinaisons du Minimal Part Set nous permet d'apprécier sa flexibilité.

7. Bibliographie.

[AYGALINC P. & DENAT J-P., 92] AYGALINC P. & DENAT J-P., 92: "Validation de modèles GRAFCET fonctionnels et évaluation des performances du système associé par l'utilisation des réseaux de PETRI.". GRAFCET'92, PARIS, 25-26 mars 1992.

[BONABEAU E. & THERAULAZ G., 94] BONABEAU E. & THERAULAZ G., 94: *"Intelligence collective."*. Editions HERMES.

[COHEN G., 95] COHEN G., 95. Théorie algébrique des systèmes à événements discrets. Centre Automatique et Systèmes - Ecole des Mines de Paris, INRIA Rocquencourt.

[COMMONER F., 71] COMMONER F., 71 :*"Marked Directed Graphs"*. Journal of Computer and System Science, Vol 5.

[DONALD V. & STEWARD, 65] DONALD V. & STEWARD, 65 :*"Partitioning and tearing systems of equations."*. J. SIAM Number. Anal Ser.B, Vol 2, n°2.

[DONALD V. & STEWARD, 81] DONALD V. & STEWARD, 81 :*"The design structure system: A method for managing the design of complex systems."*. IEEE Transactions on engineering, Vol EM-28, n°3.

[DORIGO M., MANIEZZO V. & COLORNI A., 91] DORIGO M., MANIEZZO V. & COLORNI A., 91:*"Positive feedback as a search strategy."*. *RR N°* 91-016, Politecnico di Milano, MILAN, June 1991.

[ESQUIROL P. & LOPEZ P., 99] ESQUIROL P. & LOPEZ P., 99: *"L'ordonnancement."*. ECONOMICA - PARIS.

[FROMENT B., 93] FROMENT B., 93 :*"Structural Design with Grafcet :Towards a global method"*. Automatic Control Production Systems Review AFCET HERMES, Vol 27, n°1.

[GONDRAN M. & MINOUX M., 95] GONDRAN M. & MINOUX M., 95: *"Graphes et algorithmes"*. EYROLLES 3ième édition.

[HARARY F., 62] HARARY F., 62 :*"A graph approach to matrix inversion by partitioning."*. Numerische Mathematik., Vol 4.

[KORBAA O., 98] KORBAA O., 98:"Commande cyclique des systèmes flexibles de production manufacturière à l'aide des réseaux de Pétri: De la planification à l'ordonnancement des régimes transitoires.". **Thèse** de Doctorat, 2295, Ecole Centrale de LILLE - URA CNRS D1440, LAIL - LILLE.

[KORBAA O. & GENTINA JC., 00] KORBAA O. & GENTINA JC., 00 :"Ordonnancement cyclique en production manufacturière". Revue Internationale d'Ingénierie des Systèmes de Production Mécanique. Juin 2000.

[KROMMENACKER N. & R. E., 99] KROMMENACKER N. & R. E., 99: *"Organisation des architectures de communications dans les environnements de travail coopératif.".* JDA'99 - Journées Doctorales d'Automatique., NANCY, 21/22/23 septembre 1999.

[LOPEZ P. & ROUBELLAT F., 01] LOPEZ P. & ROUBELLAT F., 01: *"Ordonnancement de la Production.".* HERMES Science, Publication.

[MARES M., 97] MARES M., 97:"Sur la modélisation et l'analyse des systèmes complexes à événements discrets par réseaux de PETRI". **Thè**se de Doctorat, 225, Ecole doctorale des Sciences de l'ingénieur de NANTES, LISA - ANGERS.

[MERLAUD CH1., 84] MERLAUD CH1., 84 :"Une méthodologie d'analyse descendante appliquée à la description du fonctionnement des systèmes automatisés. Partie 4.". TECHNOLOGIE et FORMATION, Vol 1.

[ROGERS JL. & SALAS AO., 99] ROGERS JL. & SALAS AO., 99 :"Toward a more flexible web-based framework for multidisciplinary design.". Vol 30.

[SMITH RP. & EPPINGER SD., 97] SMITH RP. & EPPINGER SD., 97 :*"A predictive model of sequential iteration engineering design.".* Managment science, Vol 43, n°8.

[TAILLARD P., 91] TAILLARD P., 91 :"Description fonctionnelle par Grafcet : Méthodologie d'analyse globale". TECHNOLOGY et FORMATION, Vol 41.

[TANG D., & al., 00] TANG D., & al., 00 :*"Re-engineering of the design process for concurrent engineering.".* Computer & industrial engineering, Vol 38, n°pp 479-491.

Troisième partie : Cas d'étude
Impact sur la flexibilité.

Table des matières.

1. Objectifs.

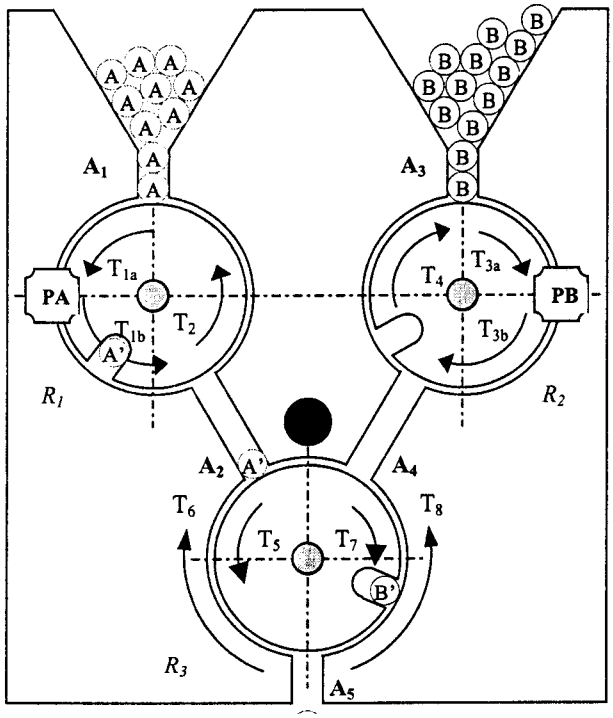
Nous revenons à la première question que nous nous sommes posée au début de ce mémoire : « *Un système automatisé de production peut-il être à la fois productif et flexible ?* »
A la suite de cette question et avant même d'y répondre, nous pouvons nous demander comment quantifier les performances de productivité et de flexibilité.

La productivité mesure le triple objectif QCD (Qualité, Coût, Délais) et l'unité commune d'évaluation de ces trois paramètres disparates ne peut être que l'euro. En première approche, le coût global résulte du coût de production, du coût de la non-qualité et du coût lié au dépassement des délais. En outre, ces coûts ne sont pas forcément indépendants : par exemple, la diminution des coûts de production, à travers la baisse en deçà d'une limite du temps de cycle d'une machine, pourrait entraîner une augmentation des coûts de non-qualité.
De façon analogue, la flexibilité peut s'exprimer à travers le coût des stocks et le coût de la non-réactivité, autrement dit, le coût lié à l'incapacité de l'entreprise à répondre aux demandes des clients dans les délais que ces clients lui imposent. Les coûts liés à l'utilisation flexible d'un SAP dépendent également des scénarii de production et du coût des régimes transitoires.
L'évaluation exacte des coûts de cette liste non-exhaustive s'avère donc très difficile, même en possession de tous les paramètres techniques et économiques relatifs à une étude de cas précis.
Sans aller jusqu'à une appréciation qualitative, nous suggérons en guise de compromis, de baser l'évaluation de la productivité d'un SAP sur sa célérité, c'est-à-dire sur son temps de cycle (démarche adoptée de manière implicite dans la deuxième partie de ce mémoire). Dans le même esprit, nous proposons d'évaluer la flexibilité d'un SAP à travers sa capacité à rester productif pour chaque type de production qui lui est demandé (diversité des tâches).
Dans les faits, cette évaluation discutable traduit davantage l'impact de la productivité sur la flexibilité.

En admettant le temps de cycle comme premier critère d'évaluation, nous proposons dans cette troisième partie de montrer sur un cas pratique, l'aptitude de notre méthode à concevoir une commande à la fois productive et flexible. Nous comparerons les résultats obtenus à l'aide de notre méthode aux résultats observés sur des graphes de commande issus de méthodes d'agencement conventionnelles. Enfin, nous montrons les limites de notre méthode et sa complémentarité avec les autres.

1.1. L'application.

Nous disposons d'un processus destiné à constituer un produit fini en dosant des quantités entières et variables x et y de produits bruts de type A et B. Le processus est schématisé sur la *Figure 1*. Les zones repérées par les points A_2 et A_4 autorisent le stockage d'une quantité entière d'éléments A' ou B'.



Le processus est composé de 3 ressources.

R_1 exécute les tâches T_{1a} ,T_{1b} et T_2 qui amènent respectivement un produit brut A du point A_1 à la station **PA**, de la station **PA** au point A_2, puis retourne au point A_1.

R_2 exécute les tâches T_{3a} , T_{3b} et T_4 qui amènent respectivement un produit brut B du point A_3 à la station **PB**, de la station **PB** au point A_4, puis retourne au point A_3.

R_3 exécute les tâches T_5 et T_6 qui amènent respectivement un élément A' du point A_2 à A_5, puis retourne au point A_2.

R_3 exécute les tâches T_7 et T_8 qui amènent respectivement un élément B' du point A_4 à A_5, puis retourne au point A_4.

La petite zone comprise entre A_2 et A_4 est interdite.

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
R_1	9	3						
R_2			8	3				
R_3					3	4	3	4

Figure 1 : Description de l'application.

Lorsque la ressource R_1 prend un produit brut A au point A_1, elle lui fait subir une opération sur le poste PA, puis amène l'élément traité A' au point A_2. Les deux tâches T_{1a} et T_{1b} sont confondues dans la tâche T_1 de durée d_1 (d_{1a} + d_{1b}). La ressource R_2 traite un produit brut B de manière identique, mais le temps total d_3 (d_{3a} + d_{3b}) peut être différent. Les deux tâches T_{3a} et T_{3b} sont confondues dans la tâche T_3.

1.2. MPS variable (Minimal Part Set).

Afin de donner à notre dispositif un caractère flexible, nous envisageons de traiter des MPS variables. Il s'agit de générer la commande du processus pour divers MPS obtenus en faisant varier les quantités entières x et y de produits bruts A et B dans l'intervalle {1.. 4}. Nous allons étudier les 16 combinaisons possibles de MPS.

2. Etude.

Nous étudions successivement différentes commandes possibles du dispositif : en premier lieu, nous établissons un modèle Grafcet connexe utilisant deux compteurs et partageant la ressource commune R_3. Nous étudions ensuite un modèle Grafcet dont les composantes connexes sont spécifiques à chacune des seize combinaisons du MPS, une composante supplémentaire étant chargée de mettre en œuvre la composante associée au MPS souhaité. Enfin, nous étudions la solution basée sur notre méthode d'ordonnancement cyclique.

Pour les trois types d'étude, nous supposons qu'initialement, les ressources R_1, R_2 et R_3 sont respectivement aux points A_1, A_3 et A_5 et que les aires de stockage A_2 et A_4 sont vides. Nous supposons également que les ressources consommables représentées par les produits bruts A et B sont toujours disponibles : le problème est central.

2.1. Commande connexe – utilisation de compteurs.

2.1.1. Etablissement du Grafcet de commande.

La capacité du stockage en A_2 et A_4 est unitaire, R_1 et R_2 ne peuvent déposer respectivement un élément A' ou B' que si la ressource R_3 a libéré l'aire de stockage A_2 ou respectivement A_4. De plus, la ressource R_3 ne peut libérer l'aire de stockage A_2, c'est-à-dire travailler pour R_1, que si elle a fini son travail pour la ressource R_2 (et réciproquement). La commande doit, en fonction de l'état des deux compteurs associés au MPS et de l'activité des différentes tâches, gérer trois sections critiques associées ici aux étapes (*6-1*), (*8-3*) et (*30*). Un mécanisme identique pour les trois ressources prélève et remet le jeton d'occupation de cette ressource dans l'étape associée.

Lorsque le compteur associé à un élément a atteint sa consigne, par exemple $C_A=x$, la fin de la tâche T_2 active l'étape d'attente (*2-2*), puis sur la fin de la tâche T_5, l'étape d'attente (*2-20*). A la fin des deux contrats (x éléments A' et y éléments B'), l'étape (*20*) est activée. Si la réceptivité *m* est vraie alors un nouveau cycle reprend, sinon, l'activité de l'étape (*10*) permet d'effectuer une nouvelle sélection. Le Grafcet de commande est présenté sur la *Figure 2*.
Ce n'est pas l'unique solution : on peut imaginer un graphe de commande principal gérant les différentes sections critiques (trois Grafcets) et pilotant chacune des trois ressources (trois Grafcets). L'interaction entre ces sept Grafcets serait alors assurée par les variables internes associées aux étapes, mais cette solution conduirait au même diagramme de suivi d'exécution.

Notons que l'utilisation de compteurs rend ce modèle de commande quasiment universel quant à la diversité des combinaisons de MPS possibles.

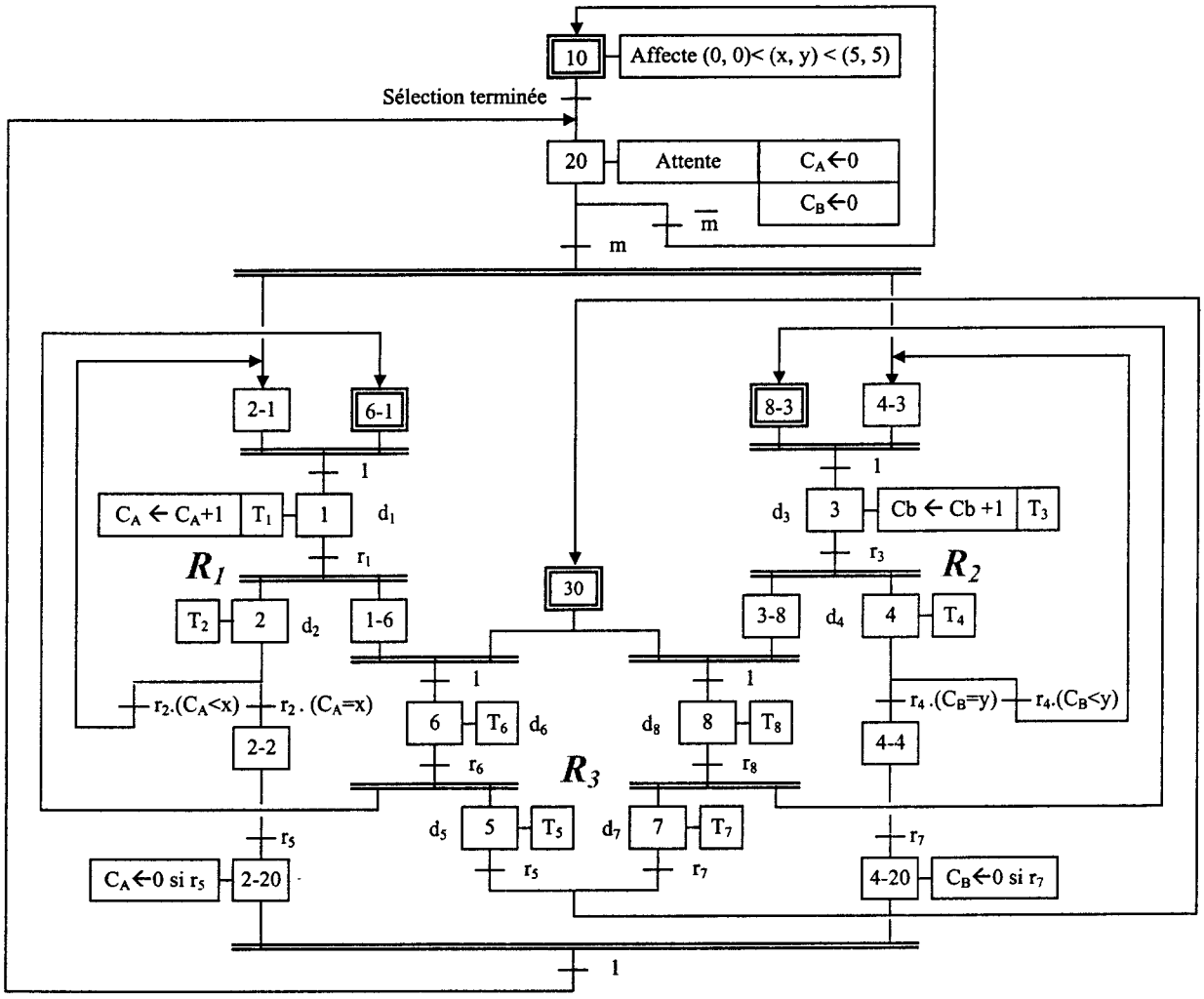


Figure 2 : Modèle Grafcet de commande connexe.

2.1.2. Suivis d'exécution.

Nous établissons le diagramme de suivi du modèle Grafcet représenté sur la *Figure 2* pour chaque combinaison des variables *x*, *y* définissant les différents MPS. Nous observons sur ces diagrammes (Cf. *Figure 3*, *Figure 4* et *Figure 5*) que chaque cycle comporte son propre transitoire, ce qui rend l'application peu performante pour les petites valeurs de *x* et *y*. Il en découle que les cas (x, y)=(2, 2), (3, 3) et (4, 4) ne sont pas des multiples du cas (x, y)=(1, 1). La même remarque peut être faite pour les cas (x, y)=(4, 2) et (x, y)=(2, 1). C'est le prix de *'l'universalité'*.

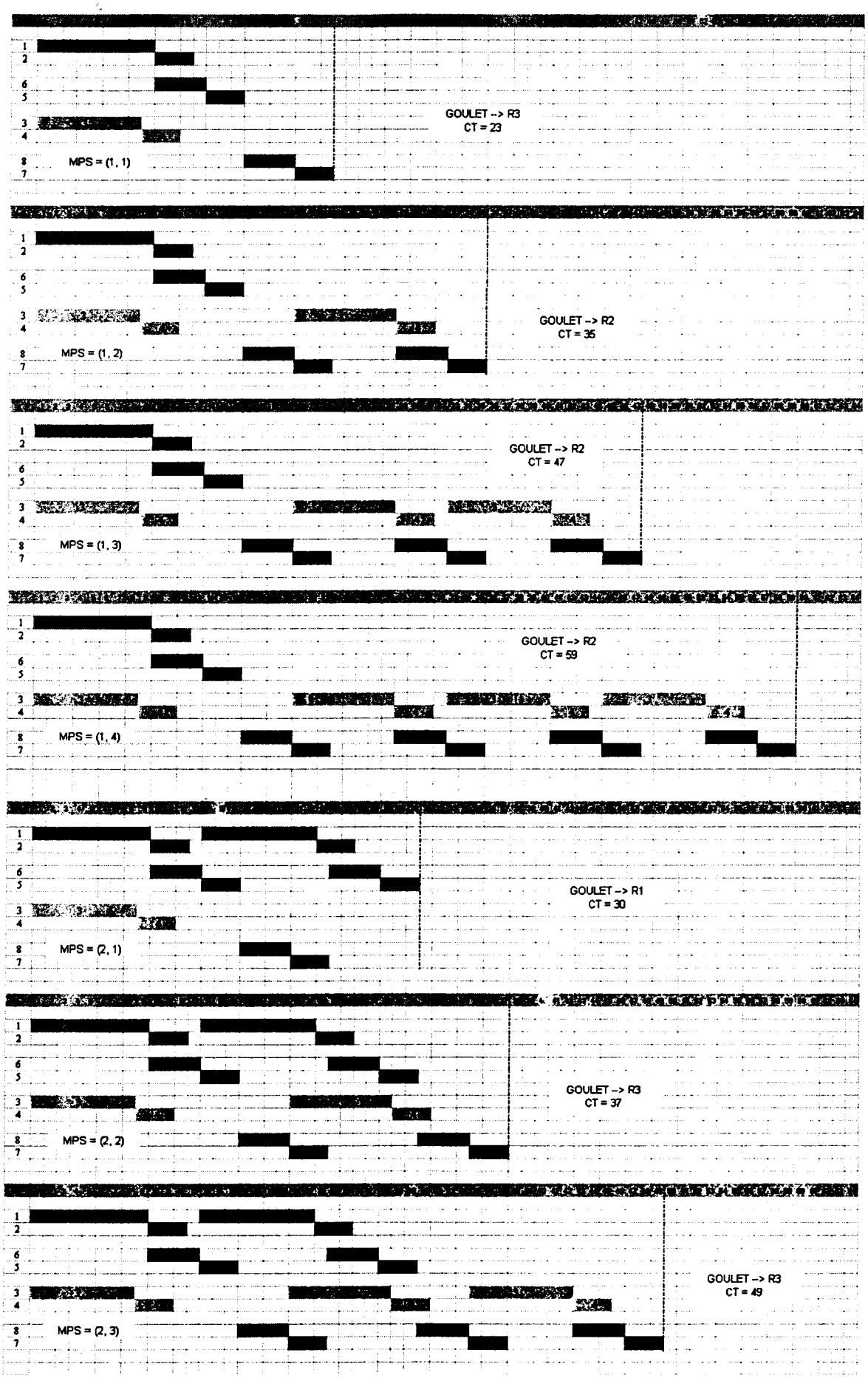


Figure 3 :Diagrammes de suivi des ressources pour (x, y) de (1, 1) à (2,3).

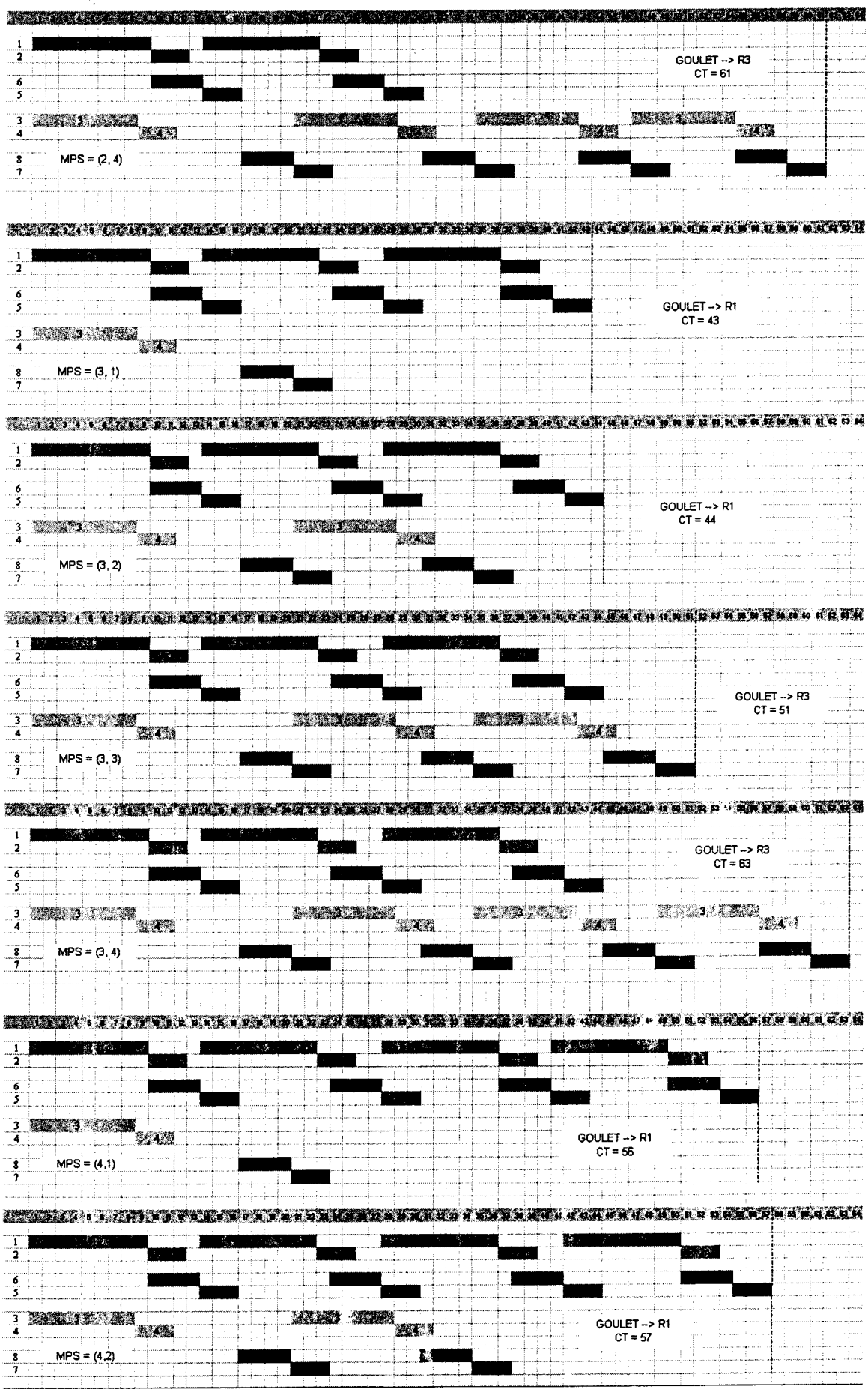


Figure 4 : Diagrammes de suivi des ressources pour (x, y) de (2, 4) à (4, 2).

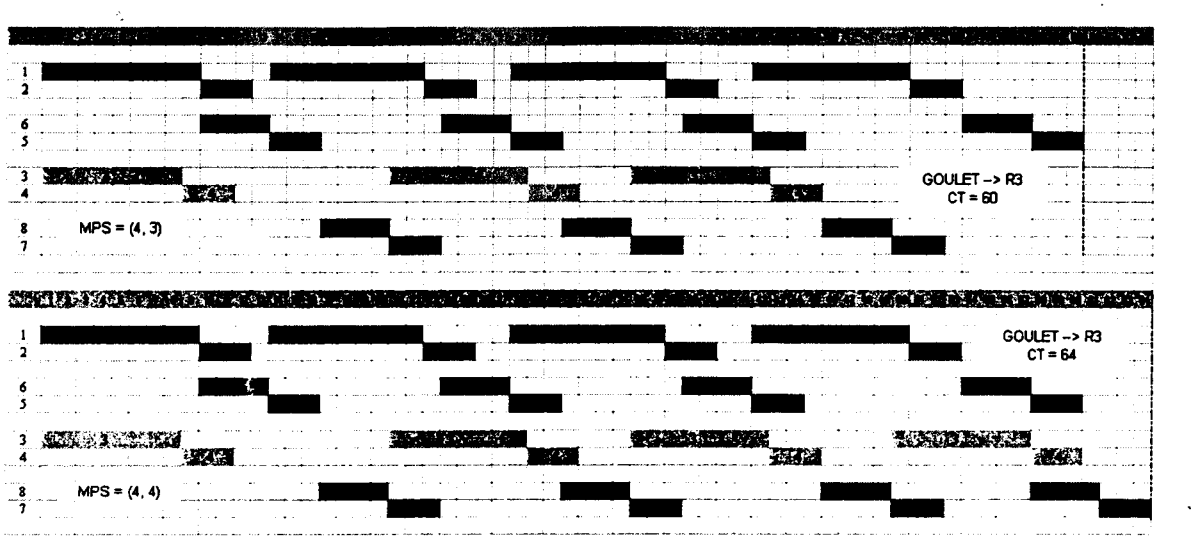


Figure 5: Diagrammes de suivi des ressources pour (x, y) = (4,3) et (4, 4).

2.1.3. Résultats.

Nous récapitulons les différents temps de cycle théoriques de l'application dans *Tableau 1*. Ces valeurs (en unités de temps) sont fonction des durées de tâches données sur la *Figure 1*.

Connexe	x = 1	x = 2	x = 3	x = 4
y = 1	23	30	43	56
y = 2	35	37	44	57
y = 3	47	49	51	60
y = 4	59	61	63	64

Tableau 1 : Temps de cycle du graphe connexe pour différentes MPS.

2.2. Commande multiple et dédiée.

2.2.1. Etablissement des Grafcets de commande.

Nous décrivons à présent une solution plus proche de notre philosophie. Nous déterminons pour chaque MPS un modèle Grafcet spécifique de commande de manière traditionnelle. Puis nous sélectionnons par l'intermédiaire d'un modèle Grafcet principal le modèle Grafcet associé au MPS souhaité. L'étude se réduit à onze cas. En effet, les cas numéro *6, 11* et *16* (Cf. *Tableau 2*) sont des multiples du cas *1*. De même, le cas numéro *8* est un multiple du cas numéro *2* et le cas numéro *14* est un multiple du cas numéro *5*.

x/y	1	2	3	4
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

⇨

x/y	1	2	3	4
1	1	2	3	4
2	5		7	
3	9	10		12
4	13		15	

Tableau 2 : Les différents MPS pour une commande multiple et dédiée.

La gestion de l'aire de stockage est assurée par le rendez-vous réalisé en aval des étapes (*1*) et (*6*). La ressource R_3 est partagée séquentiellement.

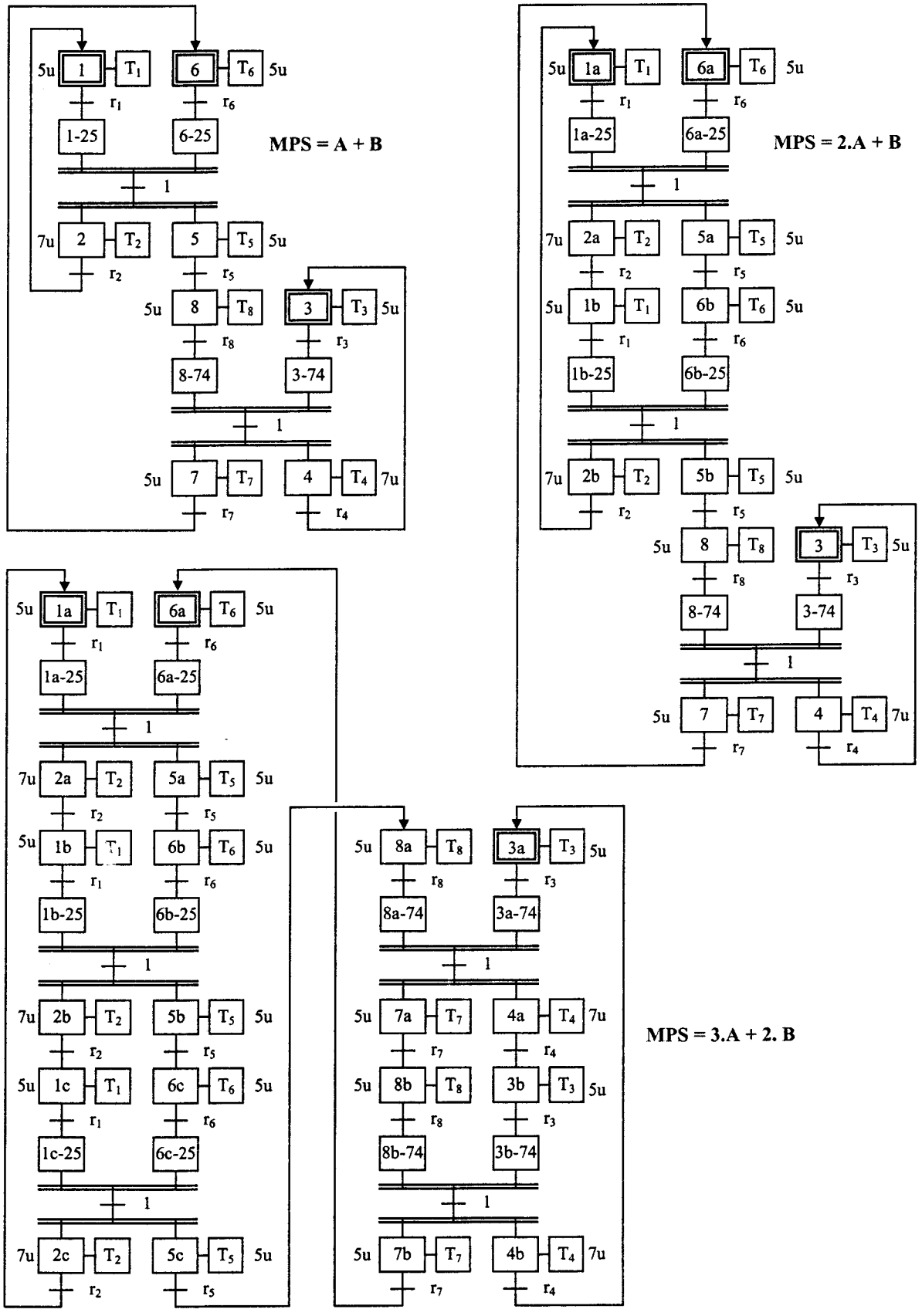


Figure 6 : Modèles Grafcet de commande pour les MPS (1,1), (2, 1) et (3, 2).

Les solutions proposées sur la *Figure 6* ne sont pas uniques, mais elles présentent l'avantage d'être généralisables (Cf. *Figure 7*). Sur cette figure, la suppression des deux parties centrales à itérer correspond au modèle Grafcet (x=1, y=1) de la *Figure 6.* La suppression de l'unique partie gauche correspond au modèle Grafcet (x=2, y=1) de la *Figure 6.* ....etc.

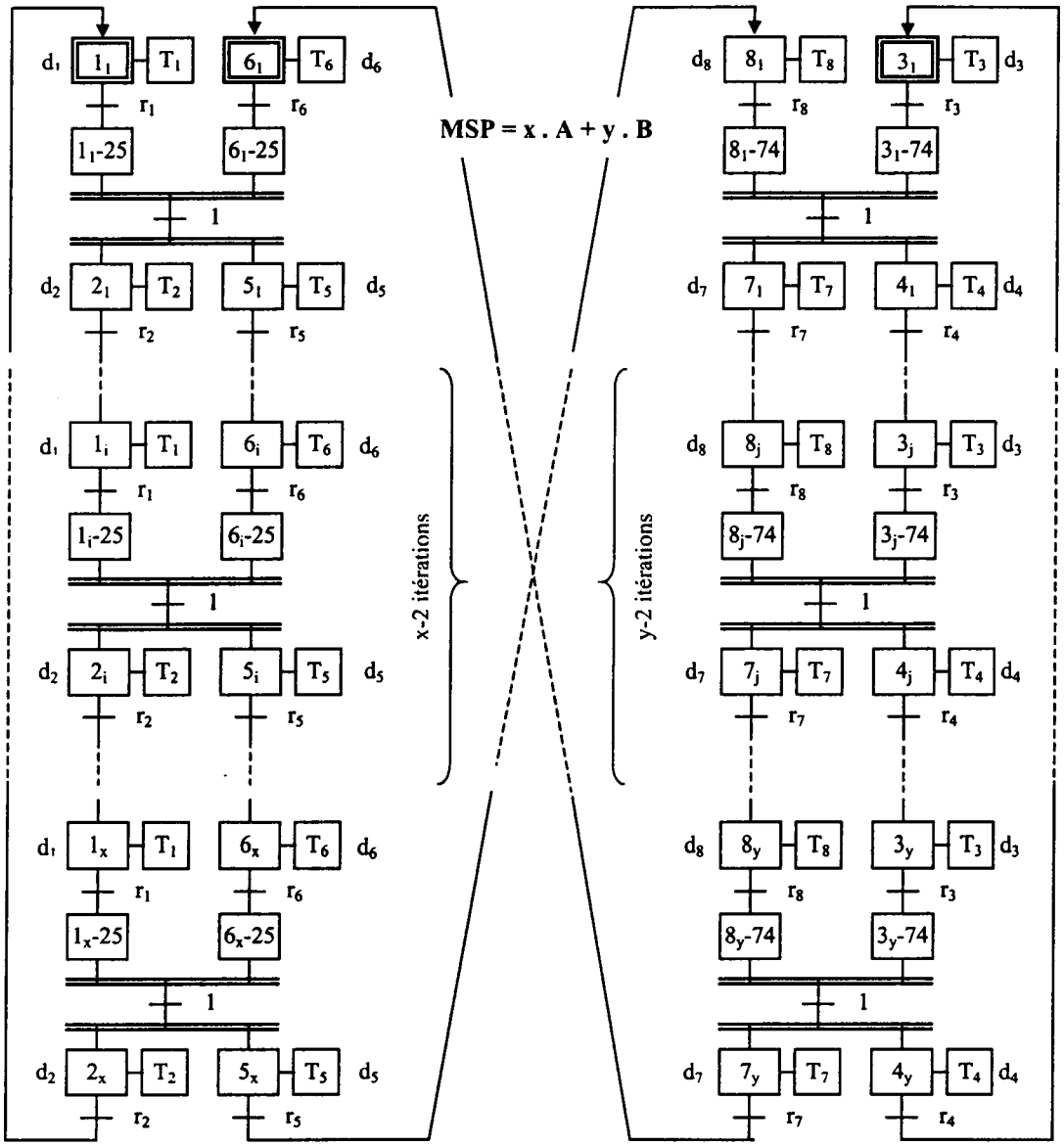


Figure 7 : Forme généralisée.

2.2.2. Suivis d'exécution.

L'étude des temps de cycle peut être faite par l'établissement des diagrammes de suivi (Cf. *Figure 8*) mais également par l'étude des graphes GMT associés puisque d'après la forme généralisée les modèles Grafcet sont de type événement. Nous établissons sur la *Figure 9* et *Figure 10,* les onze graphes GMT de l'application. Nous avons également représenté le MPS (4, 2) qui présente un temps de cycle nettement moins bon que le temps du MPS (2, 1).

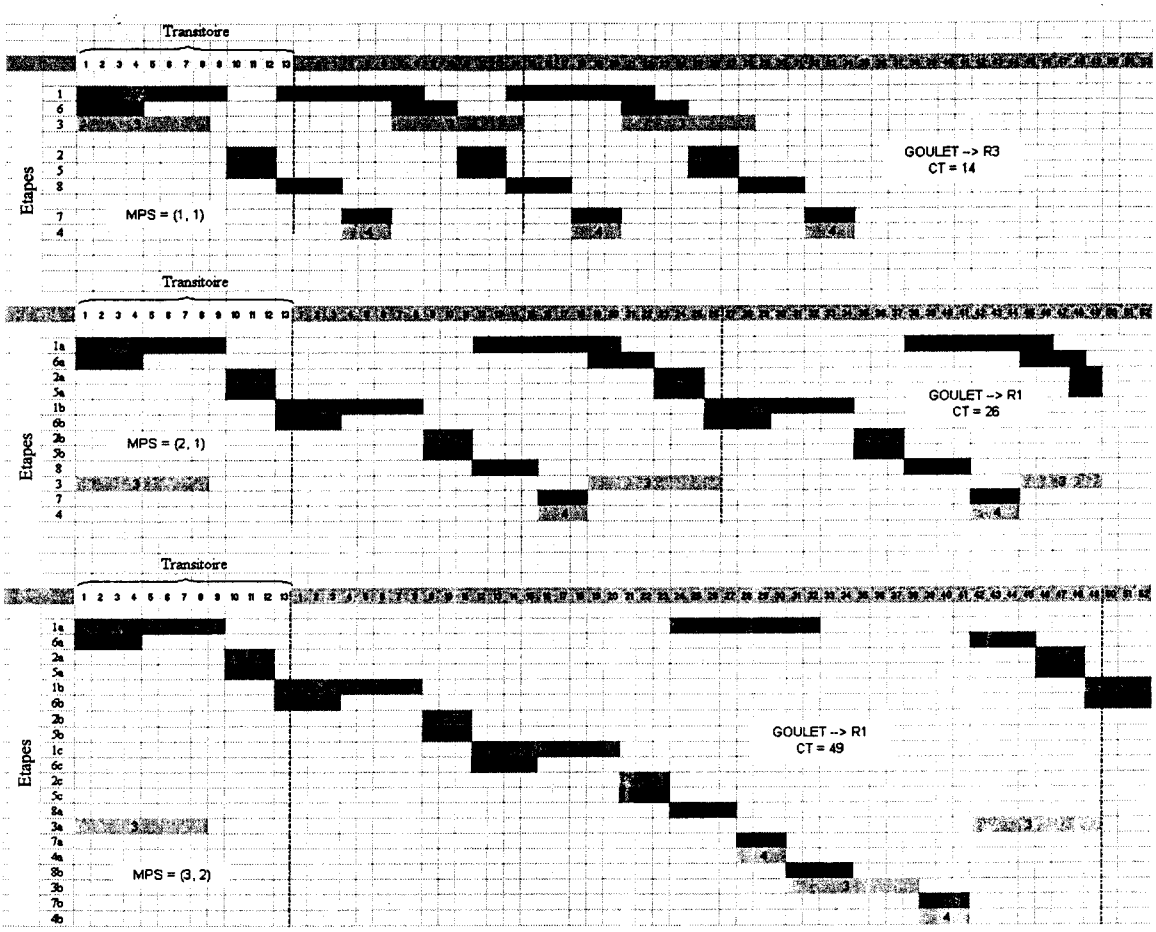


Figure 8 : Diagrammes de suivi des Grafcets pour les MPS (1, 1), (2, 1) et (3, 2).

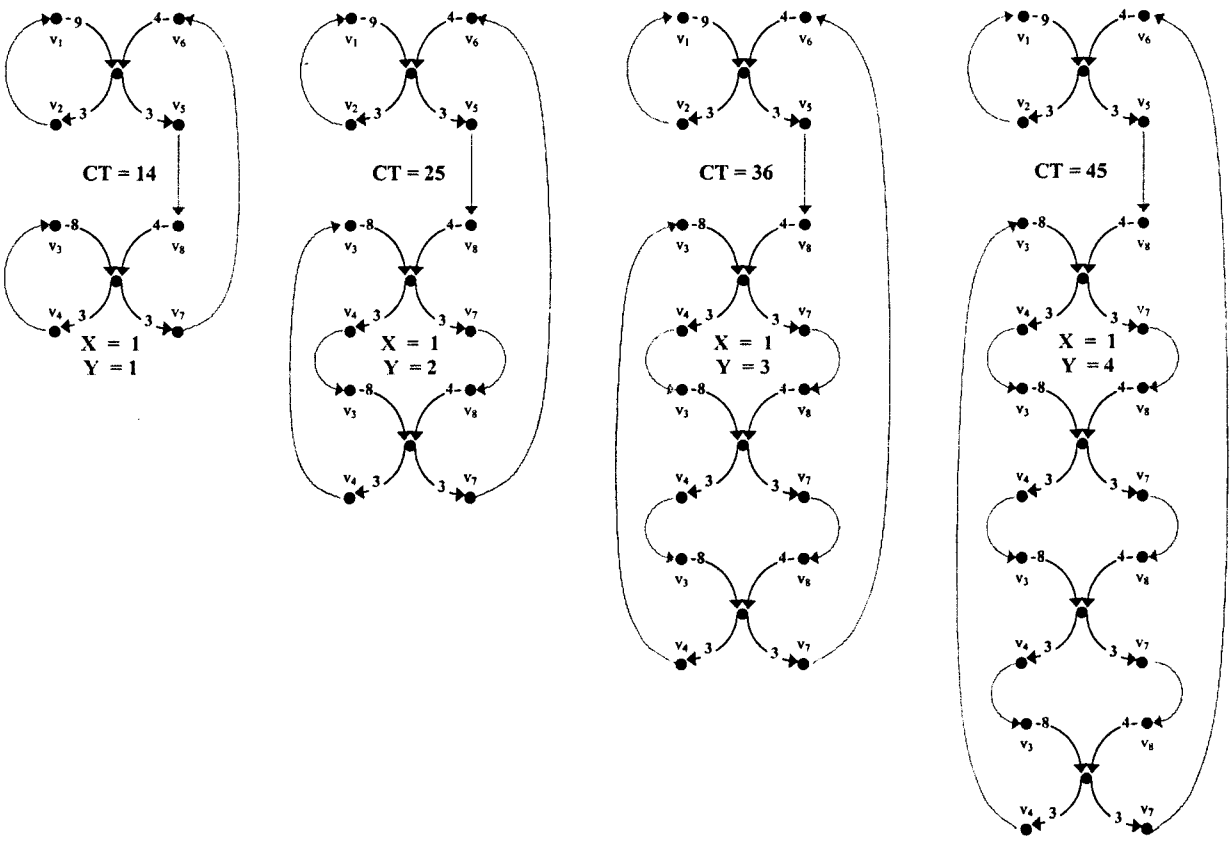


Figure 9 : Graphes GMT pour x =1 et y = 1, 2, 3 & 4.

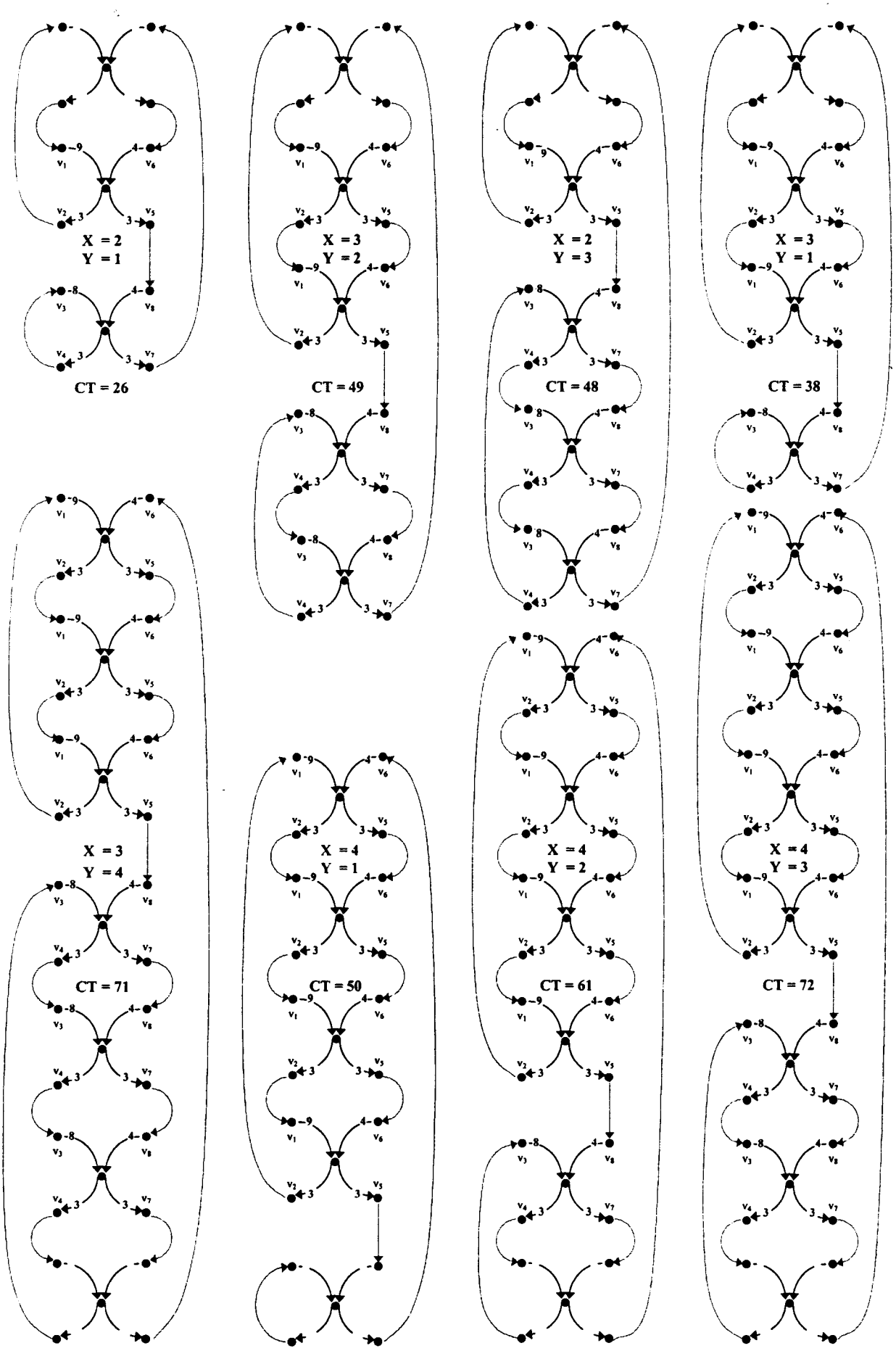


Figure 10 : Graphes GMT (suite et fin).

2.2.3. Résultats.

L'étude des circuits les plus lents des différents graphes GMT nous fournit les temps de cycle de l'application pour les différents MPS. Ces temps sont fonction des durées des tâches décrites sur la *Figure 1*et ils sont exprimés en unité de temps (Cf. *Tableau 3*). Les temps des MPS (2, 2), (3, 3) et (4, 4) sont obtenus en multipliant par *2*, *3* et *4* le temps associé au MPS (1, 1). Il en est de même pour les temps des MPS (2, 4) et (4, 2).

Dédiée	x = 1	x = 2	x = 3	x = 4
y = 1	14	26	38	50
y = 2	25	28	49	52
y = 3	36	48	42	72
y = 4	45	50	71	56

Tableau 3 : Temps de cycle des graphes multiples pour différentes MPS.

2.3. Commande multiple issue de l'ordonnancement cyclique.

2.3.1. Construction des gammes GMT initiales associées aux MPS.

La gamme opératoire initiale fait en principe partie du cahier des charges des problèmes qui nous préoccupent. Avant d'établir la gamme GMT d'un MPS, nous analysons les antériorités entre tâches qui ont été mises en place dans la solution connexe (§ *2.1*) et dans la solution multiple (§ *2.2*). Ces antériorités sont respectivement représentées pour le produit brut *A* sur la *Figure 11a* et sur la *Figure 11b* : elles ne peuvent pas être identiques. En effet et par définition, le graphe universel (cas *a*) ne connaît pas à l'avance l'ordre d'attribution de la ressource commune R_3, puisque cet ordre est fonction du MPS demandé. La tâche T_6 ne sera donc exécutée que si on a exécuté au préalable la tâche T_1 : il en est de même pour la tâche T_8 qui ne sera exécutée que si on a exécuté T_3. Dans le cas d'un graphe dédié dont nous connaissons le MPS (cas *b*), nous fixons l'ordre d'attribution de la ressource R_3, de fait, nous pouvons lancer les tâches T_1 et T_6 simultanément.

Figure 11 : Antériorités des solutions connexes et multiples.

Les deux politiques sont a priori applicables. On peut en premier lieu laisser l'algorithme gérer l'ordre d'attribution des ressources au sein d'une gamme MPS, ce qui permet d'explorer plus en profondeur l'espace des solutions. : mais il faut veiller à respecter les enchaînements

de tâches et en particulier les enchaînements séquentiellement indissociables (Cf. deuxième partie du mémoire - § *4.2.2.*).

Or, il se trouve dans notre exemple, que toutes les tâches appartiennent à des groupes de tâches séquentiellement indissociables : les tâches T_6 et T_5 sont séquentiellement indissociables : en effet, il est par exemple impossible d'exécuter séquentiellement la tâche T_6 puis la tâche T_8 : la tâche T_6 ne peut être suivie que par la tâche T_5. Il en est de même pour le groupe de tâches T_8 et T_7 (T_8 ne peut être suivie que par T_7), ainsi que pour le groupe T_1 et T_2 et le groupe T_3 et T_4. Le fusionnement des tâches indissociables conduirait à une forte réduction du nombre de tâches, ce qui limiterait l'intérêt d'une affectation libre des ressources. Nous avons donc choisi ici, la seconde solution qui consiste à imposer les enchaînements indissociables de la gamme GMT initiale représentés par les arcs horizontaux sur la *Figure 12*. Nous verrons par ailleurs (Cf. *Figure 21*) , qu'en fonction de la nature de la ressource goulet, certaines contraintes diagonales ($T_6 \rightarrow T_2$, $T_1 \rightarrow T_5$, $T_3 \rightarrow T_7$ ou $T_8 \rightarrow T_4$) peuvent être supprimées.

Figure 12 : Sous-gammes relatives aux éléments A' et B'.

La construction d'une gamme GMT initiale revient à juxtaposer dans les proportions du MPS les sous-gammes relatives aux éléments A' et B'(Cf. *Figure 12*). Sur la *Figure 13*, nous donnons à titre d'exemple la gamme GMT initiale associée au MPS(2,1). D'un point de vue notation, T_{ij} représente l'i^{ème} tâche du j^{ème} élément *A'* ou *B'*. Les numéros en italique associés à chaque nœud indiquent les numéros utilisés par le simulateur.

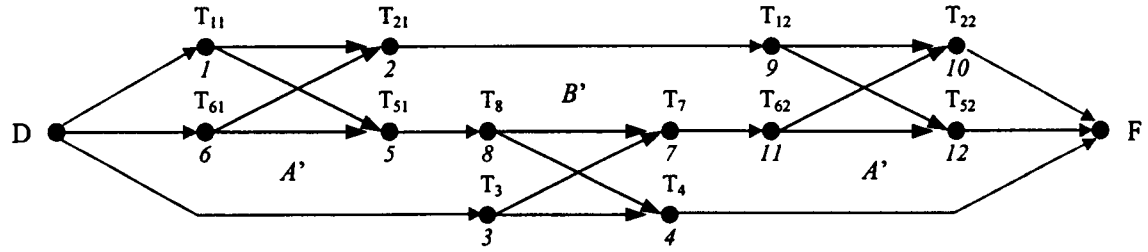


Figure 13: Gamme GMT initiale associée au MPS(2,1).

Il existe plusieurs façons de juxtaposer les sous-gammes élémentaires (bien que pour cet exemple et d'un point de vue cyclique, il n'en existe qu'une). Dans le cas où les ressources R_1 ou R_2 seraient les ressources goulet, cet ordre n'a pas d'influence sur la gamme décyclée car la séquence critique reste inchangée. Dans le cas où la ressource goulet est la ressource R_3, et

afin *d'aider* le simulateur, nous alternerons autant que possible les sous-gammes associées aux éléments *A'* et *B'*.

Pour chaque configuration de MPS, nous pouvons prévoir quelle sera la ressource goulet (Cf. *Tableau 4*). Cela peut être la ressource R_1, R_2 ou R_3, tout dépend du nombre d'éléments *A'* et *B'* constituant le produit fini.

x éléments A'	1	1	1	1		2	2	2	2		3	3	3	3		4	4	4	4
y éléments B'	1	2	3	4		1	2	3	4		1	2	3	4		1	2	3	4
(x+y) utilisations de R3	2	3	4	5		3	4	5	6		4	5	6	7		5	6	7	8
Temps d'engagement de R1	12	12	12	12		24	24	24	24		36	36	36	36		48	48	48	48
Temps d'engagement de R2	11	22	33	44		11	22	33	44		11	22	33	44		11	22	33	44
Temps d'engagement de R3	14	21	28	35		21	28	35	42		28	35	42	49		35	42	49	56
Désignation du goulet	R3	R2	R2	R2		R1	R3	R3	R2		R1	R1	R3	R3		R1	R1	R3	R3

Tableau 4 : Définition de la ressource goulet en fonction du MPS.

Le *Tableau 5* récapitule l'ordre de juxtaposition que nous avons arbitrairement choisi pour les différents MPS. De la même façon que pour la commande basée sur des graphes dédiés (Cf. paragraphe *2.2*), les cas multiples ne seront pas étudiés, puisqu'il suffit au niveau de la simulation de rallonger l'horizon temporel : réaliser 10 fois le MPS (4, 2) revient à réaliser 20 fois le MPS (2, 1).

(1,1)	(2,1)	(3,1)	(4,1)	(1,2)	(2,2)	(3,2)	(4,2)	(1,3)	(2,3)	(3,3)	(4,3)	(1,4)	(2,4)	(3,4)	(4,4)
A'	A'	A'	A'	B'	A'	A'	A'	B'	B'	A'	A'	B'	B'	B'	A'
B'	B'	B'	B'	A'	B'	B'	B'	A'	A'	B'	B'	A'	A'	A'	B'
	A'	A'	A'	B'		A'	A'	B'	B'		A'	B'	B'	B'	
		A'	A'			B'		B'	A'		B'	B'		A'	
			A'			A'			B'		A'	B'		B'	
					idem		idem			idem	B'		idem	A'	idem
					(1,1)		(2,1)			(1,1)	A'		(1,2)	B'	(1,1)

Tableau 5 : Composition des différentes gammes MPS initiales.

2.3.2. Obtention des gammes décyclées.

Chaque gamme décyclée est obtenue à partir de la simulation d'un exemplaire unique de la gamme GMT initiale correspondante. Ainsi, la simulation de la gamme GMT de la *Figure 13* fournit le diagramme de GANTT de la *Figure 14* duquel nous déduisons la gamme décyclée (Cf. *Figure 15*).

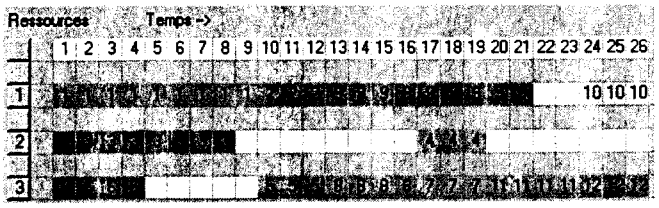


Figure 14 : Simulation de la gamme GMT initiale du MPS(2,1).

Les principes d'établissement présentés dans le paragraphe *4.2.2.* de la deuxième partie du mémoire permettent d'obtenir toutes les gammes décyclées de l'application. Elles sont représentées sur la *Figure 15* et sur la *Figure 16.*

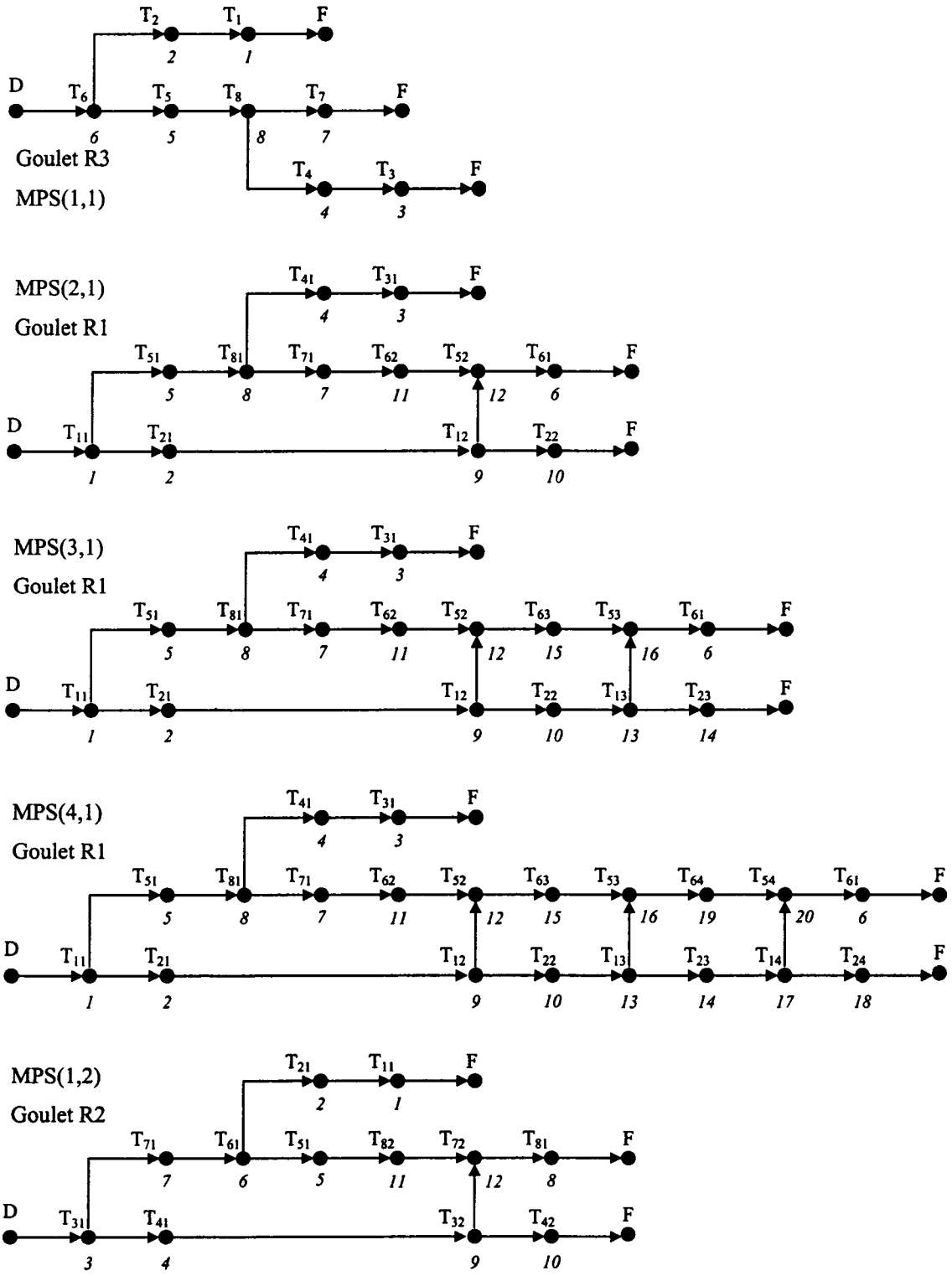


Figure 15 : Gammes décyclées pour les MPS (1,1) à (4,1) et pour le MPS(1,2).

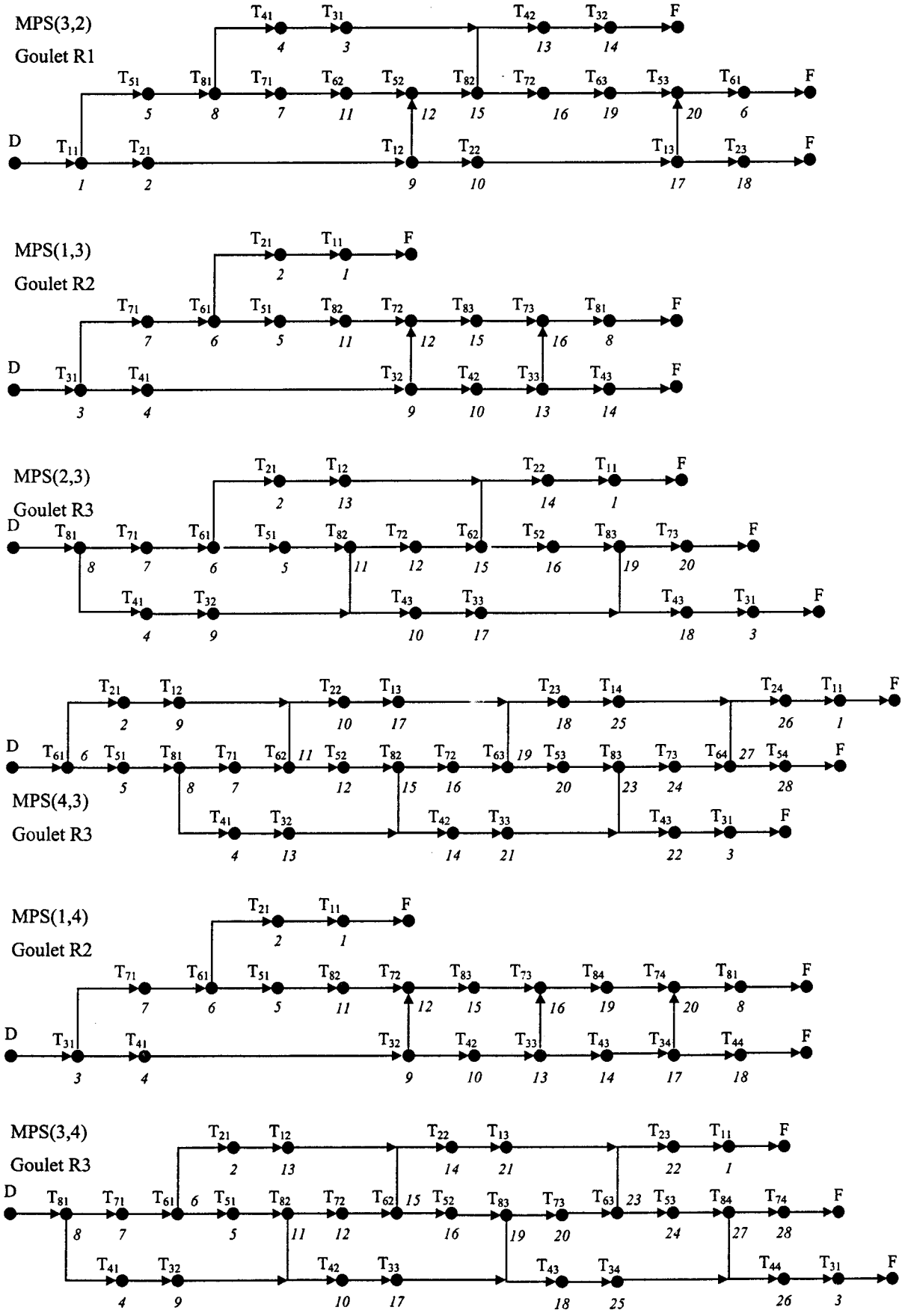


Figure 16 : Gammes décyclées pour les MPS de (3,2), (1,3), (2,3), (4,3), (1,4) et (3,4).

2.3.3. Résultats de simulation.

Les résultats de simulation sont présentés sur la *Figure* 17, *Figure 18, Figure 19* et *Figure 20.*

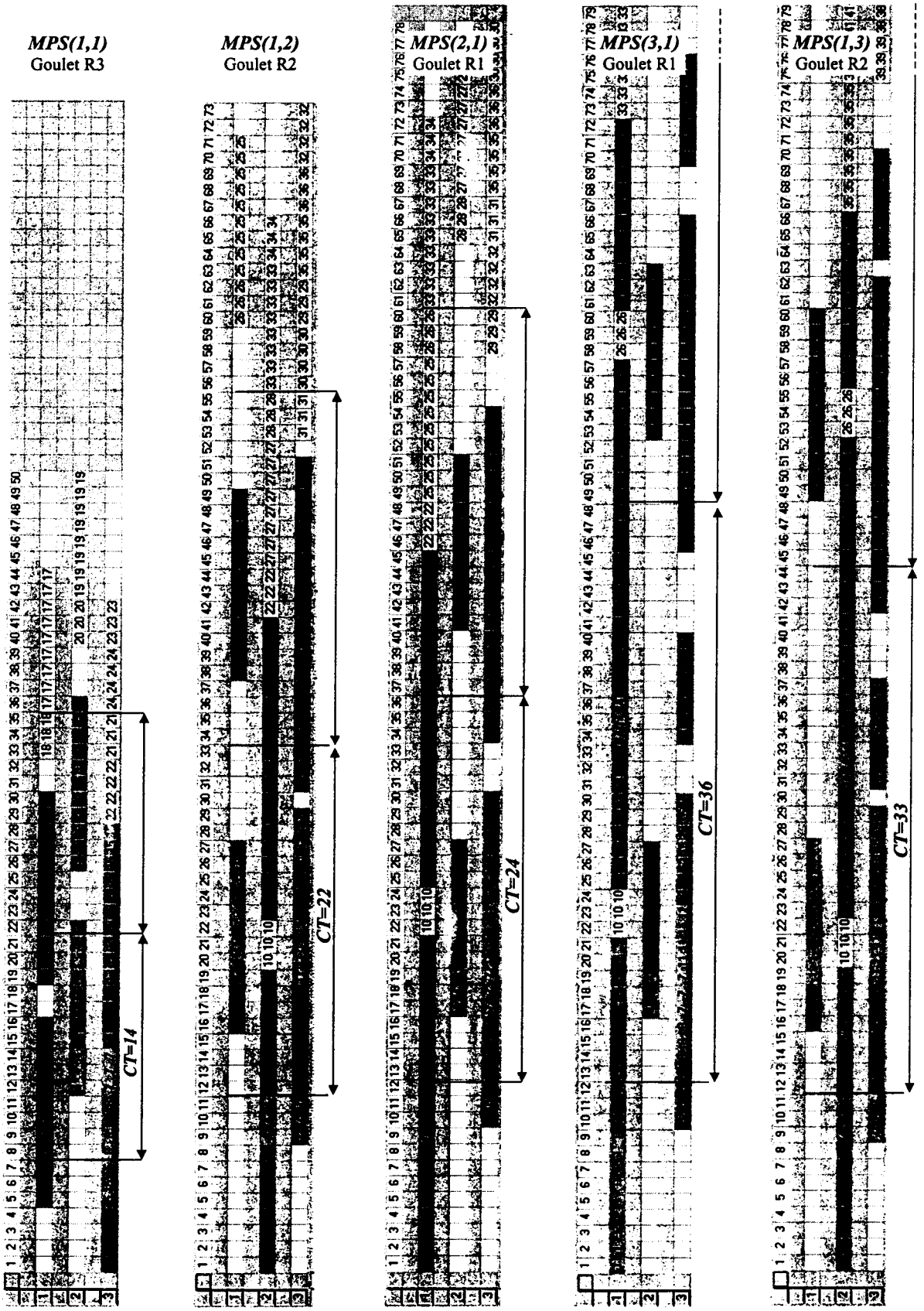


Figure 17 : Diagrammes de GANTT des MPS(1,1), MPS(1,2), MPS(2,1), MPS(3,1) & MPS(1,3)

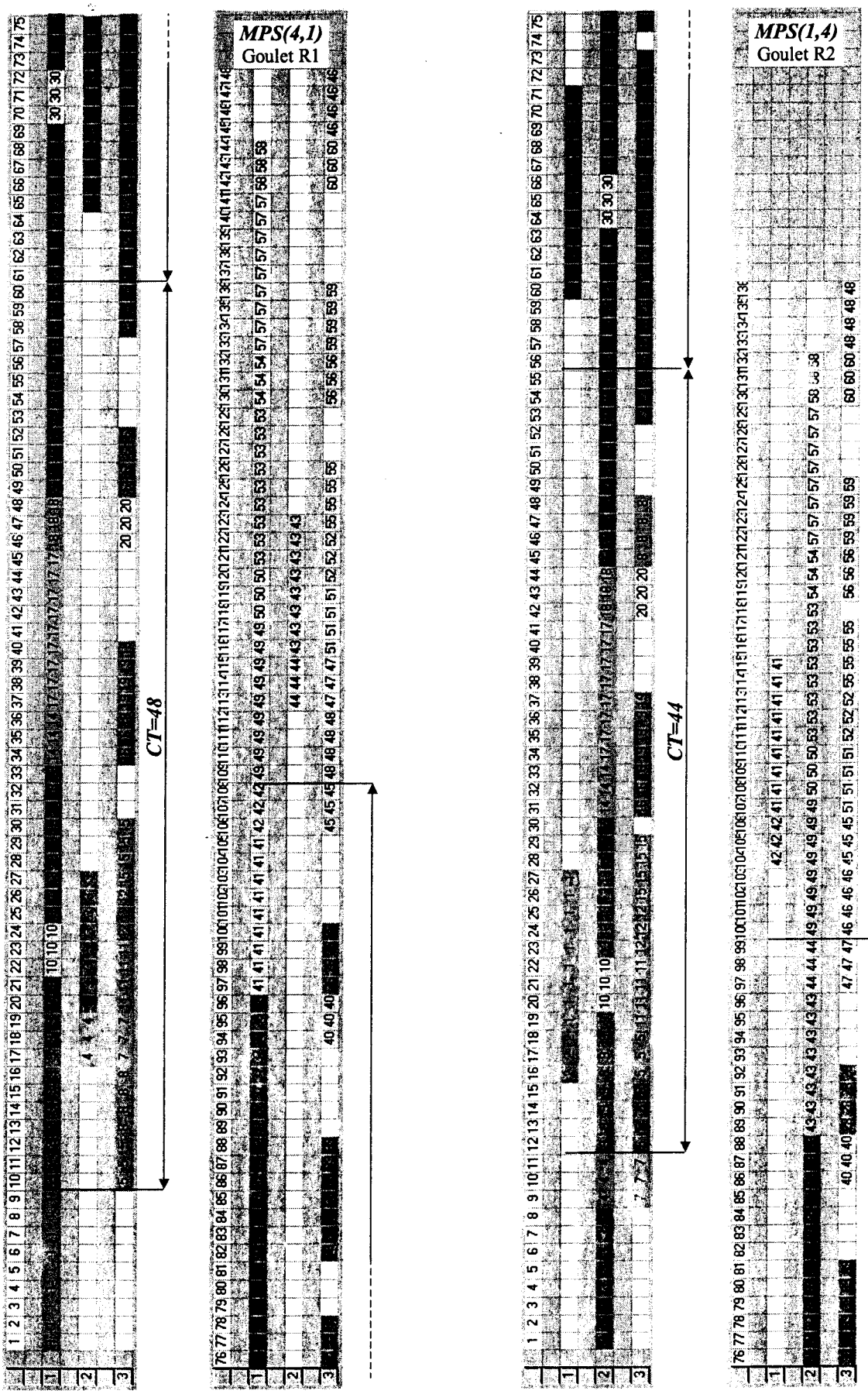


Figure 18: Diagrammes de GANTT pour le MPS(4,1) et pour le MPS(1,4).

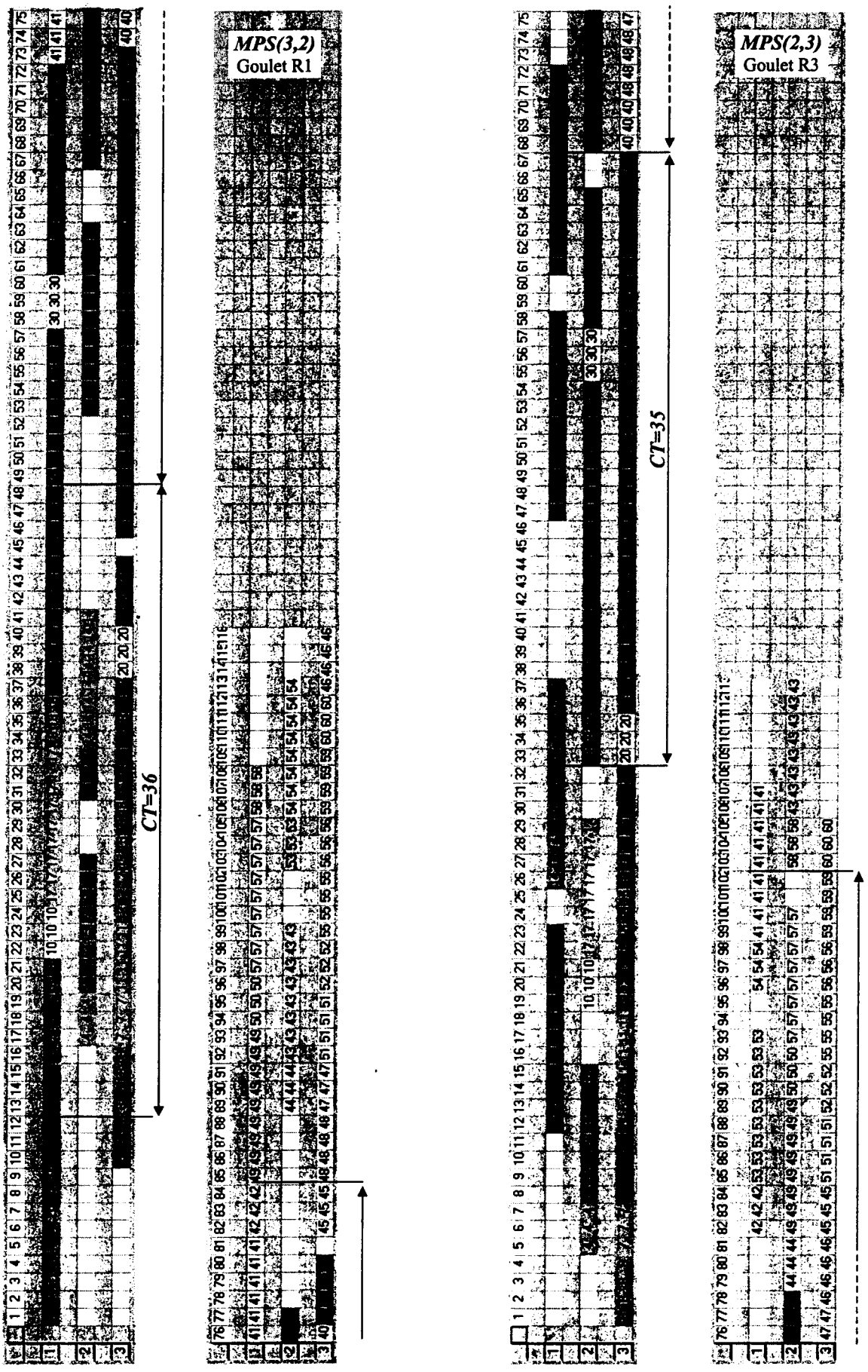


Figure 19: Diagrammes de GANTT pour le MPS(3,2) et pour le MPS(2,3).

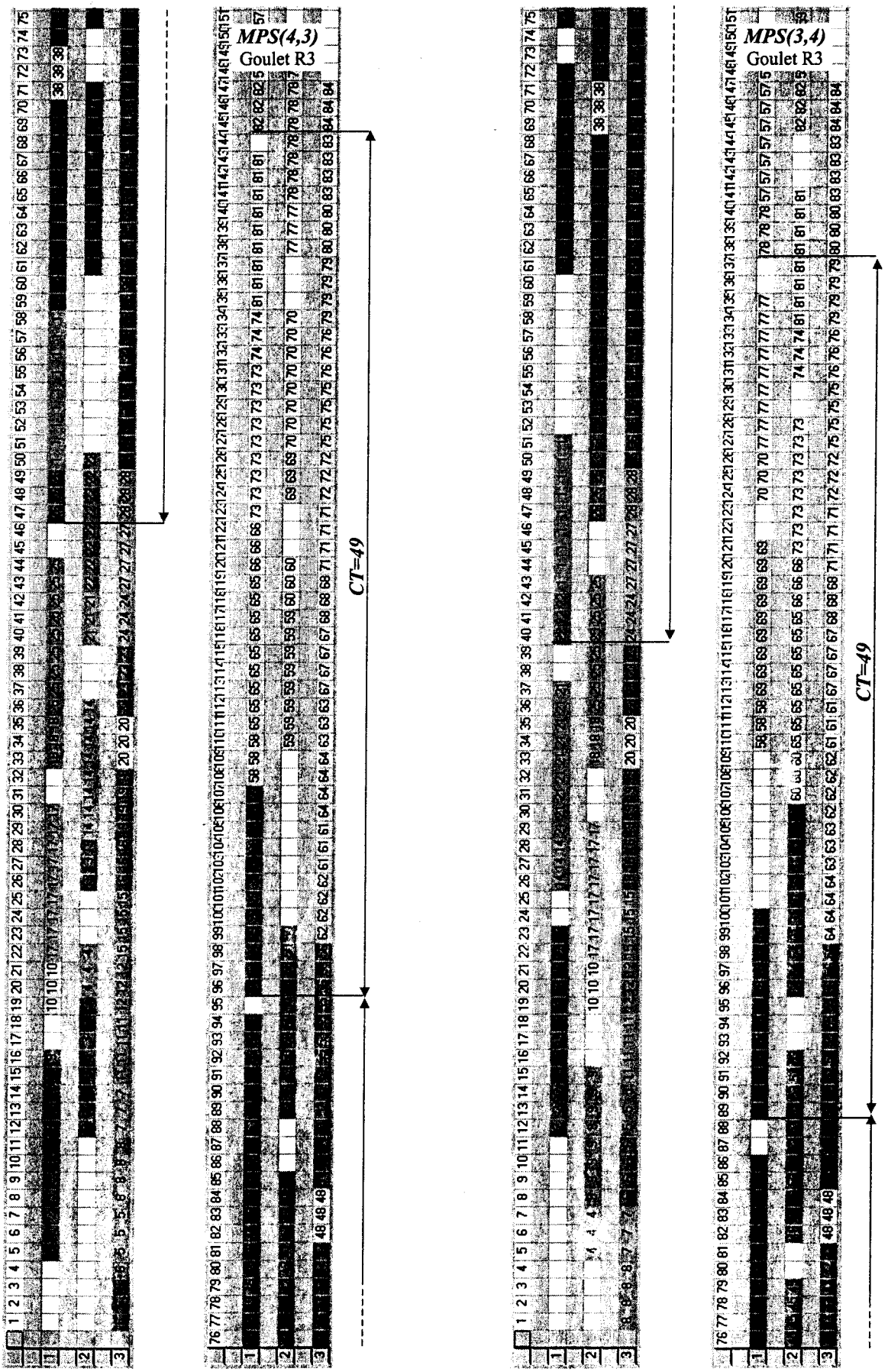


Figure 20 : Diagrammes de GANTT pour le MPS(4,3) et pour le MPS(3,4).

➢ *Exploitation. Limitation des contraintes.*

Le simulateur produit un ordonnancement qui sature la ressource goulet et celle-ci reste saturée à la condition qu'aucune des tâches qu'elle exécute ne se trouve en position d'attente.

L'étude des différents diagrammes de GANTT conduit à deux solutions :

a) Soit on respecte l'ensemble des contraintes représentées sur la *Figure 12*. Les problèmes d'attente seront alors résolus par la prise en compte d'un nombre suffisant d'encours (deux ou trois encours suivant le MPS concerné).

b) Soit on ne respecte pas toutes les contraintes que nous nous étions imposées au départ (hormis les contraintes de tâches séquentiellement indissociables) et l'encours se trouve au sens de notre définition limité à un exemplaire.

Les contraintes imposées par la *Figure 12* représentent l'union de toutes les contraintes qui permettent de garantir le bon fonctionnement de l'application, quelle que soit la ressource goulet. Si maintenant, nous tenons compte de cette ressource, nous remarquons qu'une partie seulement de ces contraintes est nécessaire. En effet, lorsque par exemple la ressource R_2 est la ressource goulet (cas du MPS(1,2)), il est inutile d'attendre la fin d'une tâche T_8 pour lancer la tâche T_4: la contrainte $T_8 \rightarrow T_4$ peut donc être supprimée. De même, lorsque la ressource R_1 est la ressource goulet (cas du MPS(2,1)), il est inutile d'attendre la fin d'une tâche T_6 pour lancer la tâche T_2, ce qui revient à supprimer la contrainte $T_6 \rightarrow T_2$. A l'inverse, lorsque R_3 cadence la ressource R_1 (respectivement R_2), la contrainte $T_1 \rightarrow T_5$ (respectivement $T_3 \rightarrow T_7$) peut être également supprimée. Ce relâchement des contraintes en fonction de la ressource goulet est présenté sur la *Figure 21*.

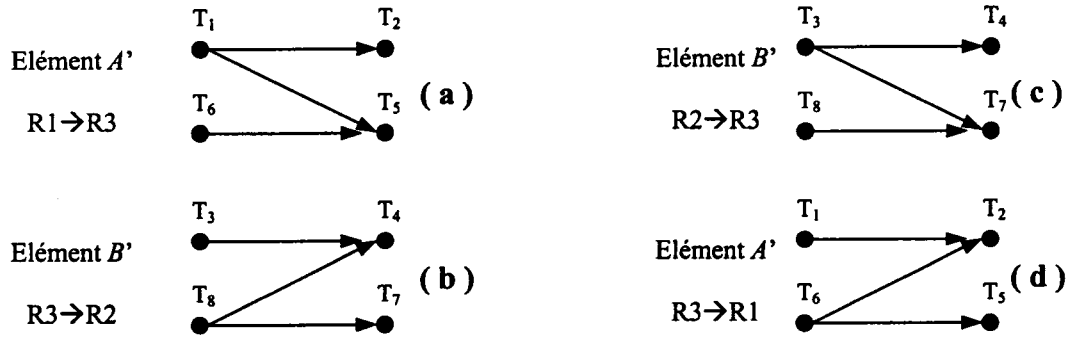


Figure 21 : Relâchement de contraintes en fonction du goulet.

Les cas (a) et (b) de la *Figure 21* représentent les contraintes minimales que doit respecter un MPS où la ressource goulet est la ressource R_1. Les cas (c) et (d) de cette même figure sont associés à un MPS qui a la ressource R_2 comme ressource goulet. Enfin les cas (b) et (d) concernent un MPS dont la ressource goulet est la ressource R_3.

2.3.4. Etablissement des Grafcets de commande.

La présence d'un seul encours, quels que soient les cas d'étude, simplifie l'établissement des Grafcets de commande et nous oriente naturellement vers des solutions connexes.

Les matrices *A1* et *A2* de la *Figure 22* représentent respectivement les contraintes de gamme et les contraintes d'ordonnancement. La matrice *A* est la somme de ces deux matrices. Toutes les contraintes de gamme sont incluses dans les contraintes d'ordonnancement.

A1	T1	T2	T3	T4	T5	T6	T7	T8
T1								
T2	A					A		
T3								
T4			B					B
T5						A		
T6								
T7								B
T8								

A2	T1	T2	T3	T4	T5	T6	T7	T8
T1		1						
T2	1					31		
T3				2				
T4			2					32
T5						3		
T6							3	
T7								3
T8					3			

A	T1	T2	T3	T4	T5	T6	T7	T8
T1		1						
T2	1					1		
T3				1				
T4			1					1
T5						1		
T6							1	
T7								1
T8					1			

Figure 22: Matrices d'antériorités relatives au MPS(1,1)

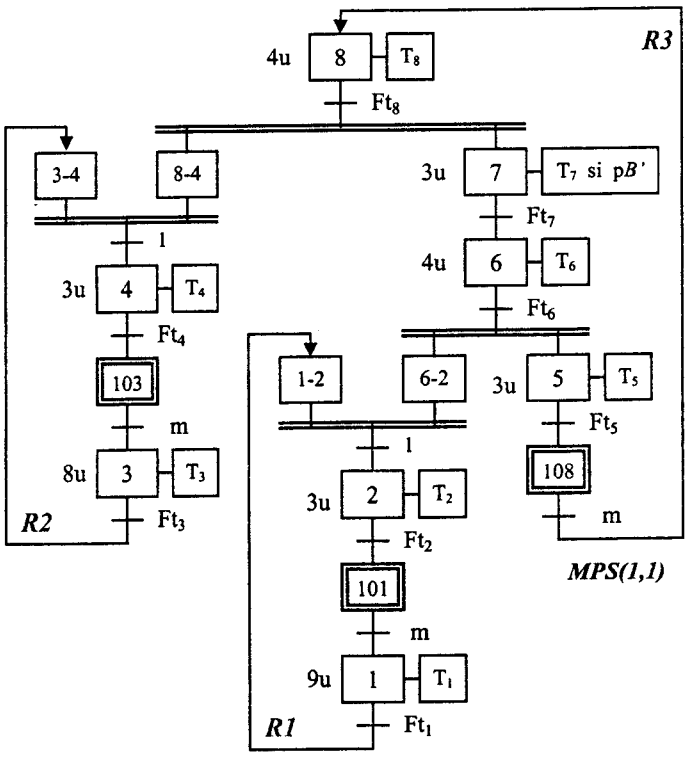


Figure 23 : Grafcet de commande du MPS(1,1)

La *Figure 23* représente une solution *poussée* par la ressource goulet R_3 du Grafcet de commande correspondant au MPS(1,1). Il est obtenu à partir de la matrice *A* par la passerelle GMT→Grafcet.

Les trois étapes (*101*), (*103*) et (*108*) placées en amont des étapes (*1*), (*3*) et (*8*) sont les étapes initiales de dernier Grafcet (Grafcet vivant). L'application est en service lorsque le booléen *m* est vrai. Afin d'éviter un premier aller et retour inutile de la ressource R_3, nous attendons la présence d'un élément *B'* au point A_4 (information p*B'*), avant lancer la première tâche T_7.

En suivant la même procédure nous établissons les Grafcets de commande des autres configurations MPS. Nous donnons ci-après les Grafcets de commande pour les MPS(1,2), MPS(3,1), MPS(2,3) et MPS (3,4) (Cf. *Figure 24* à *Figure 27*). Les architecture des Grafcets associés aux MPS(2,1), MPS(3,2) et MPS(4,3) sont identiques à celles des MPS(3,1), MPS(2,3) et MPS (3,4). Les Grafcets correspondants sont obtenus en inversant les rôles joués par les ressources R_1 et R_2. Les Grafcets des MPS(1,4) et MPS(4,1) sont des prolongations des Grafcets des MPS(1,3) et MPS(1,4).

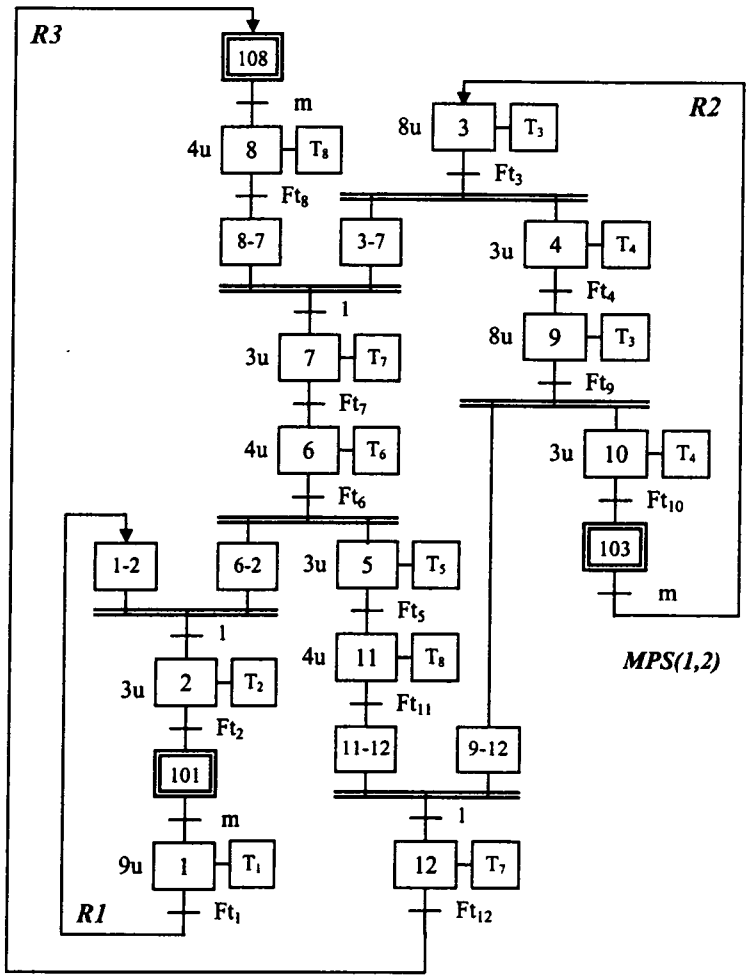


Figure 24 : Grafcet de commande du MPS(1,2).

La ressource goulet R_2 de la *Figure 24* 'pousse' la ressource R_3 qui elle-même cadence la ressource R_1. Le circuit le plus lent est associé à la ressource R_2 et il a une durée de *22 unités*

de temps. Le Grafcet du MPS(2,1) a la même architecture mais c'est la ressource R1 qui est la ressource goulet (durée, *24 unité de temps*).

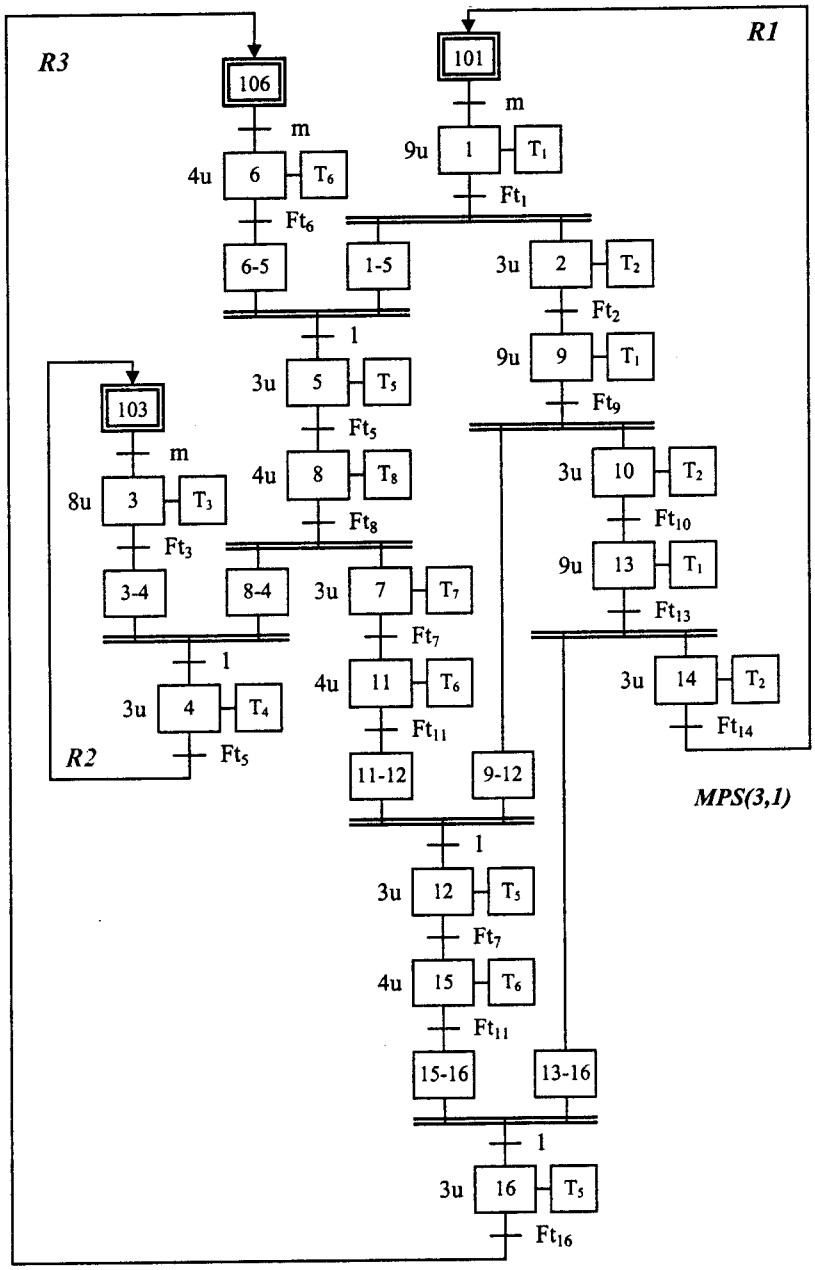


Figure 25 : Grafcet de commande du MPS(3,1).

La ressource goulet R_1 'pousse' la ressource R_3 qui elle-même cadence la ressource R_2. Le circuit le plus lent est associé à cette ressource (*36 unités de temps*). Bien que la ressource goulet ne soit plus la ressource R_2, l'architecture de Grafcet de la *Figure 25* est un prolongement du Grafcet du MPS(1,2). Les Grafcets des MPS(1,4) et MPS(4,1) sont aussi des extensions de ce Grafcet.

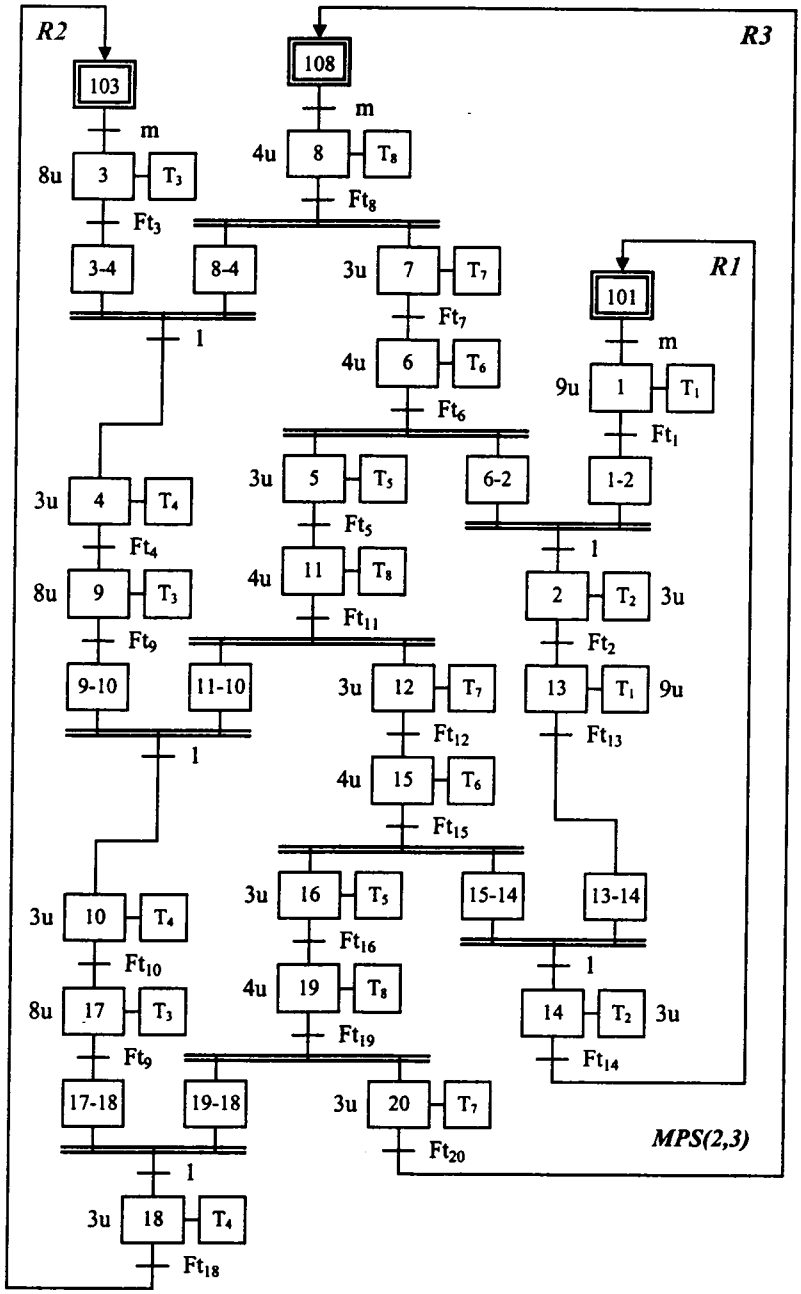


Figure 26 : Grafcet du MPS(2,3).

La ressource goulet R_3 cadence alternativement les ressources R_1 et R_2. Les tâches de type T_4 attendent les tâches de type T_8 et les tâches de type T_2 attendent les tâches de type T_6. Le circuit le plus lent est associé à la ressource R_3, sa durée est de *35 unités de temps* (Cf. *Figure 26*).

Par contre, le MPS(3,2) n'a pas la même architecture puisque c'est la ressource R_1 qui devient la ressource goulet. Le temps de son circuit le plus lent est de *36 unités de temps*. La structure du MPS(3,2) est très proche de celle du MPS (3,1) : le deuxième groupe de tâche T_3 / T_4 est masqué par l'activité de la ressource R_1.

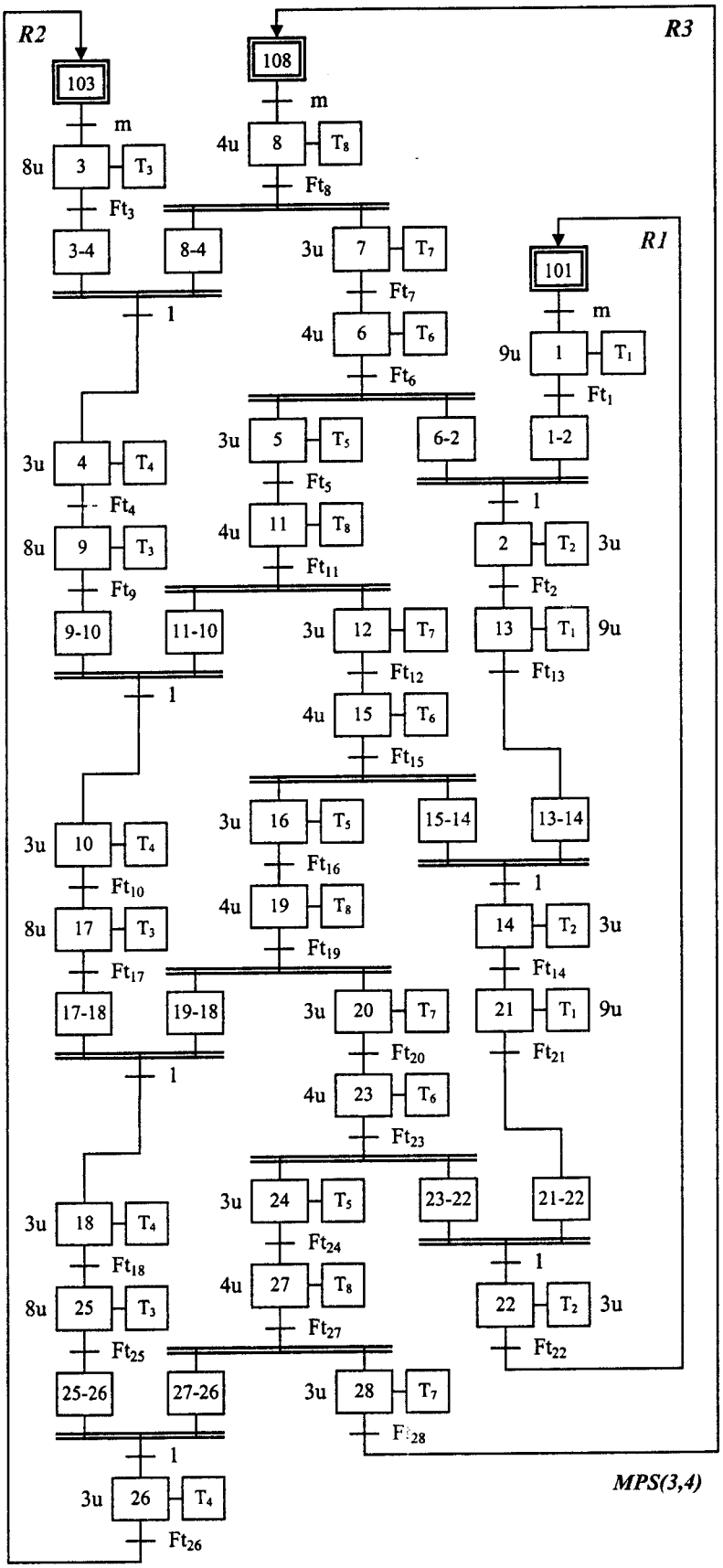


Figure 27 : Grafcet du MPS(3,4).

La ressource goulet R_3 cadence également R_1 et R_2. L'architecture est elle aussi un prolongement de celle du Grafcet du MPS(2,3) (Cf. *Figure 27*). Le temps du circuit le plus lent est le même que pour le MPS(4,3), soit *49 unités de temps*.

2.3.5. Suivis d'exécution.

Les suivis d'exécution permettent de vérifier les temps de cycle de chaque MPS, mais aussi de comptabiliser l'encours.

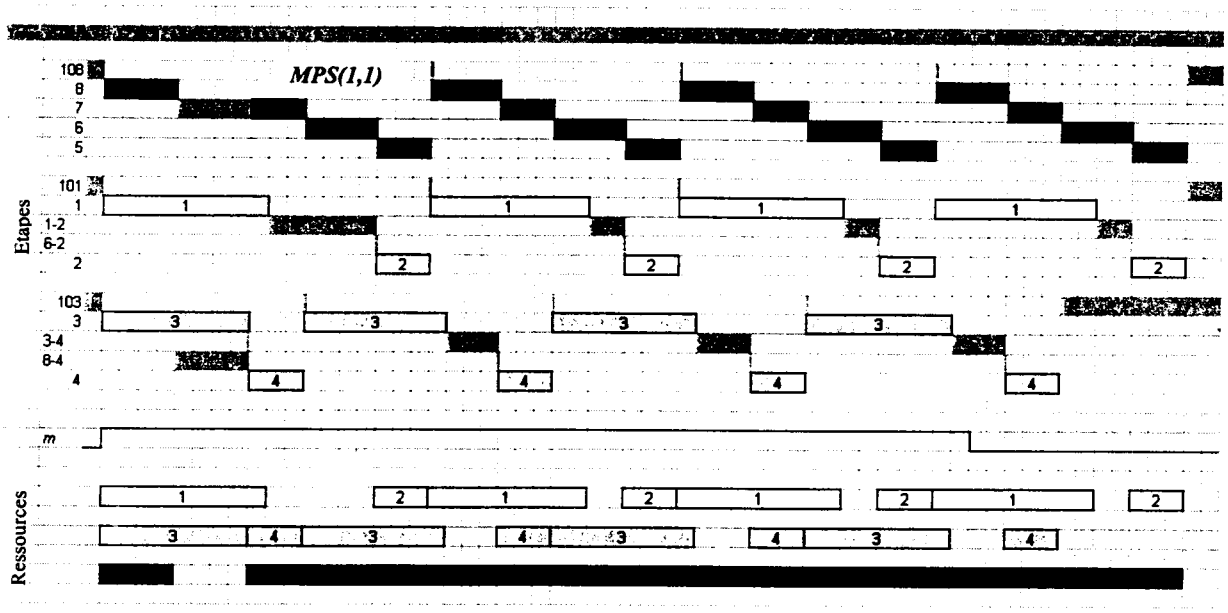


Figure 28 : Suivi d'exécution du MPS(1,1).

L'activité représentée sur la *Figure 28* des étapes (*1-2*) et (*3-4*) correspond au temps de présence respectif des éléments *A'* et *B'* sur les aires de stockages A_2 et A_4.

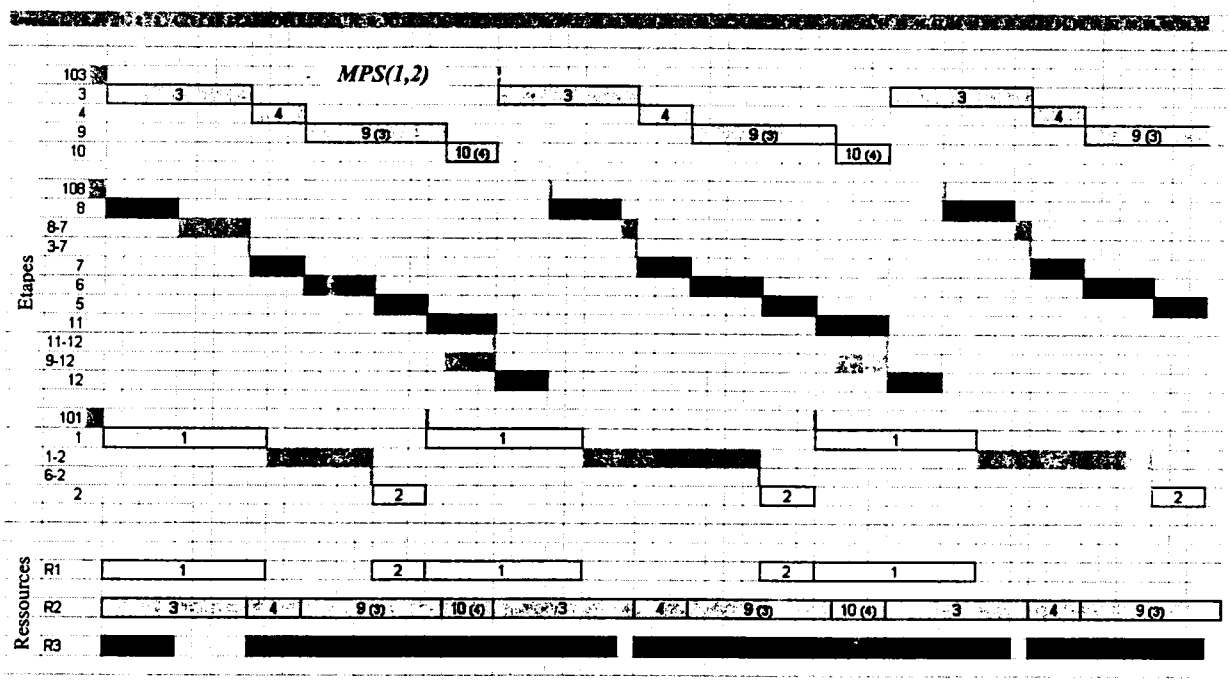


Figure 29 : Suivi d'exécution du MPS(1,2).

L'activité représentée sur la Figure 29 des étapes (*1-2*) et (*8-7*) correspond au temps de présence respectif des éléments *A'* et *B'* sur les aires de stockages A_2 et A_4.

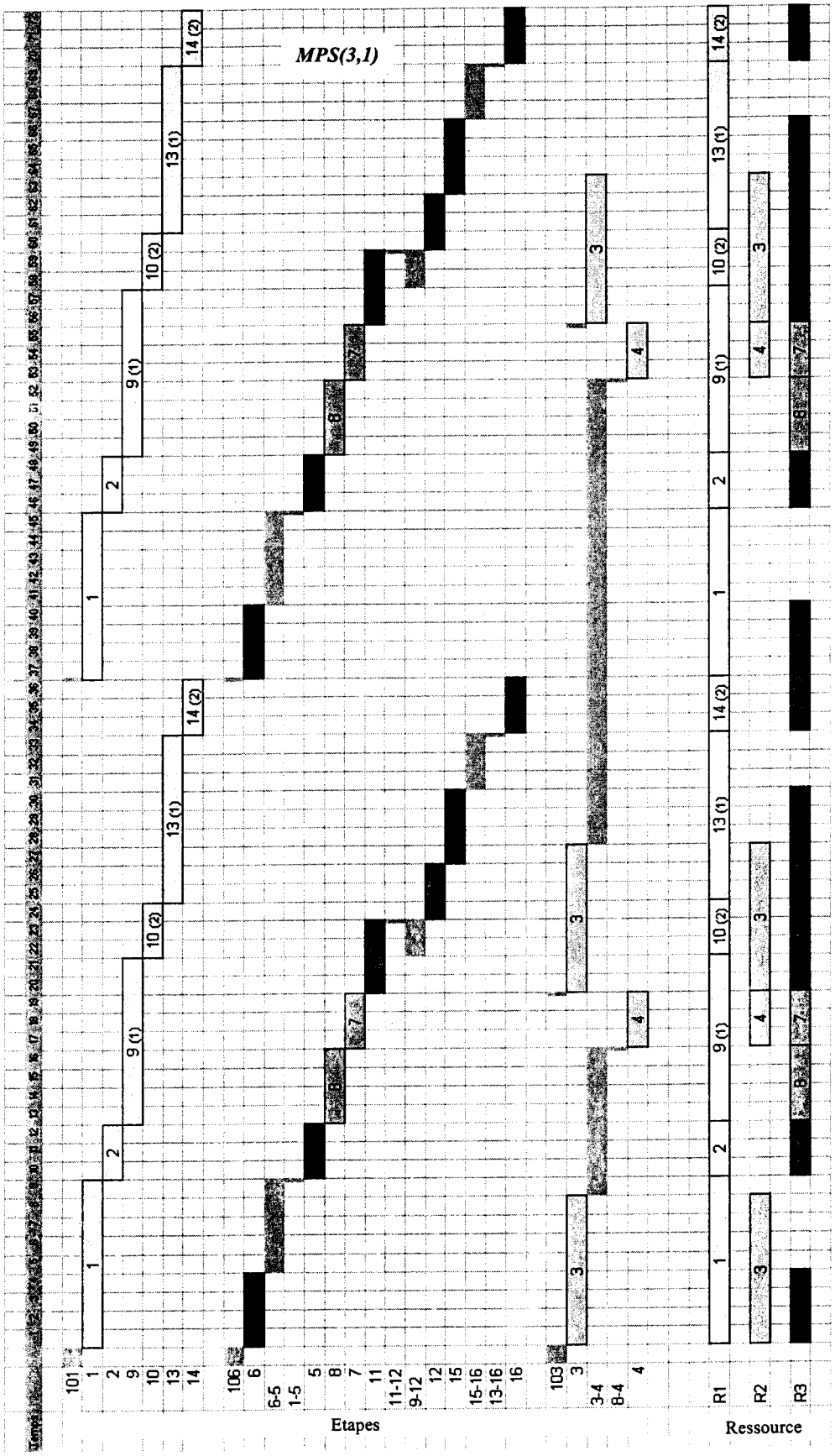


Figure 30 : Suivi d'exécution du MPS(3,1).

L'activité représentée sur la *Figure 30* des étapes (*9-12*) et (*3-4*) correspond au temps de présence respectif des éléments *A'* et *B'* sur les aires de stockages A_2 et A_4.

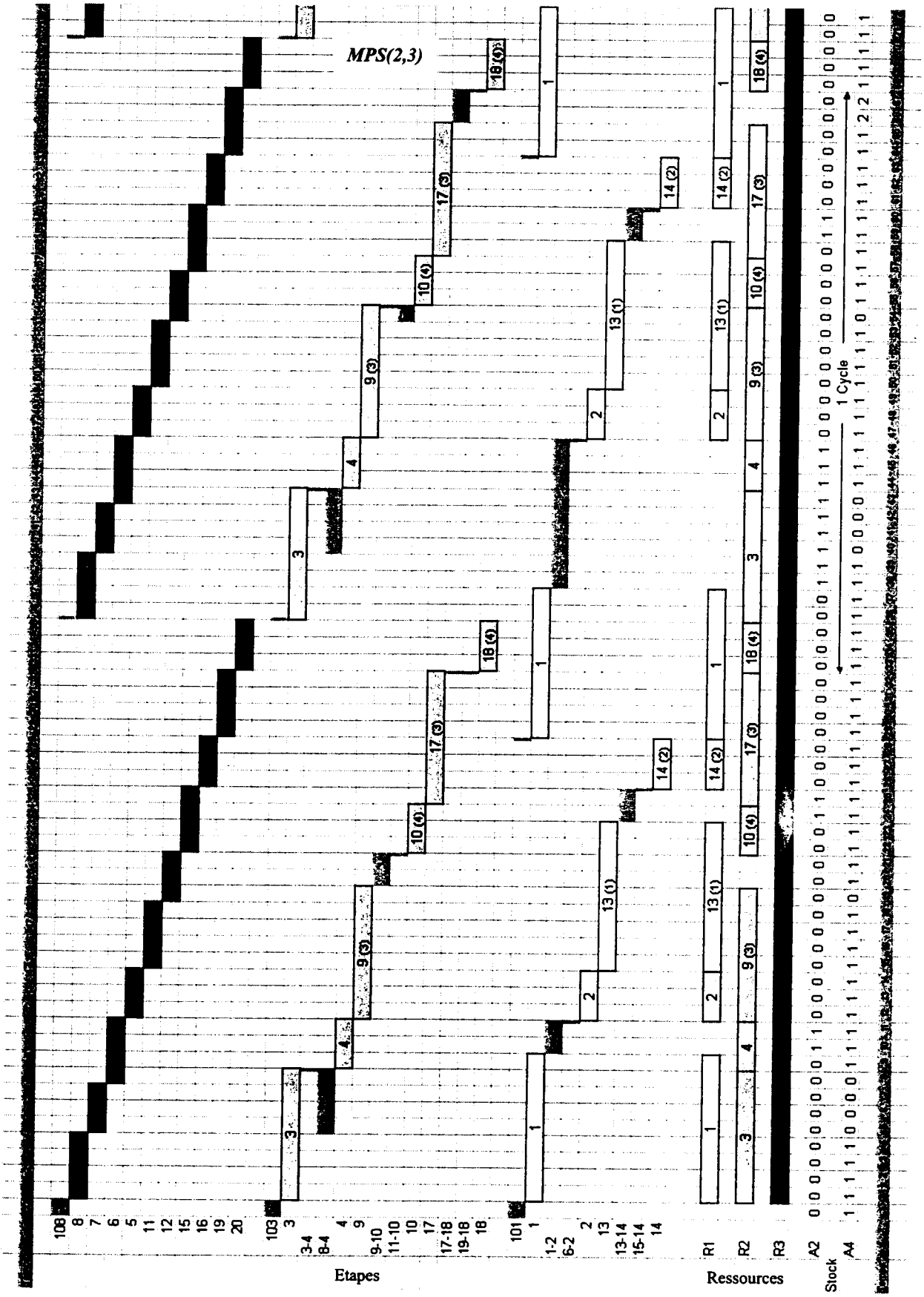


Figure 31 : Suivi d'exécution du MPS(2,3).

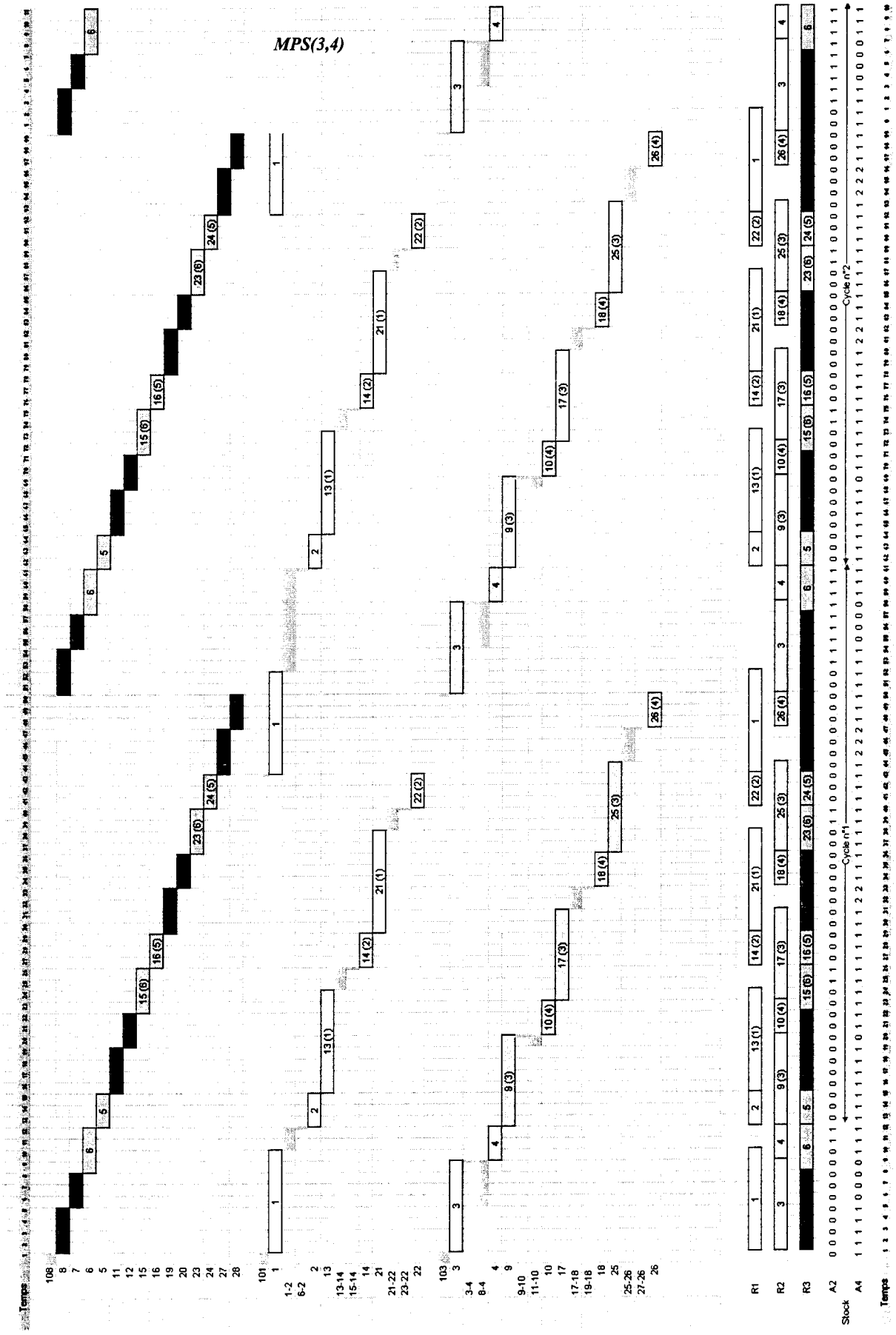


Figure 32 : Suivi d'exécution du MPS(3,4).

➢ *Comptabilité de l'encours.*

Les éléments sont comptabilisés dans l'état où ils se trouvent à la fin de l'exécution de la tâche en cours (T_1 et T_3), excepté les tâches qui élaborent un produit fini (T_5 et T_7). Pour que l'encours *Xe* soit égal à l'unité, l'encours élémentaire $xe_j(t)$ doit être quel que soit l'instant *t*, inférieur ou égal à ye_j, ye_j étant le nombre d'éléments de la classe d'équivalence *A'* ou *B'*.

Pour les MPS(1,*i*) ou MPS(*i*,1), $i \in [1..4]$ et $\forall\ t$, $xe_j(t)$ est égal à *un*. *Xe* vaut donc *un*.(Cf. *Figure 28* à *Figure 30*).Pour les autres cas de MPS, l'encours élémentaire $xe_j(t)$ n'est plus toujours égal à *un*.

Examinons le cas du MPS(2,3), (Cf. *Figure 29*). Concernant l'élément *A'*, les tâche T_1 et T_5 ne travaillent jamais en même temps, et le stock intermédiaire A_2 contient un élément *A'* lorsque T_1 et T_5 sont inactives. Il s'ensuit que l'encours élémentaire $xe_{A'}(t)$ est toujours égal à *un* (ici, $ye_{A'} = 2$). Concernant l'élément *B'*, il suffit d'étudier les instants où le stock A_4 contient deux éléments *B'* : à ces instants, ni T_3 ni T_7 ne sont actives, donc $xe_{B'}(t)$ est égal à *deux unités* et il est inférieur à $ye_{B'}$ (ici, $ye_{B'} = 3$). L'encours global *Xe* est donc d'une unité. L'étude du suivi du MPS(3,2) non représenté conduit au même résultat.

Le cas du MPS(3,4) est similaire. Le stock en A_2 est au plus égal à *un* et à ces instants ni T_1 ni T_5 ne sont actives. De plus les tâches T_1 et T_5 ne travaillent jamais en même temps. Le stock en A_4 est au plus égal à *deux* et à ces instants ni T_3 ni T_7 ne sont actives. Il s'ensuit que, quel que soit l'instant *t*, $xe_{A'}(t)$ et $xe_{B'}(t)$ valent au plus respectivement *un* et *deux*, valeurs qui restent inférieures aux valeurs *trois* et *quatre*. L'étude du suivi du MPS(4,3) non représenté conduit au même résultat.

En conclusion, quel que soit le cas, l'encours global *Xe* est d'une *unité*.

➢ *Régime cyclique et régime transitoire de départ.*

Dans tous les cas, le régime cyclique est conforme au régime cyclique[1] donné par le simulateur et dans tous les cas, la ressource goulet, qui dépend du cas de MPS étudié, est saturée. Nous notons également que les MPS(2,3) et MPS(3,4) nécessitent une pièce initiale *B'* dans le stock A_4. Celle–ci peut y être apportée par une première séquence T_3-T_4, ce qui a pour conséquence d'allonger ce transitoire de *11 unités de temps*. De même, les MPS(3,2) et MPS(4,3) nécessitent une pièce initiale *A'* dans le stock A_2. Une première séquence T_1-T_2 allongerait le transitoire de 12 *unités de temps*.

[1] Les régimes transitoires ne pourraient être comparés qu'aux régimes cycliques des diagrammes de GANTT exploitables.

2.3.6. Résultats.

Les temps de cycle de l'application pour les différents MPS sont réunis sur le *Tableau 6*. Les temps des MPS (2, 2), (3, 3) et (4, 4) sont obtenus en multipliant par *2*, *3* et *4* le temps associé au MPS (1, 1). Les MPS(2,4) et MPS(4,2) sont déduits des MPS(1,2) et MPS(2 ,1).

0rdo.	x = 1	x = 2	x = 3	x = 4
y = 1	14	24	36	48
y = 2	22	28	36	48
y = 3	33	35	42	49
y = 4	44	44	49	56

Tableau 6 : Temps de cycle des solutions ordonnancées.

En conclusion, et en dehors de toute modification technologique, l'application de la méthode conduit à un temps de cycle optimal puisque, dans tous les cas de MPS, la ressource goulet est saturée.

3. Bilan.

3.1. Comparatif.

Nous avons synthétisé dans le *Tableau 7* les temps de cycles des différents MPS établis pour les trois types de modèles (graphe connexe, graphes multiples et solution ordonnancée).

MPS		Ressource	Temps de cycle			Gains relatifs		
x	y	Goulet	Ordo	Connexe	Multiple	M/C	O/C	O/M
1	1	R3	14	23	14	39%	39%	0%
1	2	R2	22	35	25	29%	37%	12%
1	3	R2	33	47	34	28%	30%	3%
1	4	R2	44	60	47	22%	27%	6%
2	1	R1	24	30	26	13%	20%	8%
2	2	R3	28	37	28	24%	24%	0%
2	3	R3	35	49	48	2%	29%	27%
2	4	R2	44	61	50	18%	28%	12%
3	1	R1	36	43	38	12%	16%	5%
3	2	R1	36	44	49	-11%	18%	27%
3	3	R3	42	51	42	18%	18%	0%
3	4	R3	49	63	71	-13%	22%	31%
4	1	R1	48	56	50	11%	14%	4%
4	2	R1	48	57	61	-7%	16%	21%
4	3	R3	49	60	72	-20%	18%	32%
4	4	R3	56	64	56	13%	13%	0%

M/C	Gain relatif de la solution à graphes multiples sur la solution à graphe connexe.
O/C	Gain relatif de la solution ordonnancée sur la solution à graphe connexe.
O/M	Gain relatif de la solution ordonnancé sur la solutions à graphes multiples.

Tableau 7 : Synthèse des temps de cycle relatifs aux trois modèles de commande.

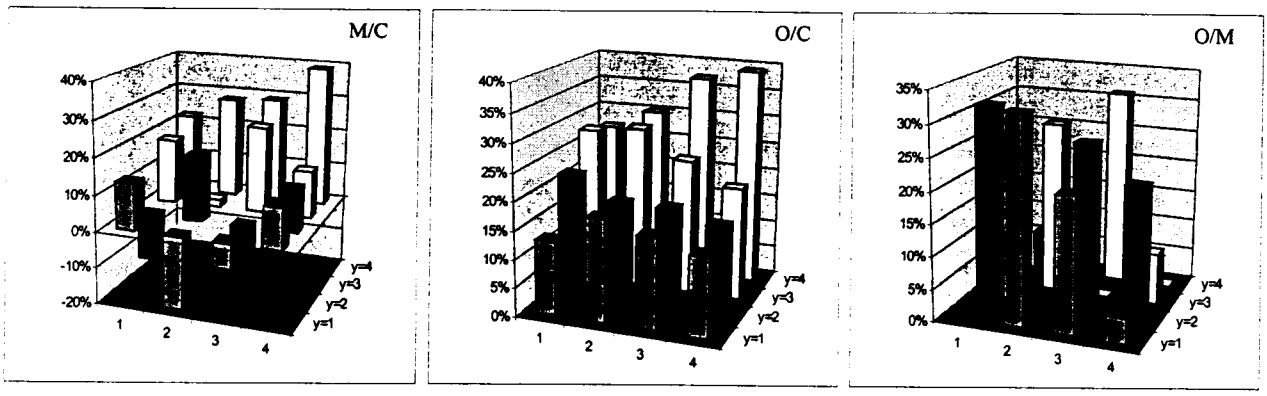


Figure 33 : Comparatifs deux à deux des trois solutions..

La *Figure 33* illustre qualitativement le tableau de synthèse précédent.

➢ *Comparaison « Multiple/Connexe ».*

Le premier constat porte sur le comparatif Multiple/Connexe, c'est-à-dire sur la comparaison entre la solution à graphes de commande dédié à chaque MPS (Cf. §*2.2)* et la solution '*universelle*' utilisant deux compteurs (Cf. § *2.1*). La première solution présente une meilleure productivité, en particulier pour les '*petites*' valeurs des variables x et y (ainsi que pour les valeurs telles que x=y). L'intérêt de graphes dédiés s'estompe dès que l'emploi d'un compteur s'avère nécessaire et cela est heureux puisque dans ce cas, le nombre de graphes dédiés augmente. La solution connexe devient même plus productive que la solution à graphes multiples pour les cas MPS(3,2), MPS(3,4), MPS(4,2) et MPS(4,3).

➢ *Comparaison « Ordonnancé/Connexe ».*

Le second comparatif Ordonnancé/Connexe, c'est-à-dire entre la solution basée sur notre méthode et la solution à compteurs, va dans le même sens. L'intérêt de la commande ordonnancée diminue également lorsque les valeurs de x et y s'accroissent, mais la productivité de la solution ordonnancée reste toujours supérieure à celle de la solution connexe.

➢ *Comparaison « Ordonnancé/Multiple ».*

Les productivités des solutions telles que x=y sont équivalentes et la solution 'multiple' est d'ailleurs plus complète. Dans les autres cas, la productivité de la solution ordonnancée est toujours meilleure, ce qui est normal puisqu'en dehors de la considération d'encours, elle est optimale.

3.2. Intérêts et limites de notre méthode appliquée au cas d'étude.

➢ *Intérêt de notre méthode.*

Nous distinguons ici deux concepts : la qualité et la multitude des graphes de commande.

Quel que soit le cas de MPS étudié, la qualité de la commande issue de notre méthode apporte une productivité optimale : les gains par rapport à la solution multiple sont appréciables (jusqu'à 30% pour le MPS(4,3)).

QUALITE de la COMMANDE ⇒ IMPACT sur la PRODUCTIVITE.

Dans une certaine limite des paramètres du MPS, la solution multiple présente également une meilleure productivité. Les gains de la solution multiple par rapport à la solution connexe varient de 10 à 40% (nous ne considérons que les cas où ce gain est positif). La multitude des graphes est donc une réponse à une application évolutive et elle améliore sa flexibilité.

MULTITUDE de la COMMANDE ⇒ IMPACT sur la FLEXIBILITE.

En définitive, la superposition de ces deux concepts qui conduisent à la définition de commandes optimisées et dédiées rendent une application à la fois productive et flexible.

➢ *Limitations de l'ordonnancement cyclique.*

Les limitations sont de deux natures : l'augmentation de la plage de variation des paramètres x et y provoque une augmentation au carré du nombre des graphes de commande et un accroissement de la taille de ces graphes. La commande composée de ces nombreux graphes devient rapidement impossible, d'autant plus que les gains de productivité apportés par ces graphes deviennent de plus en plus faible.

La seconde limitation est liée à l'encours. Il n'est pas dit qu'en augmentant la valeur des paramètres x et y, l'encours reste unitaire. L'augmentation de l'encours provoquerait donc une augmentation de la taille des graphes de commande.

Sur ce type d'étude, la limite de notre méthode devrait être fixée par des considérations économiques. Quel est le coût du développement et de la mise en œuvre de 2*(n+1) graphes de commande supplémentaires ? Quel est le taux d'utilisation des cas gérés par ces nouveaux graphes et quels sont les gains financiers escomptés ? La date du retour sur investissement est-elle admissible ?

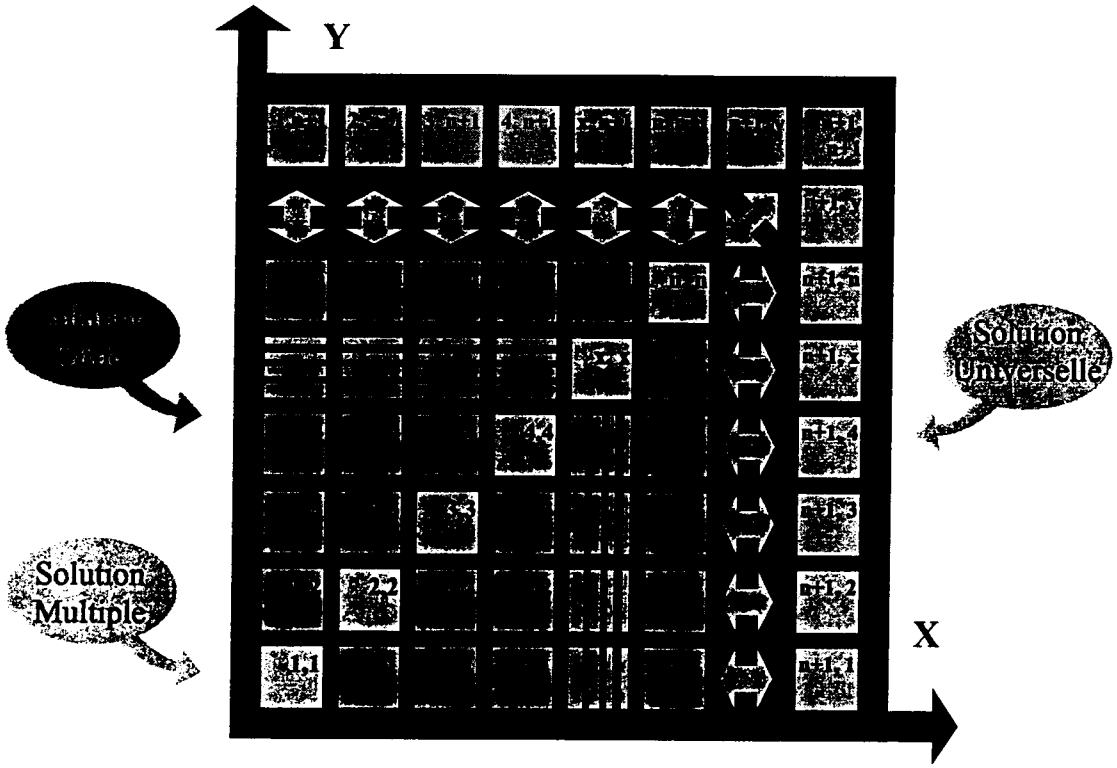


Figure 34 : Limite et complémentarité des solutions

Nous représentons sur la *Figure 34* les domaines respectifs d'application de chacune des méthodes étudiées. La solution dédiée au cas MPS(1,1) et obtenue traditionnellement, occupe la diagonale du plan (*x, y*). En dessous de la limite économique évoquée précédemment, les solutions ordonnancées sont les plus avantageuses. Au-delà de cette limite, le relais est pris par la solution universelle utilisant les compteurs.

4. Conclusions.

Bien que les critères d'évaluation de la productivité et de la flexibilité soient incomplets, nous avons souhaité montrer à travers ce cas d'étude élémentaire, l'avantage procuré par une commande dédiée et l'intérêt de notre méthode de conception.
Nous observons sur cet exemple, que sans être universelle, cette méthode trouve ici son champ d'application en complément des méthodes traditionnelles.

Dans le sens de notre évaluation, le système automatisé de production que nous venons d'étudier est donc à la fois productif et flexible.

Quatrième partie : Conclusions et Perspectives.

1. Conclusions.

Nous avons montré, à travers l'étude bibliographique de la première partie de ce mémoire, l'étendue des travaux de la communauté scientifique dans le domaine des systèmes automatisés de production : l'ensemble de ces travaux qui portent sur la spécification, la conception, les tests et l'intégration, la vérification et la validation, forme le cycle de vie de ces systèmes. Dans l'esprit des activités du génie logiciel, nous nous sommes tournés vers le cycle de vie de type SPIRALE, bornant ainsi notre étude à ses premières boucles. Le point de départ de notre action est une spécification générale de l'application. Nous en concevons globalement la commande pour vérifier ensuite ses propriétés de célérité et de vivacité.
Par rapport au cycle de vie dit en « *V* », l'approche SPIRALE présente à notre avis, un double intérêt : elle limite les écueils dus à l'analyse descendante (constat éventuel d'impossibilités après une phase plus ou moins longue de développement) et elle favorise la ré-utilisation propre à une démarche ascendante : dans cette optique, une étude peut prématurément toucher à sa fin, soit parce que la vérification partielle nous montre que les objectifs finaux ne peuvent être atteints, soit parce qu'elle a mis à jour des solutions existantes que nous pouvons ré-utiliser.

Dans la deuxième partie de ce mémoire, les outils d'analyse et de vérification ainsi que la méthode de conception que nous avons développés, nous ont conduit à la définition d'un formalisme basé sur les graphes, noté GMT (*Graphe Marqué et Temporisé*). Ce formalisme participe à la constitution d'une plate-forme multi-modèles qui admet le formalisme GRAFCET comme pivot central et qui autorise ainsi le ré-utilisation de tous les travaux de la communauté s'y rapportant (travaux relatifs aux développements du modèle Grafcet).
Les outils d'analyse basés sur la théorie des graphes ont été adaptés pour être exploités en amont et en aval de la phase de conception ou de re-conception d'une application.
En amont, la mise en évidence des composantes du graphe, par les outils traditionnels ou par la méthode DSM, donne au concepteur une vision globale favorisant la phase de conception préliminaire et distribuée de l'application étudiée. Cette vue macroscopique guide le concepteur dans les opérations de partition qui peuvent par exemple se faire autour d'un point d'articulation (ou autour d'un isthme en connaissance des risques encourus).
En aval, l'évaluation des performances temporelles et la vérification des propriétés '*graphe vivant*' et '*graphe sauf*' permettent d'apprécier la conformité des performances du modèle GMT à celles décrites dans le cahier des charges. Par le biais de la passerelle GMT → Grafcet, le concepteur accède alors au modèle Grafcet de l'application.

La deuxième partie de ce mémoire porte aussi sur la méthode de conception que nous développons autour des techniques de l'ordonnancement cyclique. En réponse au caractère général des tâches assurées par un SAP, cette méthode est capable d'ordonnancer des gammes opératoires graphiquement non linéaires, pouvant comporter des opérations d'assemblage et/ou de désassemblage. Les gammes associées à une production cyclique sont transformées, puis dupliquées afin de constituer un plan de production. La méthode de conception est basée sur la simulation répétée de l'exécution de ce plan et la recherche du meilleur ordonnancement s'effectue en imitant le comportement des colonies de fourmis. Le résultat de l'ordonnancement est donné sous la forme d'un diagramme de GANTT. Ce diagramme est optimal au sens du temps de cycle, puisqu'il conduit à la saturation de la ressource critique. Enfin, après avoir donné une définition de l'encours propre à des productions composées et multiples, nous proposons deux techniques d'établissement du graphe de commande de l'application étudiée. L'utilisation d'un critère d'optimisation basé uniquement sur le temps total d'exécution (makespan), ne nous a pas permis de contrôler l'encours. Aussi, nous préconisons dans le paragraphe suivant (perspectives), de redéfinir une fonction 'coût' qui tiendrait également compte de cet encours.

Les résultats obtenus sont encourageants. Les deux exemples traités dans la deuxième partie montrent l'aptitude de notre méthode à concevoir des graphes de commande qui saturent la ressource critique, malgré la variété des gammes opératoires initiales. L'étude effectuée dans la troisième partie de ce mémoire concerne une application à MPS variable. Elle illustre pour sa part, l'intérêt de dédier une commande spécifique à chaque MPS. Ces deux groupes d'études montrent respectivement l'impact positif de notre méthode sur la productivité et sur la flexibilité.

Nous avons évoqué la difficulté présentée par la mesure[1] des performances de productivité et de flexibilité d'un SAP. Si nous ne pouvons pas quantifier exactement l'apport de notre méthode sur ces deux propriétés, l'extrapolation des résultats obtenus nous permet cependant de penser qu'elle contribue à leurs améliorations et en ce sens, nous répondons à la problématique de départ.

[1] Louis PASTEUR nous enseigne qu'une science a la maturité de ses outils de mesure. Devons-nous en déduire la jeunesse de ce secteur d'activité ?

2. Perspectives.

Le travail que nous venons d'effectuer ouvre plusieurs perspectives. En premier lieu, nous aimerions consolider nos 'acquis' sur l'algorithme des fourmis (robustesse, domaine d'efficacité), puis étendre l'utilisation de la méthode d'ordonnancement cyclique sur laquelle elle s'appuie vers d'autres classes de problème (HSP, ordo non cyclique...) : nous aimerions aussi appliquer notre méthode à la conception de graphes de plus haut niveau (architecture, partitionnement, exécution parallèle...) et l'associer éventuellement à la notion d'objet.

2.1. Sur l'algorithme basé sur les fourmis.

La première perspective de développement que nous envisageons concerne la fonction coût de cet algorithme. Actuellement, cette fonction est uniquement basée sur le temps total d'exécution. Or, nous avons proposé au paragraphe *4.4.3.* de la deuxième partie de ce mémoire, une nouvelle définition et par conséquent une nouvelle méthode d'évaluation de l'encours, définition et méthode que nous avons adaptées à notre classe de problèmes (gammes graphiquement non-linéaires). En intégrant dans l'algorithme le calcul de l'encours *Xe* pour chaque ordonnancement de chaque fourmi, cet encours qui pour le moment n'est qu'une conséquence de l'ordonnancement, peut devenir acteur en participant à l'élaboration de la fonction coût. Dans cette optique, nous définirons deux coefficients de pondération et d'adaptation λ_1 et μ_1 tels que :

$$Fct\_cout = \lambda_1.TU + (1-\lambda_1).\mu_1.Xe$$

où μ_1 homogénéise la taille de l'encours *Xe* vis-à-vis du temps total d'exécution *TU* et où λ_1 pondère l'importance du makespan devant l'encours homogénéisé. Il s'agit ici, d'accroître l'efficacité de notre méthode.

La deuxième perspective est propre à l'emploi d'une méthode heuristique. A travers des campagnes de mesures exécutées sur des séries d'exemples, nous souhaiterions apprécier pour chacun d'eux, l'impact des paramètres de réglage sur l'algorithme : nous chercherons l'influence des coefficients de bruitage, d'héritage et de non affectation tâche/ressource sur la rapidité de convergence de l'algorithme, mais aussi sur sa capacité à explorer finement l'espace des solutions. L'objectif de ce travail sera donc de confirmer la robustesse de l'algorithme et de maîtriser ainsi son utilisation.

La troisième perspective est davantage comparative et fait appel aux travaux actuels de la communauté. Nous souhaiterions disposer d'une série de benchmarks à travers lesquels nous pourrions comparer les résultats fournis par l'algorithme des fourmis, par les algorithmes génétiques, ou par toute autre heuristique (Tabou, recuit simulé...). Ce travail devrait permettre d'ébaucher les limites du champ d'application de chacune de ces heuristiques. Il s'agira ici, d'estimer l'efficacité.

2.2. Sur l'application de l'ordonnancement à d'autres classes de problèmes.

La seconde famille de perspectives porte sur l'extension de notre méthode d'ordonnancement à d'autres classes de problèmes. Nous envisageons d'appliquer notre méthode aux problèmes de galvanoplastie appelés aussi HSP (*Hoist Scheduling Problem*). En première approche, une production flexible de type HSP comprend une ressource de transport et cette production est caractérisée par le fait que la durée de chaque tâche est définie sur un intervalle [min, max]. Pour gérer ce nouveau type de contraintes (durée de tâche inférieure à un maximum), nous projetons d'associer à chaque tâche de traitement une seconde tâche, séquentiellement indissociable de la première. Dans cette hypothèse, la durée de la tâche de traitement serait fixée par la borne minimale de l'intervalle et la durée de la seconde tâche par la longueur de cet intervalle : la fin réelle d'une tâche de traitement devra alors s'effectuer avant la fin de la tâche associée. Le bon ordonnancement d'un problème de ce type présenterait le temps total d'exécution le plus court, tout en interdisant qu'une tâche se termine après la fin de sa tâche associée. Il se dessine a priori deux possibilités : soit ces interdictions seront traduites sous forme de contraintes que le simulateur devra respecter (une tâche associée active formulera une demande prioritaire au système de transport), soit on éliminera après coup les fractions d'ordonnancement qui auront occasionné ces situations interdites (marquage particulier de la matrice des phéromones).

Nous envisageons également, en adaptant la fonction coût, de traiter des problèmes non cycliques où nous chercherons par exemple à minimiser la somme des retards pondérés. Comme pour la prise en compte de l'encours dans la fonction coût (Cf. paragraphe précédent *2.1*), cette somme notée *SRp*, serait calculée pour chaque ordonnancement de chaque fourmi, puis injectée dans la fonction coût telle que :

$$Fct\_cout = \lambda_2.TU + (1-\lambda_2).\mu_2.SRp$$

où μ_2 homogénéise la somme des retards pondérés *SRp* vis-à-vis du temps total d'exécution *TU* et où λ_2 pondère l'importance du makespan devant la somme homogénéisée des retards pondérés.

2.3. Sur les outils d'analyse et la méthode de conception.

La méthode de conception et les outils d'analyse développés présentent un caractère de complémentarité que nous souhaiterions exploiter dans la construction d'architectures de plus haut niveau. A l'image de la démarche 'technologie de groupe', la méthode DSM (Cf. §*3.4*, deuxième partie du mémoire) devrait nous permettre de regrouper adroitement les tâches interdépendantes d'une application tout en minimisant les interactions entre ces différents groupements (*clustering*). Nous pourrions alors concevoir à l'aide de notre méthode, un graphe de commande de haut niveau chargé de piloter ces groupements, considérés alors comme des super-tâches de niveau '*zéro*'. Chaque super-tâche de niveau '*zéro*' pourrait à son tour faire l'objet d'un nouveau découpage DSM et donner lieu à un nouveau graphe de commande pilotant des super-tâches de niveau '*un*'...etc. L'architecture résultante de ce processus itératif serait sans doute très proche de l'architecture '*Transparent Factory*' évoquée au §*3.2.1.* de la première partie de ce mémoire. L'orientation 'Objet' de ce type d'application est alors sous-jacente.

Enfin, la finalité des propositions exposées dans ce mémoire étant d'améliorer les performances des outils de production, nous souhaiterions valider notre travail dans le contexte industriel. L'activité de conception ou de re-conception d'une application réelle ne doit pas manquer d'intérêt.

Lexique.

A	Matrice d'antériorité,
AE	Matrice d'antériorité expansée,
CF	Temps de cycle d'une macrogamme,
CT	Temps de cycle de la ressource goulet,
I	Matrice des liens antérieurs transition/étape d'un grafcet,
E	Matrices des entrées,
m	Nombre d'arcs d'un graphe,
n	Nombre de sommets ou de nœuds d'un graphe,
ne	Nombre de gammes initiales d'une macrogamme,
nbt	Nombre de tâches,
nbr	Nombre de ressources,
O	Matrice des liens postérieurs transition/étape d'un grafcet,
q	Nombre d'étapes d'un grafcet,
r	Nombre de transitions d'un grafcet,
S	Matrice des sorties,
TU	Durée de l'ordonnancement (makespan),
X	Nombre de duplication,
Xe	Encours Global,
$Xe(t)$	Encours instantané,
$x_{ei}(t)$	Encours élémentaires,
$x_{ai}(t)$	Encours atomique,
Xr	Nombre toléré de ressources simultanément actives,
λ_1	Pondération de l'importance du makespan par rapport à l'encours
μ_1	Coefficient d'adaptation de l'encours par rapport au makespan.
SRp	Somme des retards pondérés.
λ_2	Pondération de l'importance du makespan par rapport à la somme SRp.
μ_2	Coefficient d'adaptation de la somme SRp par rapport au makespan.

Annexe 1: Procédures spécifiques à l'analyse des graphes.

1. Présentation

Nous avons établi l'univocité entre un grafcet et son graphe associé lorsque le grafcet est soit purement un grafcet d'événements, soit purement un grafcet d'état. L'objectif des procédures ci-après est de déduire à travers l'analyse d'un graphe, les caractéristiques du grafcet associé. Elles permettent d'analyser l'architecture de ce grafcet et d'évaluer ses propriétés de vivacité et de célérité.

Après une description des structures de données utilisées par ces procédures, nous présentons successivement, la procédure permettant le passage du grafcet au graphe (dans le cas de graphe d'événements ou exclusivement de graphe d'état), la procédure inverse, les procédures d'analyse de l'architecture (connexité, circuits, chemins, isthmes, points d'articulation et partition) et finalement, les procédures d'évaluation des temps de cycle et de la vivacité.

Ces procédures sont écrites sous DELPHI et représentent un fichier exécutable de 3 Mo.

2. Organisation.

2.1. Les structures de données.

La matrice d'antériorité *A* constitue la donnée principale d'entrée. A partir de cette matrice, la procédure « DSM » ré-organise cette matrice et les procédures « Graphe-Grafcet » et « Grafcet-Graphe » donnent l'équivalent du graphe dans le formalisme grafcet. Les procédures « Génère_ES », « Latine » et « Sigma » construisent des matrices intermédiaires, respectivement *E, S, X* et *SIG*, matrices exploitées par les procédures dédiées à l'analyse de l'architecture. Un second groupe de données caractérisant les temps des tâches et leur marquage initial, permet, après l'étude des circuits et des chemins, d'évaluer la célérité et la vivacité du graphe de l'application.

La variable globale *dimgra* représente le nombre de nœuds du graphe traité. *n* et *ncirmax* sont deux constantes globales qui fixent la taille maxi des graphes traités et le nombre maximal de circuits. La machine sur laquelle a été implémenté notre logiciel (Pentium III, 700MHz, 128 Mo de RAM) accepte le traitement de graphe de densité moyenne ($\approx 20\%$), tel que $n=50$ et $ncirmax=800$ sans pratiquer d'échange mémoire avec le disque dur.

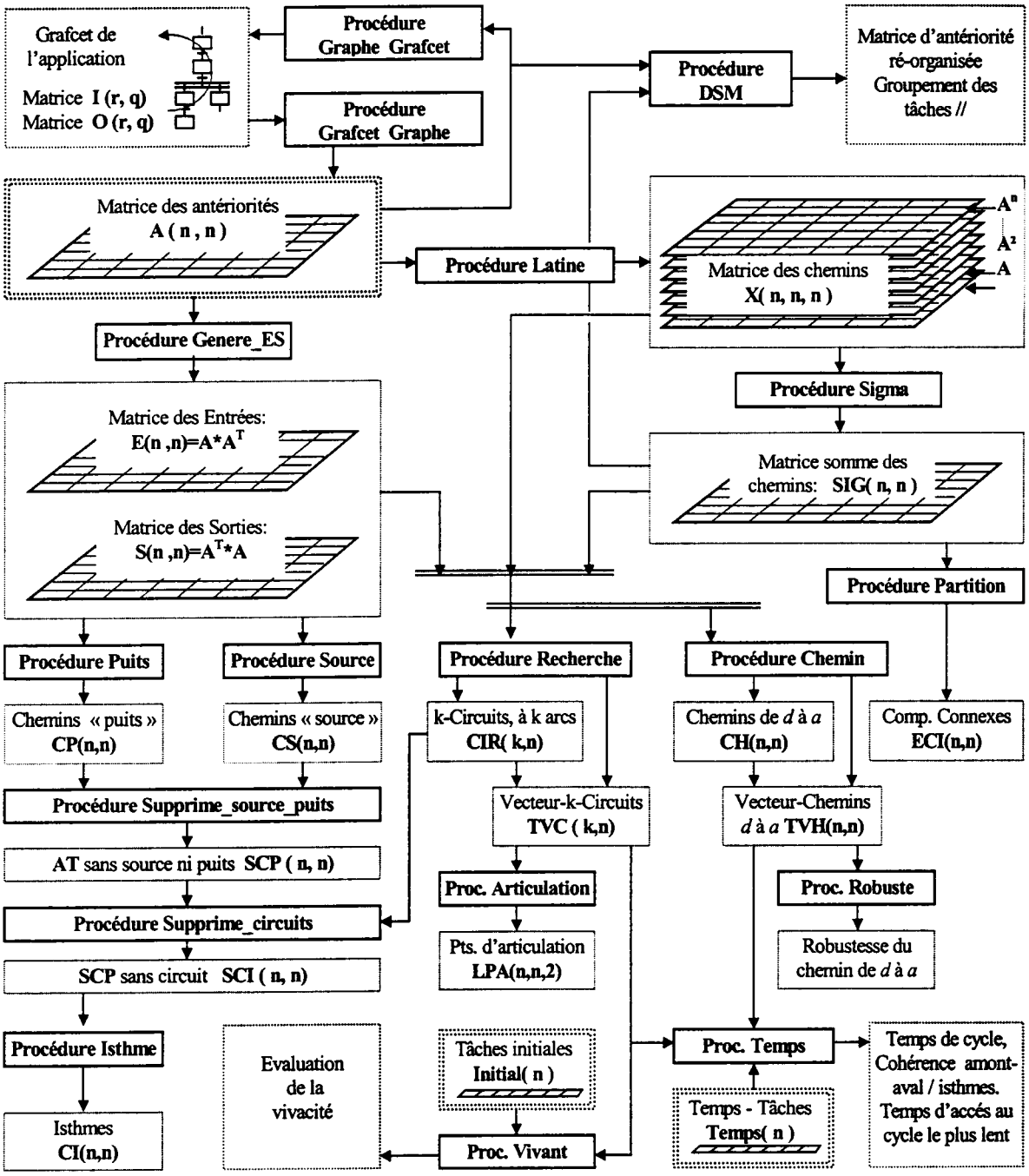


Figure 1 : Organisation logicielle autour de la structure de données.

2.2. *Les Procédures préliminaires.*

2.2.1. Les procédures « Latine » et « Sigma ».

La sous-matrice *X[1, n, n]* est la copie conforme de la matrice des antériorités *A[n, n]* exceptés les termes diagonaux *X[1, i, i]* qui prennent la valeur *0*, ce qui revient à éliminer les boucles. La sous-matrice *X[2, n, n]* est le produit de *X[1, n, n]* par elle-même. Les sous-matrices X[k, n, n] pour $k>2$ sont obtenus par le produit récurrent *X[k-1, n, n] * X[1, n, n]*.

Les termes *X(k, i, j)* de la matrice *X* représente le nombre de chemins orientés à *k* arcs allant du sommet *j* au sommet *i* . Si $j = i$ alors le sommet *i* est impliqué *X[k, i, i]* fois dans un circuit à *k* arcs. En effet, si *(A[u , j] <> 0 et u<>j)* alors il existe un arc allant du sommet *j* vers le sommet *u* et *X[1, u , j]* est différent de *zéro*. Si *X[k-1, i, u]* $= \alpha_u$ ce qui est équivalent à dire qu'il existe α_u chemins à *k-1* arcs allant du sommet *u* au sommet *i* alors le terme *X [k , i , j]* $= \sum_u X[k-1, i, u] * X[1, u , j]$, soit, $X[k , i , j] = \sum_u \alpha_u * A[u , j]$ représente le nombre de chemin à *k* arcs allant du sommet *j* au sommet *i*.

La procédure « Sigma » réalise la somme sur *k* des matrices *X[k ,n ,n] et fournit la matrice SIG[n, n]*. Les termes $SIG[i , j] = \sum X[k , i , j]$ représente les chemins orientés allant du sommet *j* au sommet *i*, quelque soit le nombre d'arcs qui les composent.

2.2.2. La procédure « Génére_ES ».

Cette procédure fournit à partir de la matrice *A*, les matrices *E* et *S*, respectivement matrice des Entrées et matrice des Sorties. *E* est définie par le produit $A*A^T$ et *S* par le produit A^T*A.

E[i,j] représente le nombre d'antériorités communes aux sommets *i* et *j*, Si $i = j$, *E[i, j]* est égal au degré intérieur $d_G^-(i)$ du nœud *i*, c'est à dire au nombre d'arcs entrant sur ce nœud.

S[i,j] représente le nombre de postériorité communes aux sommets *i* et *j*, Si $i = j$, *S[i, j]* est égal au degré extérieur $d_G^+(i)$ du nœud *i*, c'est à dire au nombre d'arcs sortant de ce nœud.

2.2.3. Réduction de l'espace de recherche.

Outre ces structures de données communes à la quasi-totalité des procédures, nous avons tenté de réduire spécifiquement à chaque type de recherche, l'espace sur lequel travaille chacune des procédures.

3. Les procédures « passerelle » et DSM.

Ces procédures permettent de passer des matrices d'entrées et de sorties *I* & *O* du grafcet à la matrice d'antériorité *A* du graphe, et réciproquement. Dans le cas de grafcets d'événements, purement non disjonctifs, nous utilisons une matrice intermédiaire, notée *AE*. Cette matrice est dite expansée, car en plus des nœuds associés aux tâches réelles, elle contient également les nœuds associés aux étapes de synchronisation.

3.1. La passerelle Grafcet → Graphe.

3.1.1. Cas d'un grafcet d'événements (purement non-disjonctif).

Nous rappelons que dans ce type de grafcet chaque étape n'est précédée et/ou suivie que par une seule transition, mais une transition peut-être précédée et/ou suivie par plusieurs étapes: Il n'y a ni choix ni conflit, mais éventuellement des parallélismes.

Début { grafcet non disjonctif → graphe non disjonctif }
  $AE := O^T * I$
  Pour *i* de *1* à *n* **faire**
    Pour *j* de *1* à *n* **faire, si** $ae_{ij} = 1$ **alors** $a_{ij} := 1$ **sinon** $a_{ij} := 0$;
  Pour *i* de *1* à *n* **faire**
    Pour *j* de $n+1$ à *q* **faire si** $ae_{ij} = 1$
      alors pour *k* de 1 à *q* **faire**
        si $ae_{jk} = 1$
        alors $a_{ik} := 1$
        sinon *rien*
      sinon *rien*
Fin

3.1.2. Cas d'un Grafcet d'état (purement disjonctif).

Dans ce cas, la matrice expansée est égale à la matrice d'antériorité. Il s'en suit que cette dernière est directement obtenue par le produit de la transposée de la matrice des sorties par la matrice des entrées du grafcet.

Début { grafcet disjonctif → graphe disjonctif }
  $A := O^T * I$
Fin

3.2. La passerelle Graphe → Grafcet.

Il s'agit ici du passage inverse, de la matrice des antériorités A du graphe aux matrices I & O du grafcet. Dans le cas d'une interprétation du graphe en graphe d'événement, nous construisons dans un premier temps la matrice expansée AE, puis, à partir de AE, les matrices I et O.

3.2.1. Cas d'un graphe interprété comme un graphe. d'événements.

Début { graphe non disjonctif → grafcet non disjonctif }

{ Initialisation de AE, I et O }
$E := A*A^T$; *Calcul de la trace(E) et de* u_e

$q :=trace(E) + n-u_e$; $r := 2.n - u_e$;
$AE[q, q] :=[0]$; $I[r, q] :=[0]$; $O[r, q] :=[0]$

Pour i de 1 à n **faire**
Pour j de 1 à n **faire, si** $a_{ij} =1$ **alors** $ae_{ij}:=1$ **sinon** $ae_{ij}:=0$;

{ Construction de AE }
$qc :=n$;
pour i de 1 à n **faire si** $e_{ii}>1$ **alors**
pour j de 1 à n **faire**
si $a_{ij}=1$
alors début
$qc :=qc+1$;
$ae_{iqc} :=1$; $ae_{qcj} :=1$; $ae_{ij} :=0$
fin
sinon *rien*

{ Construction de I et O }
$rc :=n$;
pour i de 1 à n **faire**
début
$i_{ii} :=1$;
si $eii>1$
alors début
$rc :=rc+1$; $o_{rci} :=1$;
pour j de 1 à q **faire** $i_{rcj} :=ae_{ij}$
fin
sinon *rien*
pour j de 1 à q **faire** $o_{ij} :=ae_{ji}$
fin
Fin

3.2.2. Cas d'un graphe interprété comme un graphe d'état.

```
Début { graphe disjonctif → grafcet disjonctif }
   rc :=0 ; I[r, q] :=[0] ; O[r, q] :=[0]
   Pour j de 1 à n faire
      Pour i de 1 à n faire
               si aij =1
               alors
                  début
                     rc :=rc+1;
                     ircj :=1;
                     orci :=1
                  fin
               sinon rien
Fin
```

3.3. La procédure DSM.

La finalité de cette procédure est de paralléliser un ensemble de tâches en regroupant par niveaux des tâches simultanément exécutables vis à vis de leurs propres contraintes de précédences. Cette procédure ré-organise la matrice d'antériorité *A* sous forme triangulaire. Les conditions de triangularisation sont décrites dans [HARARY F., 62] et la méthode s'appuie sur deux techniques dues à [TANG D. & al., 00] et [DONALD V., 81 #140]. Dans l'algorithme proposé, la partition par niveau est empruntée à TANG, mais contrairement à ce dernier, la ré-organisation des tâches au sein de chaque niveau, est effectuée selon la technique de DONALD.

Nous utilisons en données d'entrées, la matrice *A* et la matrice « somme des chemins » *SIG*. *PB* n'est autre que la matrice *SIG* dans laquelle les termes non nuls sont remplacés par la valeur 1. Si PB_{ij} est différents de zéro, alors il existe un chemin de *j* vers *i*. Si PB_{ji} est différents de zéro, alors il existe un chemin de *i* vers *j*. Le terme PP_{ij} de la matrice *PP* est égale au produit de PB_{ij} par PB_{ji}. Si PP_{ij} est différents de *zéro* alors les nœuds *i* et *j* sont fortement connectés. Les lignes identiques de la matrice *PP* sont ensuite fusionnées et les colonnes associées à ces lignes sont supprimées. Nous obtenons alors la matrice réduite *PR*, de dimension *(m, m)* où *m* représente le nombre de sous ensembles ou listes au sein desquels les nœuds sont fortement connectés.

Le produit de cette matrice PR, par un vecteur « masque » permet à la façon d'un peigne de sélectionner à chaque passage les ensembles de tâches simultanément exécutables. Toutes les composantes de ce vecteur sont initialisées à 1 ; La $i^{ème}$ composante de ce vecteur est ensuite annulée après chaque passage, lorsque si le $i^{ème}$ élément vient d'être sélectionné par le peigne.

```
Début { Procédure DSM, Design Structure Matrix }
    { Booléennisation de la matrice SIG }
    Pour i de 1 à dimgra faire
        Pour j de 1 à dimgra faire
            Si SIG[i,j] =0 alors PB[i,j] :=0 sinon PB[i,j] :=1 ;
    { Construction de la matrice PP, PP = PB ∩ PB^T }
    Pour i de 1 à dimgra faire
        Pour j de 1 à dimgra faire PP[i,j] : = PB[i,j] * PB[j,i] ;

    { PR← Réduction de la matrice PP, suppression des lignes identiques }
    ∀ [i,j] ∈ [1..n]² faire LF[i,j] :=0 ; PR[i,j]:=0;
    LF[1,1] :=1; LF[1,n] :=1; { compteur des lignes fusionnées }
    Pour i de 1 à dimgra faire PR[1,i] : = PP[1,i] ;
    Pour i de 1 à dimgra faire | Si ∀ [ j, x] ∈ [1..n]², PR[i,j] : = PP[x,j]
                               | Alors LF[x, n] :=LF[x, n]+1  { Fusion de i et x        }
                               |       LF[x ,LF[x ,n]] :=i ;  { Ajouter i à la liste x  }
                               | Sinon m :=m+1                { Ajouter i à la liste PR }
                               |       Pour j de 1 à dimgra faire PR[m, j] :=PP[i, j] ;

    {Fusion des colonnes associées aux lignes supprimées }
    Pour i de 1 à m faire
        Pour j de 2 à LF[i,n] faire
            Pour k de 1 à m faire
                Pour l de LF[i,j] à dimgra-1 faire | PR[k,l] := PR[k,l+1] ;
                                                   | PR[k,l+1] :=0 ;
    { Définition des niveaux }
    LR :=0 ; ∀ [i,j] ∈ [1..n]² faire LEV[i,j] :=0 ; { init nombre de niveau }
    Pour i de 1 à m faire VEC[i] :=1 ;
    Vecteur_non_nul := vrai ;
    Tant Que Vecteur_non_nul faire
        Pour i de 1 à m faire VP[i] :=0 ;
                              Pour j de 1 à m faire VP[i] :=VP[i]+PR[i,j]*VEC[j] ;
        LR :=LR+1 ; NLR :=0 ;
        Pour i de 1 à m faire | Si ( VP[i]=0 ou VP[i]=1 )
                              | Alors VEC[i]:=0;
                              | Si VP[i]=1
                              | Alors NLR :=NLR +1 ;LEV[LR,NLR] :=i ;
        LEV[LR,n]:=NLR;
        Si ∀ i ∈ [1..n], VEC[i]=0 Alors Vecteur_non_nul :=faux ;
    Fin Tant Que
Fin
```

4. Les procédures d'analyse du graphe.

4.1. La procédure « partition ».

La procédure « partition » travail uniquement sur la matrice « somme des chemins » *SIG*. Elle retourne la matrice *ECI*. *ECI[i , j]* représente la $j^{ème}$ coordonnée de la $i^{ème}$ composante connexe, i.e. l'appartenance du sommet *j* à la composante connexe *i*. : si $j \in i^{ème}$ composante connexe alors *ECI[i , j] = 1*, sinon *ECI[i , j] = 0*. La base de l'algorithme est la suivante :

Si *(SIG[i , j]+SIG[j , i] = 0)* **Alors** | *i* et j ∈ à *2* composantes connexes *disjointes*
| **si** *i ou j* ∉ à une composante connexe
| **alors** on crée une nouvelle composante
Sinon | *i et j* ∈ à la *même* composante connexe
| **si** *i ou j* ∈ à une composante connexe
| **alors** on ajoute *j ou i* ∈ à cette composante
| **si** *i et j* ∉ à une composante connexe
| **alors** on crée une nouvelle composante
| **si** *i & j* ∈ *à* deux composantes disjointes
| **alors** on fusionne les deux composantes.

Un balayage supérieur droit de la matrice *SIG[n , n]* permet l'examen complet du graphe.

4.2. Les procédures « source » et « puits ».

Les procédures « source » et « puits » utilisent les matrices *A, E* et *S*. Le principe de la recherche d'un chemin source est le suivant :

Si *E[i , i] = 0* **Alors** le sommet *i* est une source ;
Pour *k* de *1* à *dimgra* **faire**
Si *A[k, i]<>0* **Alors**
| *j := A[k, i]* ; { *k* est le suivant de *i* }
| **tant que** *(E[j , j] = 1 et S[j , j] = 1)* **faire**
| *j* ∈ au chemin *i*
| *j* ← suivant de *j*
| **fin tant que**

La condition *(E[j , j] = 1 et S[j , j] = 1)* assure que le nœud courant ne reçoit qu'un arc d'entrée et qu'il ne possède qu'un arc de sortie.

Le principe de la recherche d'un chemin puits est très similaire :

Si *S[i , i] = 0* **Alors** le sommet *i* est un puits ;
Pour *k* de *1* à *dimgra* **faire**
Si *A[i, k]<>0* **Alors**
j := A[i, k] ;{ *k* est le précédent de *i* }
tant que *(E[j , j] = 1 et S[j , j] = 1)* **faire**
j ∈ au chemin *i*
j ← précédent de *j*
fin tant que

On note que les chemins puits sont découverts en partant du puits, ils seront stockés en partant de la fin de la recherche.

4.3. La procédure « recherche » des circuits.

Cette procédure recherche les circuits du graphe à partir des matrices *A* et *X*. Elle fournit deux résultats, *CIR[n, ncirmax]* et *TVC[n, ncirmax]*, de type matrice de listes. *CIR[k, i]* représente le $i^{ème}$ circuit à *k* arcs et *TVC[k, i]* le vecteur associé à ce circuit. A titre d'exemple, supposons qu'un graphe à *7* nœuds contienne *3* circuits de rang *4*, alors le deuxième circuit *(i=2)* de rang *4 (k=4)* est consigné dans les matrices *CIR[n, ncirmax]* et *TVC[n, ncirmax]* par les éléments suivants : *CIR[4,2]= 1 / 2 / 4 / 6 / 0 / 0 / 0* et *TVC[4,2]= 1 / 1 / 0 / 1 / 0 / 1 / 0.* Le premier élément mémorise l'ordre et les composantes du circuit alors que le second ne mémorise que les composantes. La seconde forme s'avère plus commode lorsqu'on souhaite supprimer et/ou évaluer les circuits.

La recherche des circuits se fait en fonction du nombre d'arcs formant ces circuits. Ce fait constitue *la première originalité* de notre algorithme. Elle s'effectue en profondeur et elle utilise deux procédures « Monter » et « Descendre » qui gèrent à leur tour une structure basée sur la matrice *SU[-1..n,1..n]* d'entiers. Cette matrice ne contient que les liens statiques *SU[i,j] (avec i>0)* entre les sommets uniquement impliqués dans des *k*-circuits : Ces liens forment un sous-ensemble du graphe de départ, appelé *k*-arbre. La matrice *SU[-1..n,1..n]* est construite à chaque étape de recherche de *k*-circuits par la procédure interne « Créer_table ». Les procédures « Monter » et « Descendre » mettent à jour les données relatives au parcours du *k*-arbre selon la codification ci-après :

SU[i, -1] pointe le prédécesseur de *i*
SU[i, 0] pointe le successeur de *i* en cours dans la liste *SU[i,j] (avec i>0)*
SU[i, j] représente le $j^{ème}$ successeur de *i*, ou *0* s'il n'y a plus de successeur

La recherche de k-circuits dans le k-arbre s'arrête après examens de la diagonale de la matrice *X[k, n, n]*. Si cette diagonale ne contient pas au moins *k* éléments non nuls alors il n'existe pas ou il n'existe plus de *k*-circuit. A la découverte de chaque *k*-circuit, la diagonale de la matrice *X[k, n, n]* est mise à jour en retranchant à chaque élément diagonal sa participation à ce *k*-circuit. De ce fait, la recherche de *k*-circuits dans un k-arbre est de plus en plus aisée, puisqu'à chaque découverte d'un *k*-circuit, le *k*-arbre est de moins en moins dense. C'est *la seconde originalité* de notre algorithme. En outre, cette procédure peut avantageusement travailler sur une matrice *X[n , n , n]* elle-même construite à partir de la matrice *SCP* (matrice *A* dépourvue des chemins 'source' et 'puits'). Elle peut également travailler sur des matrices *A* restreintes à chaque composante connexe.

L'exploration complète et non redondante de cette procédure de recherche a été testée sur des graphes complets (∀ i, ∀ j , ∃ l'arc [i, j]) et nous avons pu constater que le nombre des *k*-circuits ainsi trouvés dans un graphe de *n* sommet vérifiait l'expression $\alpha_n^k = \frac{n!}{k.(n-k)!}$

Cette relation est vraie car elle vérifie la récurrence suivante: $\alpha_{n+1}^r = \alpha_n^r + (r-1).\alpha_n^{r-1}$. Cette récurrence est vraie pour *(n=2 et r=1 ou r=2),* pour *(n=3 et r=1, r=2 ou r=3)* et enfin pour *(n=4 et r=1, r=2, r=3 ou r=4).*

Supposons que $\alpha_n^r = \frac{n!}{r.(n-r)!}$ représente le nombre de *r*-circuits dans un graphe complet à *n* sommets. Nous ajoutons un sommet *u* et nous cherchons quel est le nombre de *r*-circuits contenus dans ce nouveau graphe. Nous établissons les liens pour que le graphe à *n+1* sommets soit à nouveau un graphe complet. Calculons $\delta = \beta + \gamma$ avec :

$\beta = \alpha_n^r$ est égal au nombre de *r*-circuits ne contenant pas *u*

$\gamma = (r-1).c$ représente le nombre de façon de placer le sommet *u* dans un circuit à *(r-1)* arcs du graphe ne contenant pas *u* multiplié par *c*, nombre de ces circuits, c'est à dire par α_n^{r-1}, d'où $\gamma = (r-1).\alpha_n^{r-1}$

$$\delta = \alpha_n^r + (r-1).\alpha_n^{r-1} = \frac{n!}{r.(n-r)!} + \frac{r-1}{r-1}.\frac{n!}{(n-(r-1))!} \text{ or,}$$

$(n-(r-1))! = (n+1-r)! = (n-r)!.(n-r+1)$ donc,

$$\delta = \frac{n!}{r.(n-r)!} + \frac{n!}{(n+1-r)!} = n!.\left[\frac{1}{r}.\frac{n-r+1}{(n+1-r)!} + \frac{1}{(n+1-r)!}\right] = \frac{n!}{(n+1-r)!}.\left[\frac{n-r+1+r}{r}\right]$$

$$\delta = \frac{(n+1)}{r}.\frac{n!}{(n+1-r)!} = \frac{(n+1)!}{r.(n+1-r)!} = \alpha_{n+1}^r \text{ et finalement, } \alpha_{n+1}^r = \alpha_n^r + (r-1).\alpha_n^{r-1} \quad \text{CQFD.}$$

A titre indicatif et il faut moins d'une seconde pour trouver les 2365 circuits d'un graphe complet de rang 7, mais il en faut 84 pour les 125673 circuits d'un graphe complet de rang 9!

La procédure « Recherche » fait appel à 9 booléens ou expressions relationnelles :

La fonction *DIAG(k)* retourne un booléen vrai si la diagonale de la matrice *X* contient au moins k éléments non nuls.
Le booléen *MOK* est vrai s'il existe encore des *k*-circuits.
Le booléen *DOK* est vrai s'il existe encore des *k*-circuits depuis le nœud de départ en cours.
Le booléen *CIOK* est vrai si la procédure descendre a trouvé un k-circuit.
Le booléen *DSOK* est vrai si on peut encore descendre sur une place non visitée.
Le booléen *ADJ* est vrai s'il existe encore une place adjacente non visitée.
Le booléen ENOK est vrai si le circuit trouvé n'est pas un doublon et s'il a été enregistré.

Nous donnons ici les algorithmes des procédures, «Monter», «Descendre» et «Recherche».

4.3.1. Les Procédures « Descendre » et « Monter ».

La procédure « Descendre » descend d'un nœud *d* à un nœud *a*. Elle met à jour les deux booléens *CIOK* et *DSOK*. La procédure « Monter » permet de monter d'un nœud *d* à un nœud *a*. Elle retourne le booléen *ADJ* à vrai s'il elle à trouvé un prédécesseur à *d* qui avait encore un successeur non visité, c'est à dire un adjacent.

```
Début { Procédure Descendre }
    Répéter
        Suivant := SU[ d, SU[ d, 0 ]] ;
        Si le suivant est le nœud de départ alors Bool1 :=vrai ;
        Si place déjà dans le chemin en cours alors Bool2 := vrai ;
        Si le chemin en cours possède k nœuds alors Bool3 :=vrai ;
        Si il n'y a plus de suivant alors Bool4 :=vrai ;
        CIOK := Bool1 and Bool3 ;
        DSOK := not ( Bool1 or Bool2 or Bool3 or Bool4 );
        Si not( CIOK or DSOK ) alors incrémenter le pointeur de succession ;
    Jusqu'à ( CIOK or DSOK or Bool4 );
    Si ( CIOK or DSOK ) alors
        Début
            a := suivant ; SU[ a, -1 ] := d ; { Affecter le pointeur de précédence }
            Si a<>début alors SU[A,0] :=1 ; { init pointeur succession}
        Fin
Fin
```

```
Début { Procédure Monter }
    a :=SU[d,-1] ; ADJ :=vrai ; { affectation de la place d'arrivée }
    Si a <>0 alors si SU[ a, SU[a, 0] = 0 alors ADJ := faux sinon a :=d ;
    SU[d, -1] :=0 { ré-initialisation du pointeur de précédence }
    Si d<>début alors SU[ d, 0 ] :=1 ;{ ré-initialisation du pointeur de succession }
Fin
```

4.3.2. La procédure « Recherche » des circuits.

```
Début { recherche des k-circuit pour k ∈ [ kmin.. kmax] }
    {Initialisation de la structure}
    k :=kmin ;
    ∀i ∈ [1.. n], ∀j ∈ [1.. ncirmax], Diagonale [i]:=0; CIR[i, j]:=0; TVC[i, j]:=0;
    Tant Que k<=kmax Faire { rang du circuit max non atteint }
        { initialisation d'indexes et de la k-diagonale }
        Pour i de 1 à dimgra faire Diagonale[i] :=X[k, i, i] ;
        MOK :=DIAG(k) ; Début :=0;
        Tant Que MOK Faire { Il existe encore des k-circuits }
            Choix de la place de départ la moins impliquée début et non déjà choisie ;
            Créer_Table( k, X, SU ) ;
            DOK :=vrai ; Courant :=début ; Chemin ← liste-vide ;
            Tant Que DOK Faire { la place Début est encore un départ de k-circuit }
                Répéter
                    Descendre(Courant, arrivée, DSOK, CIOK ) ;
                    Si DSOK or CIOK alors | Stockage(arrivée dans chemin) ;
                                          | Courant :=arrivée ;
                Jusqu'à ( CIOK or DSOK )
                Si CIOK alors | Stockage(chemin dans CIR) ;
                              | Si ENOK alors Mise à jour(Diagonale)
                Si DIAG and not(DSOK) alors
                Répéter
                    Monter( Courant, arrivée, ADJ ) ;
                    Si Courant<>arrivée alors Déstockage(courant dans chemin) ;
                Jusqu'à ( ADJ ou (courant=Début) );
                Si ( il n'y a plus de k-circuits ) or
                   ( arrivée = Début et Début n'a plus de suivant ) or
                   ( il n'y a plus de k-circuit impliquant Début )
                alors DOK :=faux sinon DOK :=vrai ;
            Fin Tant Que { la place Début est encore un départ de k-circuit }
            Si ( il n'y a plus de k-circuits ) or ( on n'en a pas trouvé )
            alors MOK :=faux sinon MOK :=vrai ;
        Fin Tant Que { Il existe encore des k-circuits }
        k :=k+1 ;
    Fin Tant Que { rang du circuit max non atteint }
Fin
```

Nous illustrons le principe de recherche des k-circuits sur la *Figure 2* .

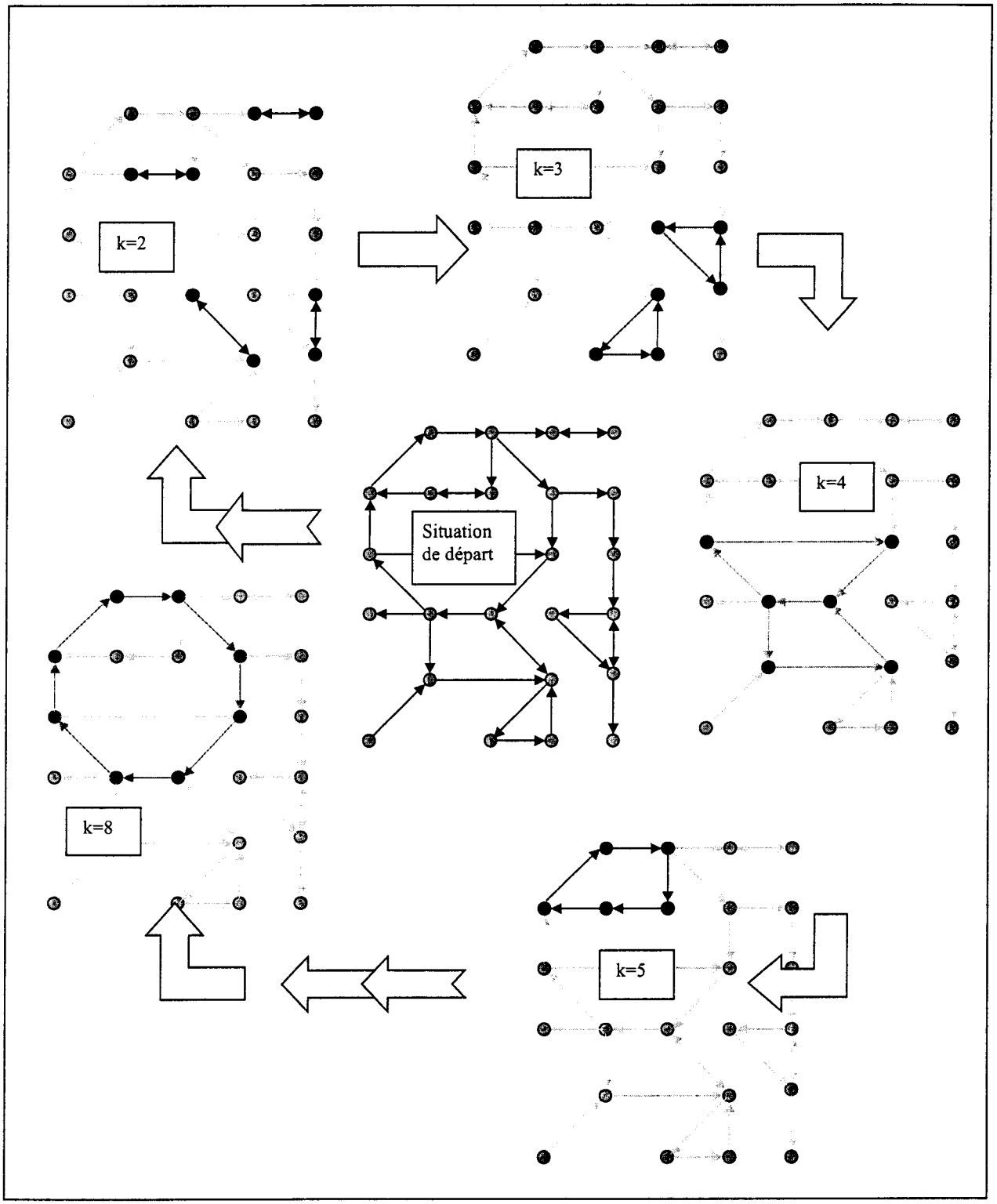


Figure 2 : Illustration de la recherche successive des circuits par rang.

A chaque rang k, nous ne considérons que les nœuds appartenant à des k-circuits et la recherche de ces circuits s'effectuent uniquement parmi ces nœuds. La diagonale de la matrice de précédence agit à la manière d'un filtre.

4.4. La procédure « Chemin ».

La procédure « Chemin » retourne tous les chemins permettant d'atteindre dans le graphe considéré, le sommet d'arrivée noté *a* depuis le sommet de départ noté *d*. Elle retourne également la robustesse de la liaison *d-a* en calculant le nombre de chemins indépendants qui relient le sommet d au sommet *a*. Afin de rendre notre procédure efficace, nous réduisons le graphe initial en éliminant de ce dernier tous les arcs reliant des sommets qui n'interviendront pas dans les chemins de *d* à *a*. Ce travail préparatoire de réduction du graphe initial est effectué par la procédure « Prépa_Chemin ».

4.4.1. Le travail préparatoire, procédure « Prépa_Chemin ».

Pour éliminer les arcs qui n'interviennent pas dans un chemin d-a, nous exploitons les résultats fournis par la matrice « somme des chemins » SIG. L'égalité *SIG[i,d]=0* signifie qu'il n'existe aucun chemin permettant d'aller de *d* à *i*, de même l'égalité *SIG[a,i]=0* signifie qu'il n'existe aucun chemin de *i* à *a*. Nous consignons ces résultats dans une matrice *XT* locale notée *XTL*. Initialement, *XTL[1,i,j] :=XT[1,i,j], ∀(i, j) ∈ [1..dimgra, 1..dimgra]*.

- **Pour** *i* de *1* à *dimgra* **faire**
  Si *(SIG[i, d]=0* and *i<>d)* or *(SIG[a, i]=0* and *i<>d)*
  Alors Pour *j* de *1* à *dimgra* **faire** | *XTL[1,i,j] :=0*
  | *XTL[1,j,i] :=0 ;*

Nous éliminons également les arcs liés aux prédécesseurs de *d* ainsi qu'aux successeurs de *a* qui par définition ne peuvent intervenir dans un chemin *d-a*.

- **Pour** *i* de *1* à *dimgra* **faire**
  | *XTL[1,i,a]=0* ; { *a* n'a plus de successeur }
  | *XTL[1,d,i]=0* ; { *d* n'a plus de prédécesseur }

Nous éliminons aussi les arcs reliant les prédécesseurs de *a* qui ne sont pas accessibles depuis *d* et les successeurs de *d* qui n'accèdent pas à *a*.

- **Pour** *i* de *1* à *dimgra* **faire**
  Si *XTL[1,a,i]<>0* and *SIG[i, d]=0*
  Alors Pour *j* de *1* à *dimgra* **faire** *XTL[1,a,i] :=0 ;*

- **Pour** *i* de *1* à *dimgra* **faire**
  Si *XTL[1,i,d]<>0* and *SIG[a, i]=0*
  Alors Pour *j* de *1* à *dimgra* **faire** *XTL[1,i,d] :=0 ;*

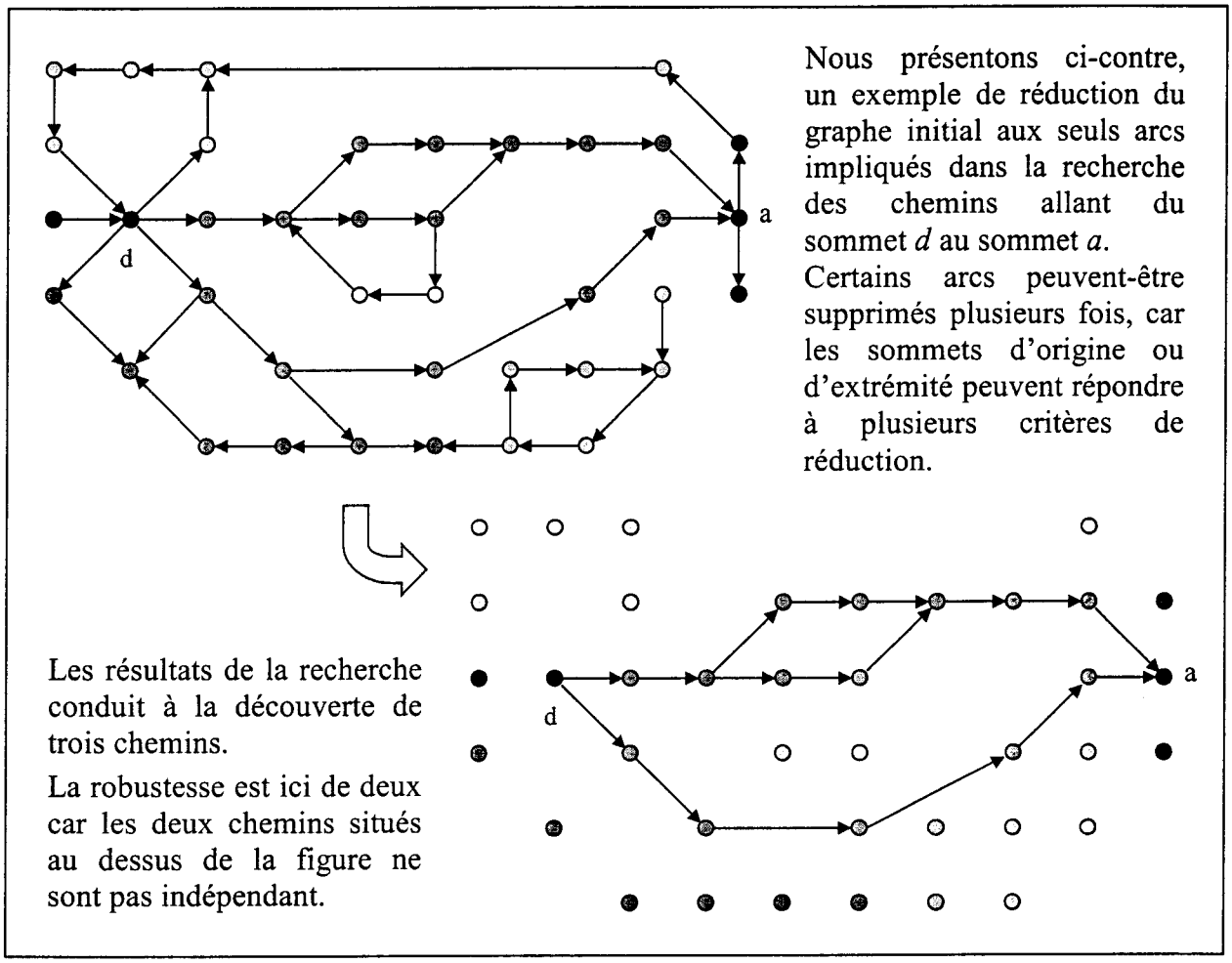


Figure 3 : Réduction du graphe pour la recherche des chemins de a à d.

Ayant éliminé l'ensemble de ces sommets, nous construisons les étages de *2* à *dimgra* de la matrice locale *XTL*, puis la matrice « somme des chemins » locale *SIGL*. La procédure « Génére_ES » étudié au paragraphe 2.2.2 permet alors de construire les matrices d'entrées et de sorties locales, respectivement *EL* et *SL*, à partir de la matrice *XTL[1, #, #]*.

4.4.2. La recherche effective des chemins.

De la même façon que pour la recherche des circuits, la recherche des chemins s'effectue en profondeur et elle utilise deux procédures « Monter » et « Descendre » qui gèrent à leur tour une structure basée sur la matrice *SU[-1..n,1..n]* d'entiers. La matrice *SU[-1..n,1..n]* est spécifique au sous-graphe de recherche des chemins de *d* à *a,* sous-graphe préparé par la procédure « Prépa_Chemin ». Cette matrice est construite par la procédure « Créer_table ». Comme pour la recherche des circuits, les procédures « Monter » et « Descendre » mettent à jour les données relatives au parcours du de l'arbre associé au sous-graphe selon la même codification que pour la recherche des circuits.

La procédure « Descendre » descend du nœud *courant* au nœud *suivant*. Elle positionne le booléen *OKDescente* à *vrai* si la descente est possible. La procédure « Monter » permet de monter du nœud *courant* au nœud *précédent*. Elle retourne le booléen *OKMontée* à vrai s'il n'y a plus de prédécesseur à *précédent* qui ait encore un successeur non visité (adjacent) ou si *précédent* est égal au nœud *début*.

```
Début { Procédure descendre }
    Suivant :=SU[courant, SU[ courant, 0]] ;
    Tant que (suivant ∈ Chemin) et (suivant<>rien) faire
        SU[courant, 0] :=SU[ courant, 0]+1 ;
        Suivant :=SU[courant, SU[ courant, 0]] ;
    Fin Tant Que
    Si ( not(suivant ∈ Chemin) et (suivant<>rien) )
    Alors { On a trouvé un suivant convenable }
    Début
        OKDescente :=vraie ;
        SU[courant, 0] :=SU[ courant, 0]+1 ; { MaJ du pointeur du prochain adjacent }
        SU[suivant, -1] :=courant ;          { MaJ du pointeur de retour          }
    Fin
    Si (suivant<>rien) alors
    Début { On n'a pas trouvé de suivant convenable }
        OKDescente :=faux ; Suivant :=courant ;
    Fin
Fin

Début { Procédure Monter }
    précédent :=SU[courant, -1] ;
    tampon :=SU[précédent, SU[ précédent, 0]] ;  { tampon est un éventuel adjacent }
    Tant que ( tampon ∈ Chemin) et ( tampon<>rien ) faire
        SU[précédent, 0] :=SU[ précédent, 0]+1 ;
        tampon :=SU[précédent, SU[ précédent, 0]] ;
    Fin Tant Que
    Si tampon=rien alors si précédent = départ alors
    début { il n'y a plus de successeur non visité au point départ }
        OKMontée :=faux ;
        précédent :=rien ;        { On note la non présence d'adjacent au point départ }
    fin
                                          sinon OKMontée :=vrai ;
                    sinon OKMontée :=faux ;
    SU[courant, 0] :=1 ; SU[courant, 0] :=1 ; { MaJ du pointeur de retour          }
Fin
```

```
Début { Procédure Chemin }
    Prépa_Chemin(A, départ, arrivée, AL, EL, SL, SIGL, XTL ) ;
    Créer_Table(AL, SU ) ;
    Si A[a,d]=1 alors Stocker_dans_CH(Chemin directe) ;
    PASFIN :=(SU[d, SU[d, 0]]<>0) ;
    Courant :=départ ;
    Tant Que PASFIN faire { il y a encore des chemins}
    |   Chemin :=liste-vide ;
    |   Tant Que courant<>arrivée faire
    |   |   Descendre(courant, suivant, OKDescente)
    |   |   courant :=suivant ;
    |   |   Si OKDescente
    |   |   alors Stocker( Suivant dans Chemin)
    |   |   sinon si suivant<>arrivée
    |   |         alors
    |   |             Début
    |   |             |   Répéter
    |   |             |   |   Monter(courant, précédent, OkMontée)
    |   |             |   |   courant :=précédent ;
    |   |             |   Jusqu'à pas(OKMontée)
    |   |             |   Si Précédent=rien alors PASFIN :=Faux
    |   |             Fin
    |   |         sinon rien
    |   |   Si PASFIN alors
    |   |    Début
    |   |    |   Stocker_dans_CH(Chemin);
    |   |    |   Répéter
    |   |    |   |   Monter(courant, précédent, OkMontée)
    |   |    |   |   courant :=précédent ;
    |   |    |   Jusqu'à pas(OKMontée)
    |   |    |   Si Précédent=rien alors  PASFIN :=Faux
    |   |    |                    sinon PASFIN :=vrai ;
    |   |    Fin { Si PASFIN ..}
    |   |   Calcul_Robustesse[1](CH) ;
    |   Fin Tant Que
    Fin Tant Que { il y a encore des chemins}
Fin
```

[1] La procédure calcule la robustesse, c'est à dire le nombre de chemins indépendants ne partageant que les nœuds de départ et d'arrivée en commun. Elle travaille sur les vecteurs chemins associés aux chemins. Ces vecteurs sont définis de façon analogue au vecteur circuit (Cf.4.3.1). Ils sont rangés dans la table de liste TVH.

4.5. La Procédure « Isthmes ».

4.5.1. Le travail préparatoire.

Toujours dans un souci d'efficacité, la recherche des isthmes s'effectue sur une structure de donnée allégée. Nous éliminons du graphe initial tous les arcs appartenant aux chemins « source » et aux chemins « puits ». C'est le rôle de la procédure « Supprime_source_puits » qui fournit à partir de la matrice de précédence initiale *A*, la matrice de précédence *SCP*. La procédure « supprime_circuits » élimine ensuite de cette structure intermédiaire *SCP*, tous les arcs appartenant à des circuits et donne finalement la matrice de précédence *SCI*.

Les procédures « Supprime_source_puits » et « supprime_circuits » sont décrites ci-dessous.

```
Début { Procédure Supprime_source_puits }
    ∀(i, j) ∈ [ 1.. dimgra, 1.. dimgra ], Z[i, j] :=0 ; { Initialisation matrice tampon }
    Pour i de 1 à n faire
        si CS[i, 1]<>0 alors                    { Stockage d'un chemin source }
            Début
                k :=1 ;
                Répéter
                    k :=k+1 ;
                    Si CS[i, k]<>0 alors Z[ CS[ i, k], CS[ i, k-1]]:=1;
                Jusqu'à CS[i, k]=0;
            Fin

    Pour i de 1 à n faire
        Si CP[i, 1]<>0 alors                    { Stockage d'un chemin puits   }
            Début
                k :=1 ;
                Répéter
                    k :=k+1 ;
                    Si CP[i, k]<>0 alors Z[ CP[ i, k-1], CP[ i, k]]:=1;
                Jusqu'à CP[i, k]=0;
            Fin
    Pour i de 1 à dimgra faire                  { suppression source-puits     }
        Pour j de 1 à dimgra faire SCP[i, j] := A[i, j] - Z[i, j] ;
    Pour i de 1 à dimgra faire SCP[i, i] :=0 ;  { suppression des boucles      }
Fin
```

```
Début { Procédure supprime_circuits }
    ∀(i, j) ∈ [ 1.. dimgra, 1.. dimgra ], Z[i, j] :=0 ; { Initialisation matrice tampon }
    Pour k de 2 à dimgra faire
    Début
        Pour i de 1 à ncirmax faire
        Début
            CC :=CIR[k,i] ;
            si CC est un k-circuit alors
            Début
                Z[CC[1],CC[k]] :=1 ;          { Stockage des k-circuit }
                Pour j de 2 à k faire Z[CC[j],CC[j-1]] :=1 ;
            Fin
        Fin
    Fin
    Pour i de 1 à dimgra faire                { suppression de tous les circuits}
        Pour j de 1 à dimgra faire SCI[i, j] := SCP[i, j] - Z[i, j] ;
Fin
```

L'action de ces deux procédures est schématisée *Figure 4*

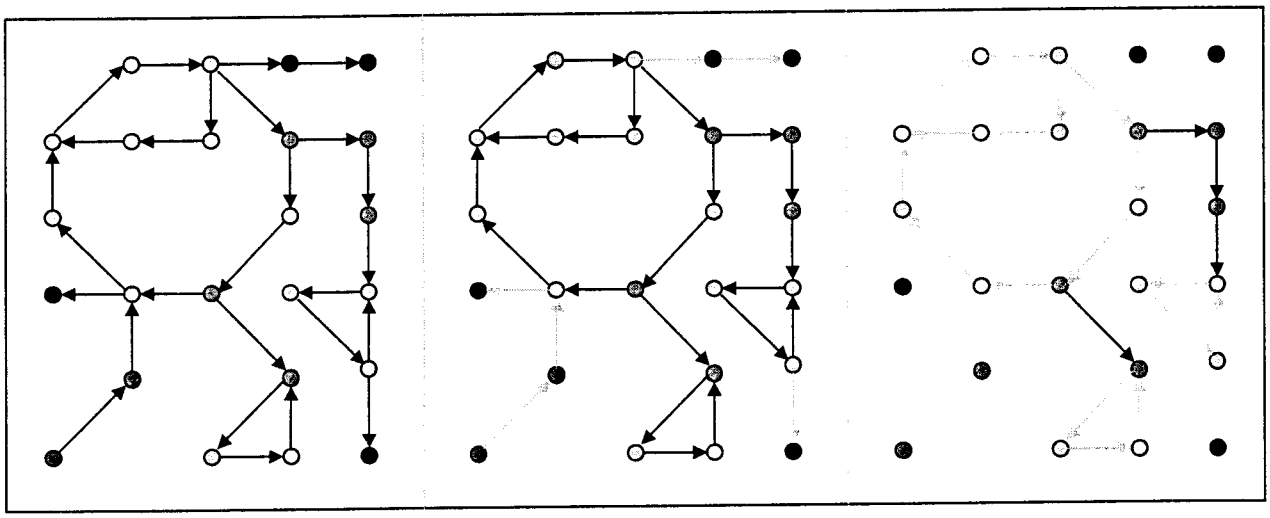


Figure 4 : Réduction de la structure pour extraction des isthmes.

4.5.2. La recherche effective des isthmes.

L'exécution de cette procédure ne peut se faire qu'après avoir rechercher les chemins source, les chemins puits et les circuits. En effet, elle génère et utilise les matrices d'entrées et de sorties locales, respectivement *EL* et *SL*, à partir de la matrice *SC*, elle même obtenue en éliminant les chemins source, les puits et les circuits de la matrice de précédence *A*. Les isthmes sont rangés dans la table *CI* de dimension [1.. n, 1.. n].

```
Début { Procédure isthme }
    Supprime_source_puits(A, CS, CP, SCP) ;
    supprime_circuits(SCP,CIR,SCI) ;
    Génére_ES(SCI,EL,SL) ;
    Latine(SCI,SIGL) ;
    ∀(i, j) ∈ [ 1.. dimgra, 1.. dimgra ], CI[i, j] :=0 ; { Initialisation matrice des Isthmes }
    Pour j de 1 à dimgra Faire
        Si ( SL[j, j]>=1 et EL[j, j]=0 ) alors        { j est un puits dans SCI        }
    Pour i de 1 à dimgra Faire
        Si ( SCI[i, j]>=1 et SIGL[j, i]=0 ) alors     { j est un départ d'isthme       }
            Début
                Stocker-dans_CI(j) ;
                Stocker-dans_CI(i) ;
                v:=i;
                Tant Que SL[v, v]<>0 Faire
                    u:=1;
                    Tant Que SCI[u, v]=0 et u<dimgra faire u:=u+1 ;
                    Si SCI[u, v]<>0 alors
                        Début
                            Stocker-dans_CI(u) ;
                            v:=u ;
                        Fin
                Fin Tant Que
            Fin
Fin
```

4.6. La Procédure « Articulation ».

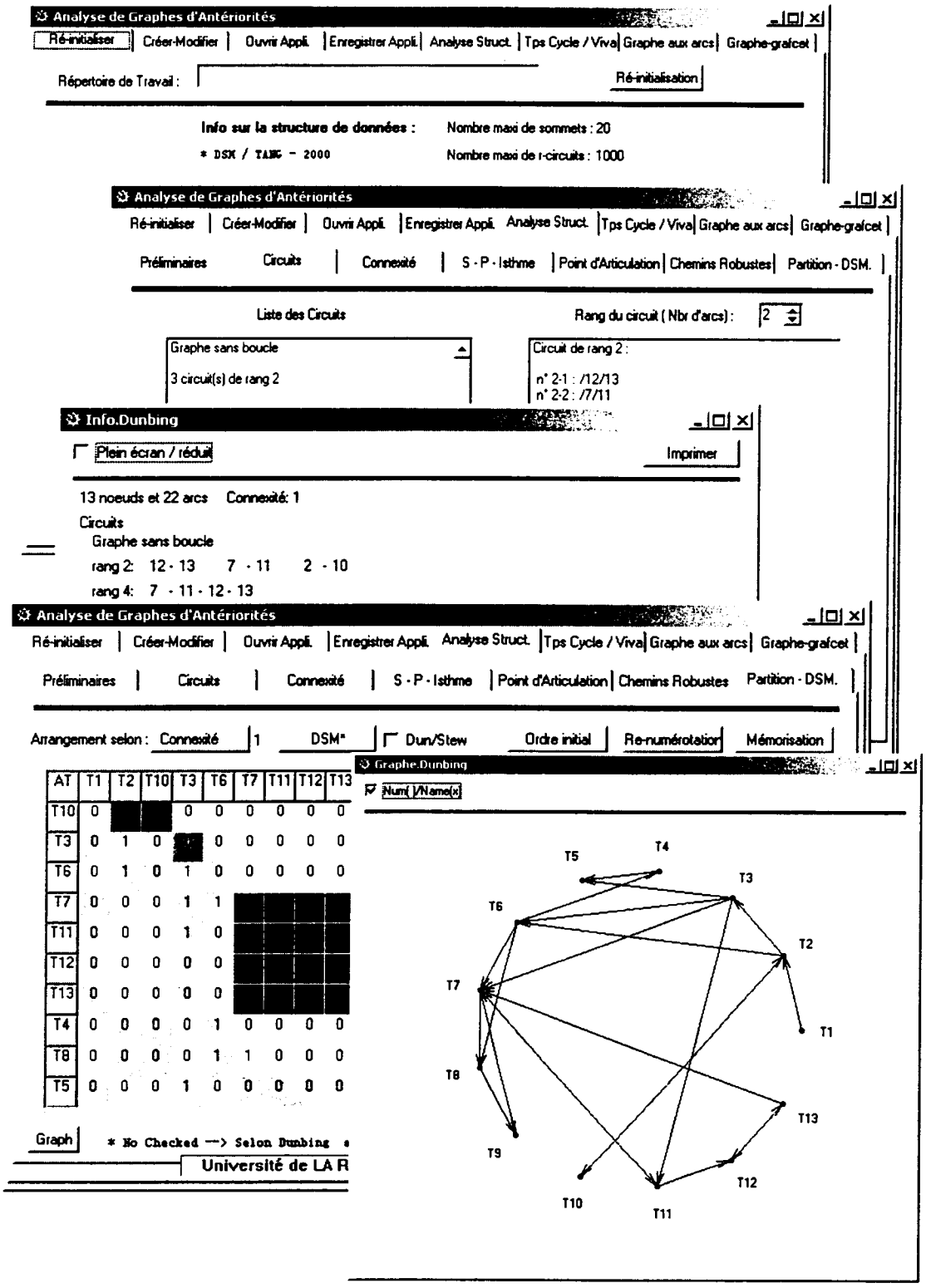
La recherche des points d'articulation nécessite la connaissance des circuits. La procédure développée examine les différentes intersections existant entre tous les circuits du graphe et vérifie que chaque intersection entre deux circuits, s'il elle existe, est limitée à un seul point. Les données d'entrée de cette procédure sont les deux tables de listes *CIR* et *TVC*, représentant respectivement les circuits et les vecteurs-circuits du graphe. Elle retourne la liste des points d'articulation *LPA*, qui contient pour chacun de ces points, le rang et le numéro des circuits impliqués dans l'articulation. *LPA* est du type tableau[1..n, 0..ncirmax,1..2]. Si i est un point d'articulation, $LPA[i, j, 1]=m_j$ et $LPA[i, j, 2]=k_j$ représentent respectivement le rang et le numéro dans le rang du $j^{ième}$ circuit impliqué dans le point d'articulation i. ($LPA[i, 0, 1]$ représente le nombre de circuits impliqués dans le point d'articulation i).

La procédure « Articulation » fait appel à deux procédures : « Suivant » et « Inter ». La procédure « Suivant » est chargée de rechercher dans la table *CIR*, le rang *mc* et le numéro dans le rang *kc* du circuit suivant le circuit encours (repéré par les deux indices *mp* et *kp*). Elle retourne un booléen *SUOK* à vrai s'il existe un suivant. La procédure « Inter » examine l'éventuelle intersection entre deux circuits respectivement repérés par les indices *(mp, kp) et (mc, kc)*. Elle retourne un booléen *PAOK* à vrai s'il existe un point d'articulation ainsi que le numéro de ce point noté *PA*. Si *PAOK* est *faux* et que *PA=0* alors les deux circuits sont disjoints. Si *PAOK* est *faux* et que *PA=x , (x<>0)* alors les deux circuits ont *x* sommets en commun. Une troisième procédure, appelée après la procédure « Articulation », vérifie le fait que les points trouvés soient bien des points d'articulation, c'est à dire que leur duplication augmente effectivement la connexité du graphe.

Nous donnons ci-dessous l'algorithme de la procédure « Articulation ».

```
Début { Procédure Articulation }
    ∀ (i, j ) ∈ [1.. n, 0.. ncirmax, 1..2 ],LPA[i, j, k] :=0 ; { Init. de la matrice LPA }
    mp :=1 ;kp :=1 ;
    Répéter
        mp :=mp+1 ; { recherche du premier circuit et de son suivant }
        Suivant(mp, kp, mc, kc, SUOK ) ;
    Jusqu'à (SUOK or mp=dimgra)
    Tant Que SUOK Faire
        Suivant(mp, kp, mc, kc, SUOK ) ;
        Tant Que SUOK Faire
            Inter(mp, kp, mc, kc, PAOK, PA ) ;
            Si PAOK
            alors { Stockage des circuits impliqués }
             Début
                LPA[PA,0,1] :=LPA[PA,0,1] +1 ;
                LPA[PA,LPA[PA, 0, 1], 1] :=mp ; LPA[PA,LPA[PA, 0, 1], 2] :=kp ;
                LPA[PA,0,1] :=LPA[PA,0,1] +1 ;
                LPA[PA,LPA[PA, 0, 1], 1] :=mc ; LPA[PA,LPA[PA, 0, 1], 2] :=kc ;
             Fin
            Suivant(mc, kc, mc, kc, SUOK ) ; { On change de deuxième circuit }
        Fin Tant Que
        Suivant(mp, kp, mc, kc, SUOK ) ; { On change de premier circuit }
    Fin Tant Que
Fin
```

5. Copies d'écran et bibliographie.



[HARARY F., 62] HARARY F., 62 :*"A graph approach to matrix inversion by partitioning."*. Numerische Mathematik., Vol 4,

[TANG D., & al., 00] TANG D., & al., 00 :*"Re-engineering of the design process for concurrent engineering."*. Computer & industrial engineering, Vol 38, n°pp 479-491.

Annexe 2 : Procédures spécifiques à l'ordonnancement cyclique.

1. Présentation

L'outil réalisé permet de calculer l'ordonnancement cyclique de gammes non linéaires, au sens des possibilités d'assemblage et désassemblage et à partir des graphes de précédences associés à ces gammes. Cet outil ne détermine que les instants où les ressources sont allouées aux tâches. Les couples « tâche-ressource » sont des données d'entrée du problème d'ordonnancement et il n'existe aucune flexibilité de permutation, d'affectation ou de procédé. En référence à la hiérarchisation des décisions proposée par KORBAA [KORBAA O., 98], le travail effectué par cet outil est représentable par les classes *D5, D6* et *D7*. Le résultat est donné sous la forme d'un diagramme de GANTT duquel on extrait un graphe de commande de type événement.

Après une description de la structure de données, nous présentons les procédures relatives à la simulation et à l'arbitrage des conflits d'allocation tâche-ressource. L'arbitrage est réglé suivant deux méthodes qui font appel à deux ensembles de procédures :

- Le premier ensemble utilise les règles standards de l'ordonnancement.
- Le deuxième ensemble utilise le comportement des colonies de fourmis auquel nous avons ajouté le paramètre temporel.

Sans développer les procédures qui y sont relatives, nous montrons enfin les possibilités de fonctionnement du simulateur en mode pas à pas. Ces procédures sont écrites sous DELPHI et représentent un fichier exécutable d'environ 500 Ko.

2. La structure de données.

2.1. Les données d'entrées.

2.1.1. Les données spécifiques à l'application.

nbt	Nombre de tâches
nbr	Nombre de ressource ($\leq$ *nbt*)
GD	Matrice du graphe de précédence des gammes opératoires, dimension *[0 .. nbt+1]* ^{2}, valeurs prises : *0 ou 1*
Duree	Tables des durées de chaque tâche, dimension *[0 .. nbt+1]* , temps entiers

Ressour: Tables d'affectation des ressources aux tâches, dimension *[0 .. nbt+1]*

$\forall i$, *Ressour[i]=j, j∈[1.. nbr]*

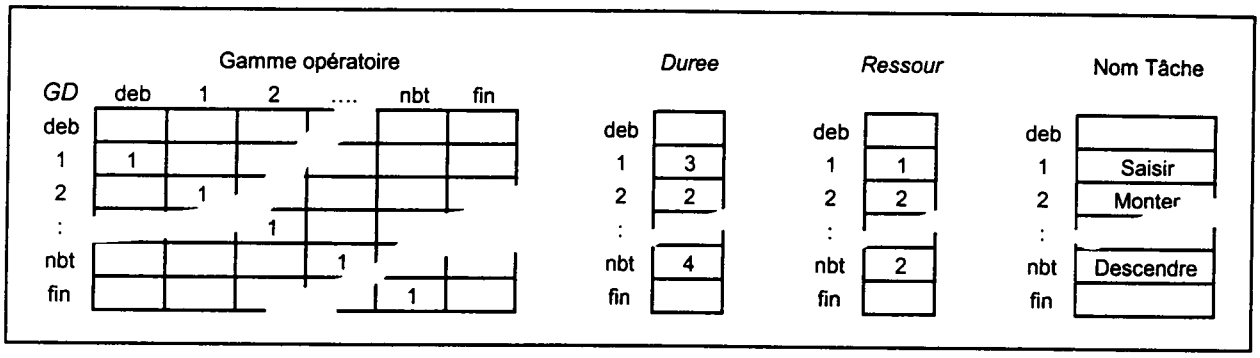


Figure 1 : Données spécifiques à une application.

2.1.2. Les données spécifiques à une simulation

- Grandeurs saisies

X Nombre de produits (ou de *MSP*) à traiter pendant la simulation

XR Nombre maximal de ressources simultanément actives

- Grandeurs calculées, Structures de données construites:

Tgoulet Temps d'occupation de la machine goulet par produit (ou *MSP*)

$$Tgoulet = \max_{j=1}^{j=nbr} \left(\sum_{i=1}^{i=nbt} Duree(i) \mid (ressour(i) = j) \right)$$

Tsigma Somme des durées $Tsigma = \sum_{i=1}^{i=nbt} Duree(i)$

TL Temps Limite de simulation : $TL \leftarrow X*Tsigma$

TE Temps exact de simulation pour absorber tous les encours.

TU Temps réel ($TU \leq TL$)

GT Matrice du graphe dupliqué, dimension *[0..(X*nbt)+1]²* dans *[0,1]*

ET, ST Matrice d'entrées/Sortie du graphe de travail,

dimension *[0..(X*nbt)+1]²*, valeurs prises : de *0* à *nbt* maxi.

MT Matrice du marquage du graphe de travail,

dimension *[0..(X*nbt)+1]²*, valeurs prises : *0 ou 1*

Avant_V Table d'avancement de la fabrication des produits (ou *MSP*)

dimension *[1..X]*, valeurs prises entre *0* et le nombre d'opérations de la gamme.

Charge_R Table mémorisant la charge globale des ressources pour la simulation,

dimension *[0..nbr+1]*, valeurs prises entre *0* et *Tgoulet*

Etat_R Table mémorisant l'état des ressources en cours de simulation,

dimension *[0..nbr+1]*, valeurs prises *0* ou *1*, respectivement libre ou occupée

Dema_R Table mémorisant le nombre de demandes enregistrées pour chaque ressource,

dimension *[0..nbr+1]*, valeur prises entre *0* et *X*nbt*

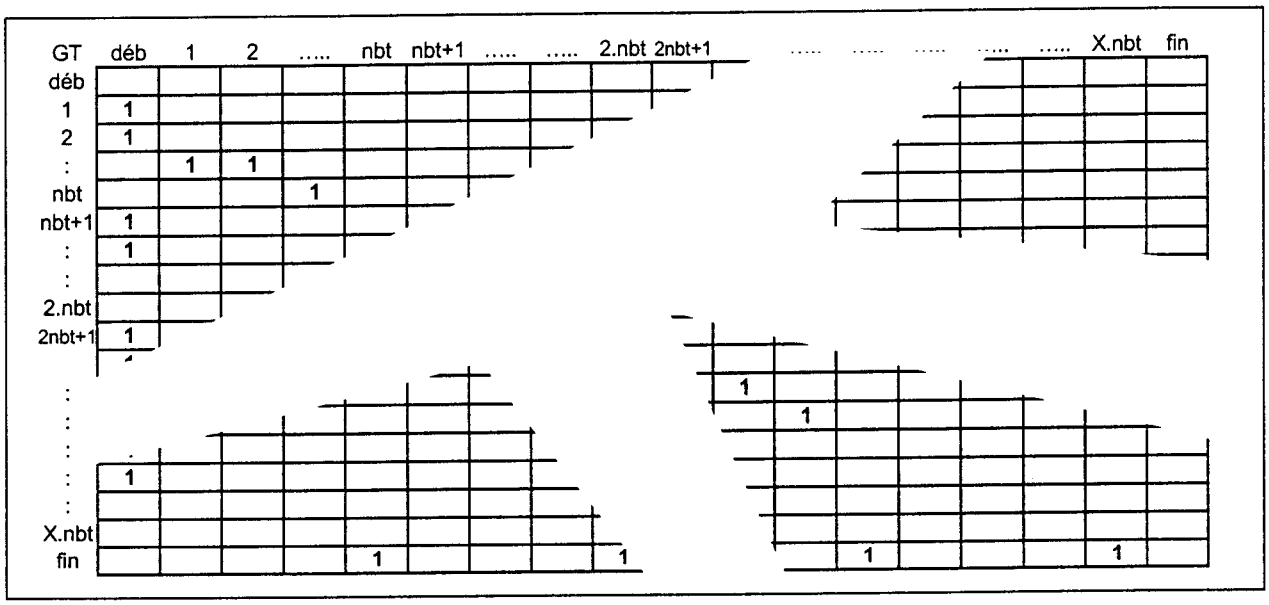


Figure 2 : Structure des matrices GT, ET, ST et MT.

DT Matrice de données de travail, dimension [*1..8 , 0..(X*nbt)+1*]

DT[1,i] Durée de la $i^{ème}$ tâche

DT[2,i] Temps écoulé depuis le début de la $i^{ème}$ tâche

DT[3,i] Nombre d'arcs entrant sur la $i^{ème}$ tâche

DT[4,i] Nombre d'arcs sortant de la $i^{ème}$ tâche

DT[5,i] Numéro de la ressource associée ou utilisée par la $i^{ème}$ tâche

DT[6,i] Etat de la $i^{ème}$ tâche

(*0* → inactive, *1* → activable, *2* → active et *3* → désactivée)

DT[7,i] Temps écoulé depuis la demande formulée par la $i^{ème}$ tâche

DT[8,i] Résultat des élections et de la sélection de la $i^{ème}$ tâche

(*0* → non élue ou non sélectionnée, *1* → élue, *2* → sélectionnée.)

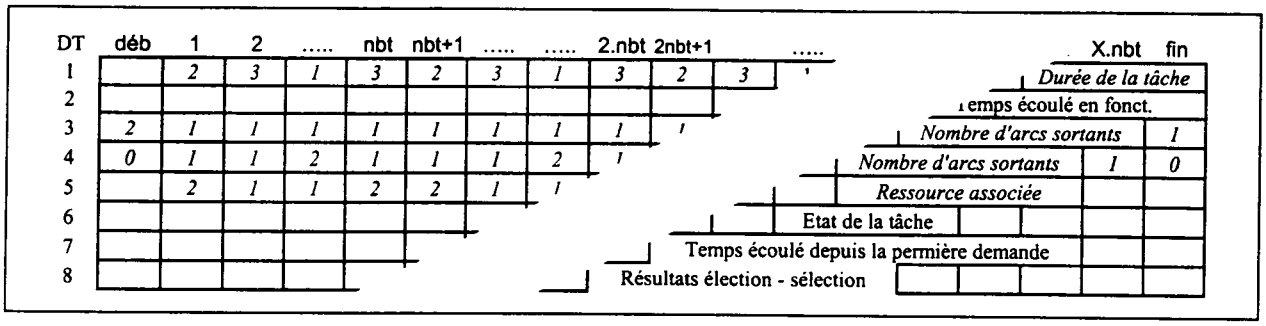


Figure 3 : Structure de la table des données de travail DT

2.1.3. Structure spécifique à l'arbitrage par colonies de fourmis.

Nb_fourmi Nombre de fourmis par génération

Nb_Cycle Numéro de la colonie ou de la génération en cours, *0<Nb_cycle <Nb_cycle_limite*

Nb_Cycle_limite, Nombre maximal de colonies ou de générations

Phero Dimension 3 *[0..X*nbt+1, 0..nbr+1,0..TL],* Matrice de phéromone. Cf. *Figure 4*

On mémorise dans *Phero[i, u, t]* la probabilité *de* chaque tâche *i*, $0 < i < X*nbt+1$, d'être affectée à la ressource *u*, $0 < u < nbr+1$ à l'instant *t*, $0 < t < TL$, *Phero[0, u, t]* représente la probabilité de choisir la ressource *u* pour l'ordonnancement à l'instant *t* (dans le cas d'un nombre de ressources sélectionnées supérieur au nombre de ressources tolérées).

Copie_Phero Matrice identique à la matrice *Phero*. On note dans *Copie_Phero*, le résultat des affectations en cours de la simulation d'une génération.

Seuil Seuil en pourcentage de déséquilibrage aléatoire de la matrice initiale *Phero*. Si *seuil=0* alors, les probabilités d'allocations sont identiques (équiprobabilité).

Bruit Pourcentage de bruitage aléatoire des probabilités calculées après passage d'une génération.

Héritage Coefficient compris entre *0* et *un* permettant de faire hériter la nouvelle matrice *Phero* utilisable à la $j^{ième}$ génération, des matrices *Phero* de générations *j-1* et *j-2*.

$Phero^{j} = Héritage * Phero^{j-2} + (1 - Héritage) * Phero^{j-1}$

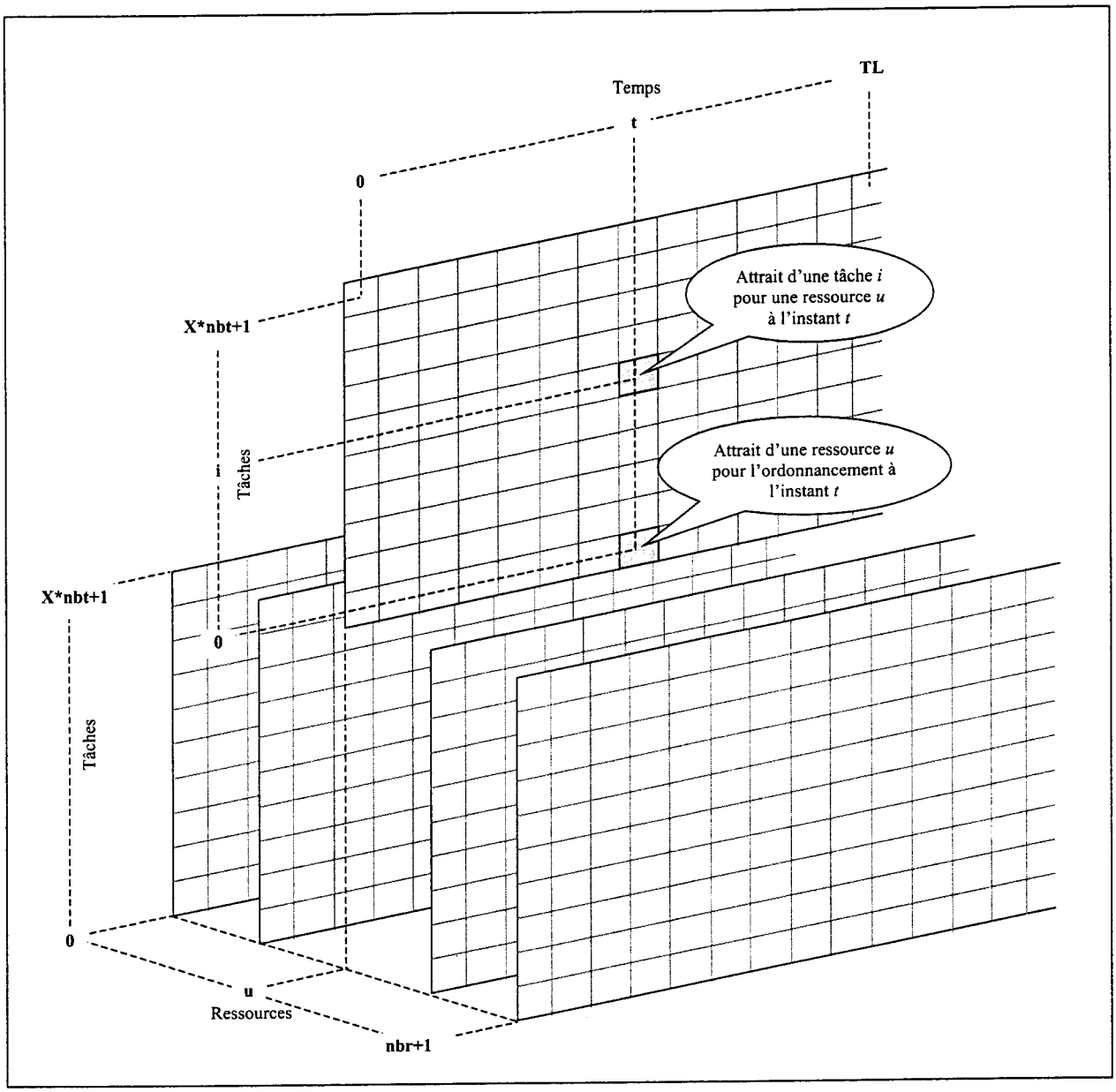


Figure 4 : Matrice Phero - Structure dédiée à l'arbitrage par colonies de fourmis.

On remarque sur cette figure le codage particulier pour chaque instant t et pour chaque ressource u, des emplacements *Phero[0,u,t]*. Chacune de ces variables représente la probabilité de choisir cette ressource à cet instant.

2.2. *Les données résultantes.*

Tgantt Résultat de l'ordonnancement donné par un diagramme de GANTT. Cette table contient pour chaque instant l'occupation par les tâches de chaque ressource, dimension *[1..nbr , 1..TE]* où *nbr* représente le nombre de ressources et *TE* le temps exact de simulation

3. Les procédures relatives à la simulation.

Un premier ensemble de procédures permet la saisie des données d'une application, c'est à dire du nombre de tâche, *nbt*, du nombre de ressource, *nbr*, de la durée (*Duree*) de chacune des tâches et de la ressource (*Ressour*) sur laquelle elle est effectuée. Nous consignons également les arcs du graphe de précédence des gammes opératoires dans la matrice *GD*. Ces données sont stockées dans un fichier **.ord* et peuvent être rechargées et/ou modifiées. Elles sont indépendantes de la longueur de l'horizon temporel, c'est à dire du nombre *X* de produits à réaliser dans le plan de charge, et du nombre *Xr* toléré de ressources simultanément actives.

Après avoir saisi *X* et *Xr*, une procédure de préparation construit le graphe de travail *GT* et la table de travail *DT*. Cette procédure construit également la table de marquage *MT*.

La simulation consiste à faire évoluer le marquage *MT* du graphe représenté par la matrice *GT*, en respectant d'une part les règles d'activation/désactivation des sommets d'un graphe de type événement (marquage des places précédentes) et d'autre part en évaluant les conditions d'activation /désactivation de ces sommets. Chacune de ces conditions est le produit logique de deux conditions élémentaires : L'élection et la sélection. La première porte sur la possibilité ou non d'attribuer la ressource à la tâche considérée et la seconde porte sur la possibilité ou non d'activer cette ressource. Dans les deux cas, nous sommes très souvent confrontés à des possibilités multiples d'attribution et/ou d'activation : Par exemple, plusieurs tâches peuvent demander simultanément la même ressource. Il importe alors de mettre en place des règles d'arbitrage permettant de gérer ces conflits (Cf. paragraphe 4.).

La deuxième condition se justifie par le fait que le nombre *Xr* de ressources simultanément actives influe sur le nombre d'encours. Lorsqu'on fait croître progressivement *Xr*, le temps de cycle diminue pour finalement prendre la valeur de saturation de la ressource goulet *CT*. A ce stade, l'augmentation de *Xr* n'apporte plus rien en terme de productivité, si ce n'est une augmentation néfaste du nombre des encours. En utilisant une méthode d'ordonnancement basée sur la simulation, le nombre des encours apparaît donc comme une *conséquence* de cet ordonnancement mais nous pouvons agir indirectement et de manière qualitative sur ce nombre en limitant le nombre de ressources simultanément actives *Xr*. C'est pourquoi, en plus des conflits d'allocation tâches-ressources, nous gardons la possibilité d'arbitrer les conflits inter-ressources d'activité, de façon à ce que le nombre de ces ressources effectivement et simultanément actives ne dépasse pas le nombre *Xr* choisi a priori (sélection, deuxième arbitrage). Le simulateur propre aux colonies de fourmis est abordé au paragraphe *4.2.1.*

```
Début { Procédure Simulation }
    Initialisation de la structure de données
        ∀( u, v) ∈ [ 0..n]², MT[u, v] :=0 ; ∀u ∈[ 2, 6, 7, 8 ], ∀v ∈ [ 0..nbr], DT[ u, v] :=0 ;
        ∀u ∈ [ 0..nbr], Etat_R[ u] :=0, Dema_R[u] :=0 ; Mise_à_jour (Charge_R[u]);
        ∀u ∈[0..nbr], ∀v ∈ [ 0..TL], Tgantt[u, v]:=0;
    Initialisation du temps, TU:=-1, DT[ 0, 6] :=2 ; DT[ 0, 1] :=1

    Tant Que ( TU<TL) et (DT[6, ((X*nbt+1)+1]<>2) Faire

        { 0- Mise à jour de l'état des tâches}
        { Incrémentation du temps } TU :=TU+1 ;
        Pour i de 0 à X*nbt+1 Faire Si DT[6, i]=2 alors DT[2, i] :=DT[2, i]+1
        { Mise à jour de l'activité }
        Pour i de 0 à X*nbt+1 Faire Si DT[6, i]>1 alors
        Si DT[2,i]<=DT[1,i] Alors {tâche encore active}
        Début
            DT[6, i] :=2 ; Mise_à_jour_GANTT ;
            Si DT[2, i] = DT[1,i] alors DT[6,i] :=3;
        Fin
                                Sinon DT[6,i] :=3{ tâche terminée }

        { 1- Désactivation des tâches désactivables }
        Pour i de 0 à X*nbt+1 Faire Si DT[6, i]=3 Alors
        Début
            DT[2, i ] :=0 ; DT[6, i] :=0 ; { la tâche i devient inactive }
            Etat_R[Nressource(i)[1]]:=0; { la ressource réalisant i devient libre }
            { On propage la désactivation de i dans le graphe }
            Pour j de 0 à X*nbt+1 Faire Si GT[j, i]=1 Alors MT[j, i] :=MT[j, i] +1 ;
        Fin

        { 2- Activabilité de la tâche i}
        Pour i de 0 à X*nbt+1 Faire Si DT[6, i]=0 Alors
        Début
            Som :=0 ; { test si tous les prédécesseurs de i sont terminés }
            Pour j de 0 à X*nbt+1 Faire Si GT[i, j]=1 et MT[j, i]>=1
                                        Alors som :=som+1 ;
            Si ( DT[3,i]<>0 et DT[3,i]=som ) Alors
             Début
                DT[6, i]=0 ;
```

[1] La procédure *Nressource(i)* renvoie le numéro de la ressource qui exécute la tâche i.

```
        |   |   Dema_R[NRessource[i]]:= Dema_R[NRessource[i]]+1 ;
        |   Fin
        Fin
        { mémorisation du nombre de demandes de ressource par la tâche i }
        Pour i de 0 à X*nbt+1 Faire Si DT[6, i]=1 Alors DT[7, i]= DT[7, i] +1 ;

        { 3- Election des tâches potentiellement activables }
        { DT[7,i] contient le nombre demandes non satisfaites effectuées par la tâche i. Au
        retour, DT[8,i] contient la liste des décisions prises par l'arbitre : Si DT[8,i] =0 alors
        la tâche i reste en attente, Si DT[8, i]=1 alors la tâche i peut-être activée. Mode_T
        spécifie la type de règles utilisées pour réaliser l'arbitrage entre les tâches; Cf.
        paragraphe 4.1}
        Arbitrage_T( mode_T, DT, DT ) ;

        { 4- Sélection des tâches élues, sans dépasser le nombre toléré de ressources
        simultanément occupées : Au retour, DT[8,i] contient la liste des décisions prises
        par l'arbitre. Si DT[8,i] =0 alors la tâche i reste en attente, Si DT[8, i]=1 alors la
        tâche i est activée de suite. Mode_R spécifie la type de règles utilisées pour réaliser
        l'arbitrage entre les ressources; Cf. paragraphe 4.1 }
        Arbitrage_R( mode_T, DT, DT ) ;

        { 5- Activation des tâches élues et sélectionnées }
        Pour i de 0 à X*nbt+1 Faire Si ( DT[6, i]=1 et DT[8, i]=1 ) Alors
        Début
            DT[6, i] :=1 ; { La tâche i est activée }
            DT[2, i] :=0 ; { Mise à jour du temps d'activité de la tâche i }
            DT[7, i] :=0 ; { Mise à jour du nombre de demande de la tâche i }
            DT[7, i] :=0 ; { On annule la décision sur la tâche i }
            { Le nombre de demande de la ressource décroît d'une unité }
            Dema_R[Nressource(i)] := Dema_R[Nressource(i)] -1 ;
            Etat_R[Nressource(i)] :=1 ; { ressource allouée }
            Charge_R[Nressource(i)] := Charge_R[Nressource(i)]-DT[1, i] ;
            { On note l'état d'avancement du produit associé à la tâche i }
            Avant_V[Nmachine_Virtuelle(i)[2]] := Avant_V[Nmachine_Virtuelle(i)]+1 ;
        Fin
    Fin tant Que
    Si non ( TU<TL) et (DT[6, ((X*nbt+1)+1]<>2) alors Affiche_Gantt
Fin { Procédure Simulation }
```

[2] La procédure NMachine_Virtuelle*(i)* renvoie le numéro de la machine virtuelle qui exécute la tâche i.

4. Les procédures d'arbitrage.

4.1. Arbitrages suivant les règles standards.

4.1.1. Arbitrage des tâches : La procédure Arbitre_T.

La procédure *Arbitre_T(Mode_T, DET, DST)* travaille sur la table *DT, Data-Travail,* selon la valeur du sélecteur *Mode_T*. Les paramètres formels *DET* et *DST* sont associés au paramètre effectif *DT*.

```
Début { Procédure Arbitre_T }
    Selon Mode_T faire
        1 :{ Arbitrage suivant la règle FIFO, Fisrt In Fist Out }
        Arbitre_FIFO(DET, DST) ;
        2 :{ Arbitrage suivant la règle SPT, Shortest Processing Time }
        Arbitre_SPT(DET, DST) ;
        3 :{ Arbitrage suivant la règle LPT, Longest Processing Time }
        Arbitre_LPT(DET, DST) ;
Fin
```

Nous détaillons ci-après les procédures *Arbitre_FIFO, Arbitre_SPT* et *Arbitre_LPT*.

```
Début { Procédure Arbitre_FIFO }
    DST :=DET ;
    Pour i de 0 à X*nbt+1 Faire DST[8, i]=0 ;
    Pour u de 0 à nbr+1 Faire
    Si ( Dema_R[u]>=1 et Etat_R[u]=0 ) Alors
    Début { Il y a plusieurs demandes pour la ressource libre u }
        Max :=0 ; num_T :=0 ;
        Pour i de 0 à X*nbt+1 Faire
        Si (( Nressource(i)=u ) et ( DET[6, i]=1)) Alors
        Si DET[7, i]>max Alors
        Début { La tâche i a momentanément la demande la plus ancienne }
            num_T :=i ;
            max :=DET[7, i] ;
        Fin
        DST[8, num_T]:=1
    Fin
Fin
```

Cette procédure assure l'allocation suivant le principe du premier arrivé, premier servi.

```
Début { Procédure Arbitre_SPT }
    DST :=DET ;
    Pour i de 0 à X*nbt+1 Faire DST[8, i]=0 ;
    Pour u de 0 à nbr+1 Faire
    Si ( Dema_R[u]>=1 et Etat_R[u]=0 ) Alors
    Début { Il y a plusieurs demandes pour la ressource libre u }
        Min :=TL ; num_T :=0 ;
        Pour i de 0 à X*nbt+1 Faire
        Si (( Nressource(i)=u ) et ( DET[6, i]=1)) Alors
        Si DET[1, i]<min Alors
        Début { La tâche i a momentanément la demande la plus ancienne }
            num_T :=i ;
            min :=DET[1, i] ;
        Fin
        DST[8, num_T]:=1;
    Fin
Fin
```

```
Début { Procédure Arbitre_LPT }
    DST :=DET ;
    Pour i de 0 à X*nbt+1 Faire DST[8, i]=0 ;
    Pour u de 0 à nbr+1 Faire
    Si ( Dema_R[u]>=1 et Etat_R[u]=0 ) Alors
    Début { Il y a plusieurs demandes pour la ressource libre u }
        Max := -TL ; num_T :=0 ;
        Pour i de 0 à X*nbt+1 Faire
        Si (( Nressource(i)=u ) et ( DET[6, i]=1)) Alors
        Si DET[1, i]>max Alors
        Début { La tâche i a momentanément la demande la plus ancienne }
            num_T :=i ;
            max :=DET[1, i] ;
        Fin
        DST[8, num_T]:=1;
    Fin
Fin
```

Les deux règles SPT et LPT sont contradictoires et correspondent à deux types de politiques. Soit on termine au plutôt les tâches les plus courtes pour en limiter le nombre, soit au contraire, on s'attache à terminer les tâches les plus longues. Il semble, qu'avec une technique de placement de tâches, la deuxième solution offre un plus grand nombre de possibilités.

4.1.2. Arbitrage des ressources : La procédure Arbitre_R

La procédure *Arbitre_R(Mode_R, DET, DST)* travaille également sur la table *DT, Data-Travail,* selon la valeur du sélecteur *Mode_R :* Si *Mode_R=1* alors la procédure choisit la ressource associée au travail le plus avancé (*LWR, Least Work Remaining*), si *Mode_R=2* alors la procédure choisit la ressource pour laquelle la charge de travail restante est la plus grande (*GWL, Greatest Work Load*).

Nous détaillons ci-après les procédures *Arbitre_LWR* et *Arbitre_GWL.*

```
Début { Procédure Arbitre_LWR }
    DST :=DET ; Pour i de 0 à X*nbt+1 Faire DST[8, i]=0 ;
    { Calcul du nombre de ressources occupées }
    Xactuel :=0 ; Pour u de 0 à nbr+1 Faire Xactuel :=Xactuel +Etat_R[u] ;
    Si Xactuel=Xr Alors { Pas d'allocation possible }
                  Sinon { Allocation possible s'il y a des tâches élues }
    Début
        { Calcul du nombre de tâches élues }
        som :=0 ; Pour i de 0 à X*nbt+1 Faire Si DET[8, i]>0 Alors som :=som+1 ;
        Si som>( Xr - Xactuel ) Alors
        Début { Il y a plus de tâches élues que de ressources tolérées en activité simultanée }
            Répéter
                { Recherche de la tâche élue associée à la machine virtuelle la plus avancée }
                Avancement :=-1 ; N_Tache :=0 ;
                Pour u de 0 à nbr+1 Faire { Pour chaque ressource }
                  Si Dema_R[u]>0 Alors { S'il y a une demande }
                     Pour i de 0 à X*nbt+1 Faire { Si la tâche associée est élue }
                     Si ( (DET[5, i]=u) et ( DST[8, i]=1)) Alors
                        Si ( Avant_V[Nmachine_Virtuelle(i)]>Avancement ) Alors
                        Début { Si la tâche élue est associée à la ressource la plus avancée }
                           N_Tache := i ;
                           Avancement := Avant_V[Nmachine_Virtuelle(i)]
                        Fin
                DST[8, i]=2 ;som :=som-1 ; Xactuel:=Xactuel+1 ;
            Jusqu'à ( Xr = Xactuel ) ou (som=0)
            Pour i de 0 à X*nbt+1 Faire Si DST[8, i]=1 Alors DST[8, i]=0 ;
            Pour i de 0 à X*nbt+1 Faire Si DST[8, i]=2 Alors DST[8, i]=1 ;
        Fin
    Fin
Fin
```

```
Début { Procédure Arbitre_GWL }
    DST :=DET ; Pour i de 0 à X*nbt+1 Faire DST[8, i]=0 ;
    { Calcul du nombre de ressources occupées }
    Xactuel :=0 ; Pour u de 0 à nbr+1 Faire Xactuel :=Xactuel +Etat_R[u] ;
    Si Xactuel=Xr Alors { Pas d'allocation possible }
                  Sinon { Allocation possible s'il y a des tâches élues }
    Début
        { Calcul du nombre de tâches élues }
        som :=0 ; Pour i de 0 à X*nbt+1 Faire Si DET[8, i]>0 Alors som :=som+1 ;
        Si som>( Xr - Xactuel ) Alors
        Début { Il y a plus de tâches élues que de ressources tolérées en activité simultanée }
            Répéter
                { Recherche de la tâche élue associée à la ressource la plus chargée }
                Charge :=0 ; N_Tache :=0 ;
                Pour u de 0 à nbr+1 Faire { Pour chaque ressource }
                  Si Dema_R[u]>0 Alors { S'il y a une demande }
                     Pour i de 0 à X*nbt+1 Faire
                     Si ( (DET[5, i]=u) et ( DST[8, i]=1)) Alors
                        Si ( Charge_R[Nressource(i)]>Charge ) Alors
                        Début
                           N_Tache := i ;
                           Charge := Charge_R[Nressource(i)]
                        Fin
                DST[8, i]=2 ;som :=som-1 ; Xactuel:=Xactuel+1 ;
            Jusqu'à ( Xr = Xactuel ) ou (som=0)
            Pour i de 0 à X*nbt+1 Faire Si DST[8, i]=1 Alors DST[8, i]=0 ;
            Pour i de 0 à X*nbt+1 Faire Si DST[8, i]=2 Alors DST[8, i]=1 ;
        Fin
    Fin
Fin
```

La première règle (LWR) fluidifie la circulation des produits puisqu'elle favorise la sortie de ceux qui sont le plus avancés. A choisir, on préfère terminer un produit plutôt que d'en injecter un nouveau dans le circuit de production.

La deuxième règle (GWL) favorise l'équilibrage de l'engagement des ressources puisque c'est celle à qui il reste la plus grande charge qui est sélectionnée.

Axées sur les tâches ou sur les ressources, ces deux règles contribuent à la limitation des encours (fluidification du trafic et résorption des goulets temporaires).

4.2. Arbitrage basé sur les colonies de fourmis.

Les principes des méthodes de recherche basée sur le comportement des insectes sociaux [BONABEAU E. & al., 94] et plus particulièrement sur celui des colonies de fourmis [DORIGO M. & al., 91] sont relativement récents. A la différence des algorithmes présentés au chapitre 3 de [LOPEZ P. & al., 01], nous ne considérons pas l'attrait d'une tâche pour une ressource, mais l'attrait d'une tâche pour l'ordonnancement à *l'instant où nous devons prendre cette décision*. La prise en compte du temps représente la principale *originalité* de notre algorithme.

Dans la situation courante de cet algorithme d'arbitrage suivant le principe des colonies de fourmis (Cf. *Figure 5.*), nous consignons dans une structure de données appropriée, à chaque instant *t* du temps incrémental, et pour chaque fourmi d'une colonie (ou d'une génération), le résultat du règlement des conflits présents à cet instant. Ces règlements s'appuient sur la probabilité d'allocation qu'a eu une tâche pour une ressource lors du passage de toutes les fourmis de la colonie précédente, et à ce même instant. Il en est de même pour les conflits de sélection des ressources entre-elles.

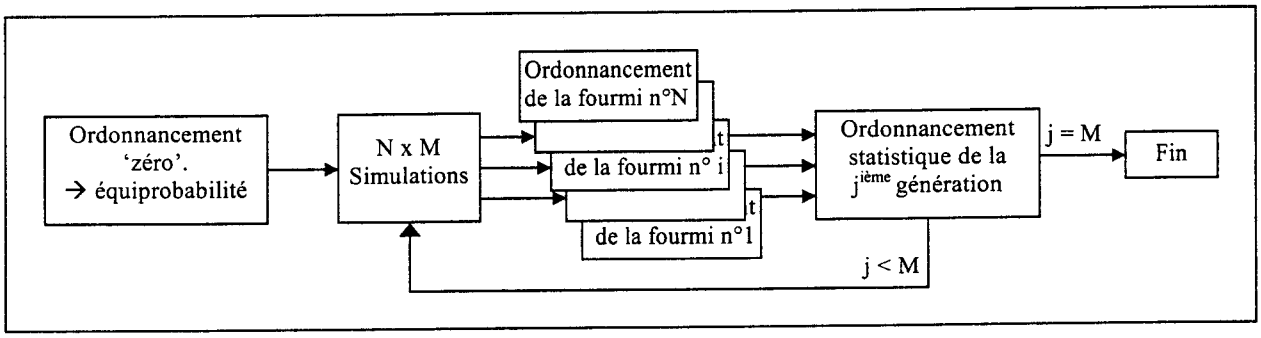


Figure 5 : Algorithme basé sur le principe des colonies de fourmis

Après le passage dans le simulateur de la dernière fourmi ($i=n$) de la $j^{ième}$ colonie (ou génération), on re-calcule d'après l'expérience de cette colonie, pour chaque instant, les probabilités d'affectation pour les fourmis de la colonie suivante ($j \leftarrow j+1$). Ces nouvelles probabilités sont ou non bruitées et prennent compte ou non l'héritage des colonies antérieures (génération *j-2, j-3*...). Le processus s'arrête après un nombre *M* de cycles arbitrairement fixé. Enfin, nous pouvons figer les probabilités du cycle zéro de façon équiprobable, mais nous avons également la possibilité de démarrer avec une matrice *Phero* déséquilibrée.

Nous présentons ci-après les procédures relatives au simulateur dédié aux colonies de fourmis, à l'élection des tâches potentiellement activables, à l'allocation réelle des ressources à ces tâches et enfin les procédures qui effectuent les tirages aléatoires.

4.2.1. Le simulateur.

Le principe consiste à retenir le meilleur ordonnancement de la meilleur fourmi de la meilleure génération (ou colonie). La qualité de cet ordonnancement est mesurée par le temps total d'exécution du plan de charge (*Makespan*). Dans cet algorithme, nous utiliserons deux grandeurs de type fitness, la première, *Best_fit_colonie*, est relative au meilleur ordonnancement d'une colonie, la seconde *Best_fit* caractérise le meilleur ordonnancement de toutes les colonies.

Début { Procédure Simulation }
  Best_fit := TL ; Nb_Cycle :=0 ;
  Initialisation de la matrice *Phero* (équiprobabilité éventuellement bruitée)
  Initialisation de la matrice *Copie_Phero* à *zéro*
  Tant Que *Nb_Cycle< Nb_Cycle_limite* **Faire**
    Nb_Cycle :=Nb_Cycle+1 ; Best_fit_colonie := TL ; k :=0 ;
    Tant Que k<Nb_fourmi **Faire**
      *k :=k+1 ; TU :=-1 ;*Initialisation du Diagramme de GANTT local, *Gantt_Loc ;*
      Tant Que *TU<=TL* **Faire**
        Incrémenter les tâches actives, Marquer[3] *Gantt_Loc*
        Désactiver les tâches désactivables[4]
        Etablir la liste des tâches[5] activables
        Pour chaque ressource **faire**
          Si plusieurs tâches demandent cette ressource
          Alors | Tirer[6] au sort une tâche activable
                    | Oter les autres de la listes des tâches activables
        Si nombre de ressources>nombre[7] de ressources tolérées
        Alors | Tirer[8] au sort les tâches correspondantes à ces ressources
                  | Oter les autres de la listes des tâches activables
        Consigner les résultats des tirages au sort dans la matrice *Copie_Phero*
        Activer les tâches restant dans la liste des tâches activables ;
      Fin Tant Que { Fin de simulation d'une fourmi }

Simulation d'une fourmi | Simulations d'une colonie de fourmis | Simulations de toutes les colonies de fourmis.

[3] Il s'agit de noter à chaque instant par quelle tâche est occupée chaque ressource.

[4] Tâche dont le temps d'activité en cours à dépasser sa durée normale d'exécution.

[5] Tâche pour lesquelles toutes les tâches directement antérieures sont terminées Cf. paragraphe 4.2.2.

[6] Avec la probabilité de la tâche pour la ressource à l'instant TU

[7] Le nombre d'encours tolérés est égal au nombre d'encours de la simulation moins le nombre de ressources déjà occupées Cf. paragraphe 4.2.2 .

[8] Avec la probabilité à l'instant TU de la ressource correspondant à la tâche. A ce stade, chaque ressource n'est plus demandée que par une seule tâche.

Simulations de toutes les colonies de fourmis.

```
    Si TU <Best_fit_colonie Alors Best_fit_colonie:=TU ; Best_Gantt := Gantt_Loc ;
  Fin Tant Que { Fin de simulation d'une colonie }

  Si Best_fit_colonie< Best_fit Alors Best_fit:= Best_fit_colonie
  { Mise à jour de la matrice Copie_Phero }
  Pour chaque instant t Faire
  Début
      Som_Conflit :=0 ;
      Pour Chaque ressource u Faire
          Som_Conflit :=Som_Conflit+Copie_Phero[0,u,t] ;
          Som_Demande :=0 ;
          Pour chaque tâche i Faire
              Som_Demande := Som_Demande +Copie_Phero[i, u, t] ;
          Copie_Phero[i, u, t] :=Copie_Phero[i, u, t] / Som_Demande ;
      Fin Pour
      Copie_Phero[0,u,t] :=Copie_Phero[0,u,t] / Som_Conflit ;
  Fin

  Si Bruit>0 Pour Chaque instant t Faire { Bruitage }
      Pour Chaque ressource u Faire
          Pour Chaque tâche i Faire
              Random ; som :=Bruit – Random(2*Bruit) ;
              Copie_Phero[i, u, t] :=trunc(Copie_Phero[i, u, t]*(1+som/100)) ;
           Fin Pour

  Si Héritage>0 Pour Chaque instant t Faire { Héritage }
      Pour Chaque ressource u Faire
          Pour Chaque tâche i Faire
              Copie_Phero[i, u, t] :=trunc(Héritage/100*Phero[i, u, t]+
                                      (1-Héritage/100)*Copie_Phero[i, u, t] ) ;
           Fin Pour

  Phero :=Copie_Phero ; { Transmission à la génération suivante }

Fin Tant Que { Fin de simulation de toutes les colonies }
Fin { Procédure simulation }
```

4.2.2. Allocation des ressources – Election et Sélection.

Dans les deux cas, nous construisons une liste de tâches candidates dans laquelle on effectuera un tirage, soit d'une seule tâche pour l'opération d'élection, soit d'une liste de tâches pour l'opération de sélection.

```
Début { Procédure d'élection d'une tâche activable }
    Pour Chaque tâche i Faire DT[8,i]:=0; { Autorisation d'allocation non accordée }
    Pour Chaque ressource u Faire
      Si Etat_R[u]=0 Alors{ Si la ressource u est libre }
        Si Dema_R[u]>=1 Alors   { Il y a plusieurs demande pour la ressource libre u  }
            Début { Constitution d'une liste de tirage }
                w:=1;
                Pour i de 1 à n Faire LI[i]:=0;
                Pour Chaque tâche i Faire
                Si ( (Nressource(i)=u) and (DT[6,i]=1) ) Alors
                { Si la ressource demandée par la tâche i est la ressource u ET que la tâche i
                est activable Alors on la met dans la liste des candidates à choisir, Sinon rien }
                Début
                    LI[w]:=i; w:=w+1;
                Fin
                { Tirage basé sur la matrices de phéromone de la tâche Num_T dans la liste
                construite LI, de tâches demandant u à l'instant TU }
                Tire_Tache( LI, u, TU, Num_T);
                { Autorisation temporairement accordée } DT[8,num_T]:=2;
            Fin
                        Sinon   { Une seule demande au plus de la ressource u }
            Pour i de 1 à n Faire
                Si ( (Nressource(i)=u) and (DT[6,i]=1) ) Alors DT[8,i]:=2;
Fin
```

Après l'exécution de cette procédure, les tâches élues sont associées à des ressources différentes. Chaque ressource n'est plus demandée que par une seule tâche. Il convient alors, si le nombre de ressources activables, augmenté du nombre des ressources déjà actives (ou encore actives), est supérieur au nombre toléré de ressources simultanément actives, de sélectionner les tâches associées au ressource présentant à cet instant la meilleure probabilité pour l'ordonnancement. C'est l'objet du second tirage.

```
Début { Procédure de sélection d'une liste de tâches }
    Xactuel:=0; { calcul du nombre de ressources déjà occupées }
    Pour Chaque ressource u Faire Xactuel:=Xactuel+Etat_R[u];
      Si Xr=Xactuel Alors | { pas d'allocation possible }
                          | Chaque tâche i Faire DT[8,i]:=0 ;
                  Sinon { allocation possible s'il y a des élus }
    Début
        som:=0;{ Calcul du nombre de tâches élues temporairement }
        Pour Chaque tâche i Faire Si DT[8,i]=2 Alors som:=som+1;
        Si (som>(XR – Xactuel)) Alors { Nbr. tâches candidates> Nbr. ressources tolérées }
        Début
            { Construction de la liste }w:=1; Initialise la liste LI à zéro ;
            Pour Chaque ressource u Faire Si Dema_R[u]>0 Alors
            Début { pour chaque ressource u, si elle est demandée}
                Pour Chaque tâche i Faire
                    Si ((DT[5,i]=u) et (DT[8,i]=2)) Alors { si i demande u et i
                    temporairement autorisée} Début LI[w]:=i; w:=w+1; Fin
                Tire_Ressource( som,(Xr- Xactuel), LI, u, TU, LIS);
                { mise à jour des autorisations réelles }
                Pour i de 0 à (Xr – Xactuel) Faire DT[8,LIS[i]]:=1;
                Pour Chaque tâche i Faire Si (DT[8,i]=1) Alors
                Début { affectation de la structure Copie_Phero }
                    Copie_Phero[i, NRessource(i),TU]:=
                    Copie_Phero[i, NRessource(i),TU]+1;
                    Copie_Phero[0, NRessource(i),TU]:=
                    Copie_Phero[0, NRessource(i),TU]+1;
                Fin
            Fin
        Fin
                                    Sinon { Il n'y a assez de candidats admissibles}
        Pour Chaque tâche i Faire Si DT[8,i]=2 Alors
        Début
            DT[8,i]:=1; { mise à jour des autorisations réelles }
            { affectation de la structure Copie_Phero }
            Copie_Phero[i, NRessource(i),TU]:=
            Copie_Phero[i, NRessource(i),TU]+1;
            Copie_Phero[0, NRessource(i),TU]:=
            Copie_Phero[0, NRessource(i),TU]+1;
        Fin
    Fin
Fin { Procédure de sélection }
```

4.2.3. Les procédures de tirage aléatoire.

La procédure *Tire_Tache* tire aléatoirement une tâche *Num_T* dans la liste *LI* en fonction de la probabilité d'attraction *Phero[Num_T, u, TU]* de *Num_T* pour la ressource *u* à l'instant *TU*. Ce tirage s'effectue suivant le principe de la roulette.

Le principe du tirage est exposé à travers un exemple présenté *Figure 6*. La liste *LI* est constituée par la procédure appelante (ici de 6 tâches candidates) et ne comporte pas forcément l'ensemble des composantes non nulles d'une colonne de la matrice *Phero*. En effet, pour l'ordonnancement donné, il est possible qu'à l'instant *TU*, certaines tâches qui avaient obtenu cette ressource au passage de la colonie précédente et au même instant *TU* ne soient plus candidates. C'est pourquoi nous devons à nouveau normaliser les probabilités des seules tâches éligibles, à cet instant et pour cette ressource. Ces probabilités normalisées sont stockées dans la table *Prob_locale*. Afin de se prémunir des éventuels défauts du générateur de nombre aléatoire la normalisation se fait à la valeur *1000*. Dans notre exemple, le nombre aléatoire tiré entre *0* et *1001* a pour valeur *590*, ce qui a pour conséquence d'élire le quatrième candidat à cette élection (sur cinq éligibles), c'est à dire la tâche numéro *5* de l'application (sur 9 éligibles par la colonie précédente).

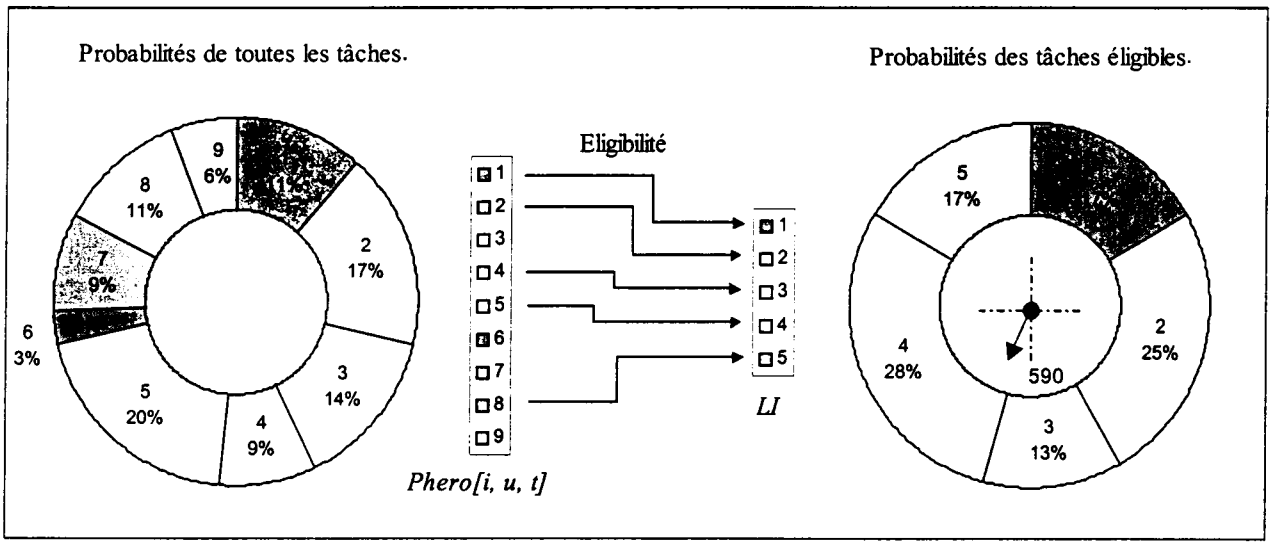


Figure 6 : Tirage suivant le principe de la roulette.

La procédure *Tire_Ressource* tire aléatoirement une liste *LIS* de *'ns'* tâches demandant les ressources *u* distinctes, parmi *'ne'* tâches contenues dans la liste d'entrée *LI* à l'instant *TU*. Ces *'ns'* tâches sont tirées également suivant le principe de la roulette et en fonction de la probabilité d'attrait pour l'ordonnancement *Phero[0, u, TU]* de la ressource *u* à l'instant *TU* .

```
Début { Procédure Tire_Tache }
    { Normalisation des probabilités concernées par le tirage }
    som :=0 ; w :=1 ; ∀u ∈ [1..n], Prob_locale[u]:=0;
    Tant Que LI[w]<>0 Faire
        som :=som +Phero[LI[w], u, TU]; w :=w+1 ;
    Fin Tant Que
    Pour i de 1 à w Faire Prob_locale[w] :=Phero[LI[w], u, TU] / som ;
    Initialisation du générateur aléatoire;
    ProbaTire:=trunc(Random(1000));
    v:=1;  ProbaCou:=1000*Prob_locale[v];
    Tant Que (ProbaTire>ProbaCou) and ( v<w ) Faire
        v:=v+1; ProbaCou:=ProbaCou + 1000*Prob_locale[v];
    Fin Tant Que
    Si ((v <=w) et ( ProbaTire<=ProbaCou)) Alors Num_T:=LI[v] Sinon Num_T:=0;
Fin { Procédure Tire_Tache }
```

```
Début { Procédure Tire_Ressource }
    { Normalisation des probabilités concernées par le tirage }
    som :=0 ; w :=1 ; ∀u ∈ [1..n], Prob_locale[u]:=0;
    Tant Que LI[w]<>0 Faire
        som :=som +Phero[0, DT[5, LI[w]], TU]; w :=w+1 ;
    Fin Tant Que
    Pour i de 1 à w Faire Prob_locale[w] :=Phero[0, DT[5,LI[w]], TU] / som ;
    Initialisation de LIS[i] à 0; y:=1; Probancien :=0 ;
    Tant Que y<ns  Faire
        Initialisation du générateur aléatoire;
        ProbaTire:=Random(1000-Probancien);
        v:=1; ProbaCou:=1000*Prob_locale[v];
        Tant Que ( ProbaTire>ProbaCou) and ( v<w ) Faire
            v:=v+1; ProbaCou:=ProbaCou + 1000*Prob_locale[v];
        Fin Tant Que
        Si  v<=w  alors LIS[y]:=LI[v] sinon LIS[y]:=0;
        y:=y+1; Probancien :=1000*Prob_locale[v] ;
        { on retire la tâche définitivement élue }
        Pour i de v à ne-1 Faire LI[v]:=LI[v+1];Prob_locale[v] :=Prob_locale[v+1] ;
        ne:=ne-1;
    Fin Tant Que
Fin { Procédure Tire_Ressource }
```

Cette deuxième procédure effectue autant de tirages qu'il y a de tâches à sélectionner. Nous normalisons dans la table *Prob_locale* les probabilités *Phero[0, u, TU]* des ressources *u* associées aux tâches précédemment contenues dans la liste d'entrée *LI* à l'instant *TU*.

Nous effectuons un premier tirage entre 0 et 1000, nous retirons alors la tâche sélectionnée et nous calculons la nouvelle borne maximale du prochain tirage. Celle-ci est égale à l'ancienne diminuée de la probabilité de la tâche qui vient d'être choisie. Sur l'exemple représenté *Figure 7*, la procédure *Tire_ressource* a permis de sélectionner les tâche numéro *1* et *2*.

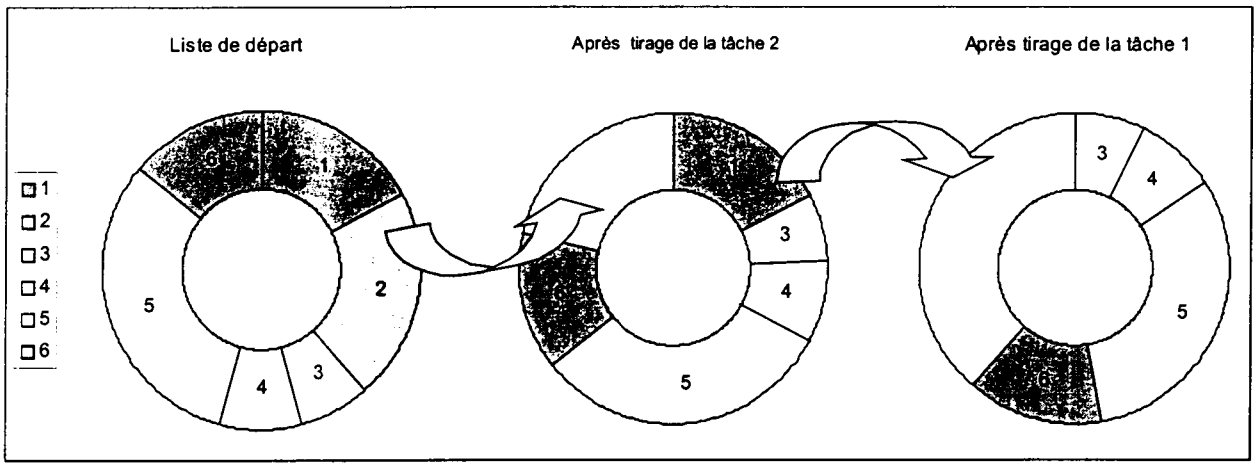


Figure 7 : Sélection de la sous liste des tâches 1 & 2.

Nous avons enfin vérifié ci-après le fonctionnement du générateur aléatoire fourni par l'environnement DELPHI. L'écart type de la distribution est d'environ 3%.

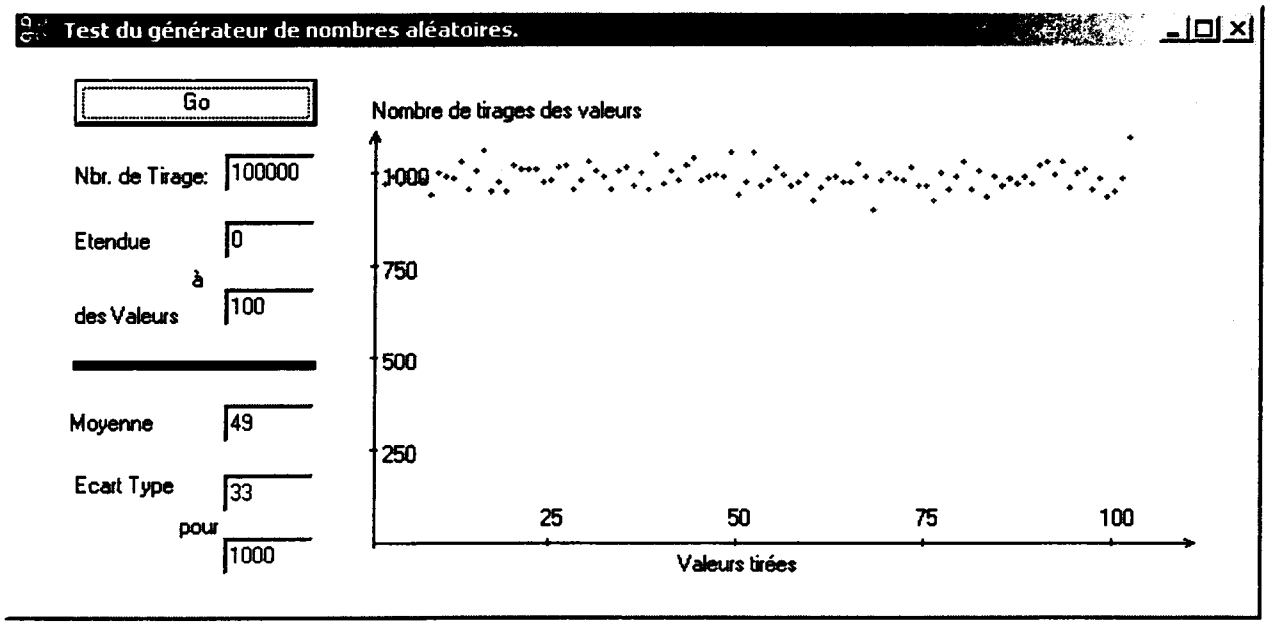
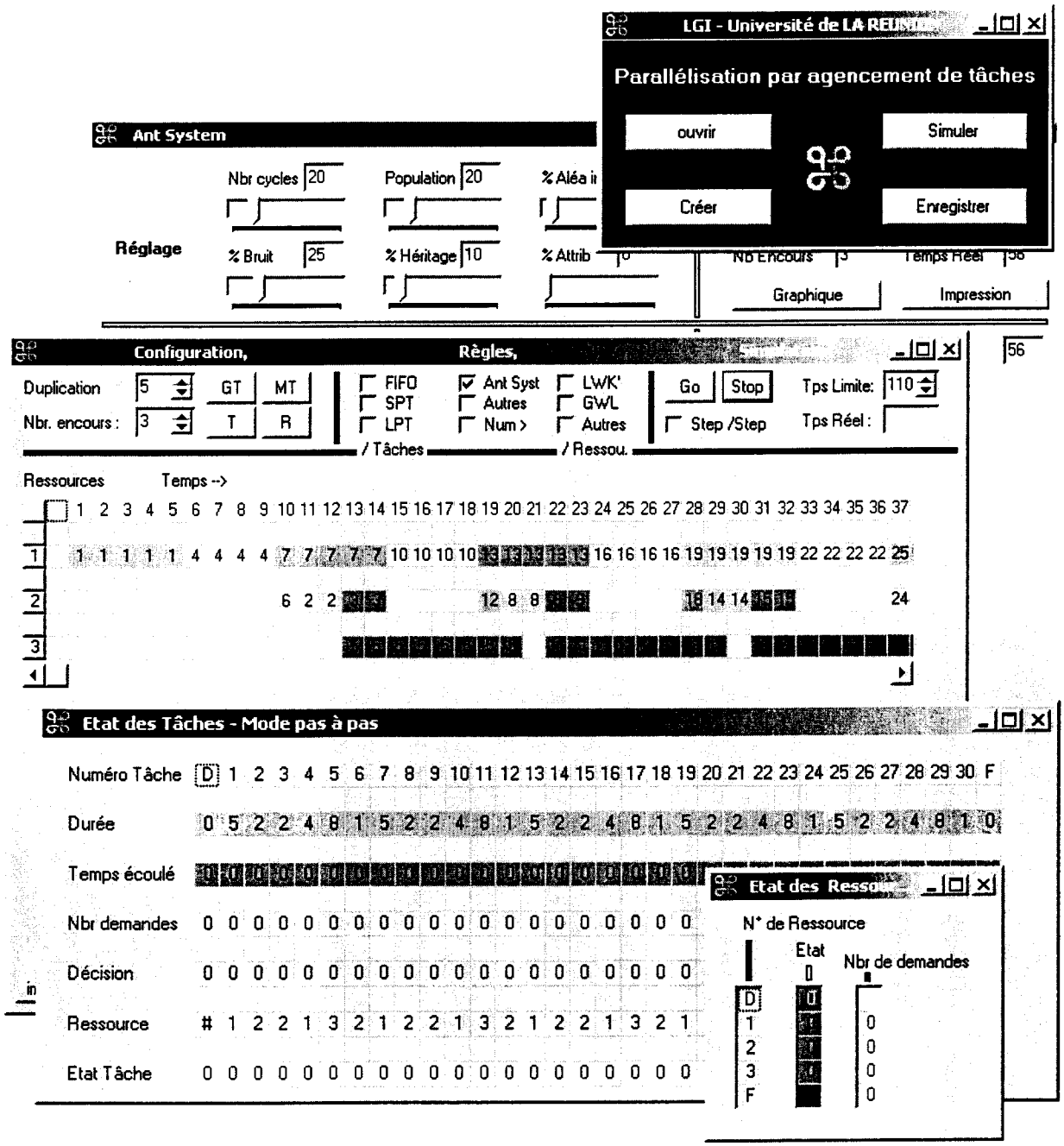


Figure 8 :*Résultats du test du générateur aléatoire.*

5. Copies d'écran.



CONCEPTION DE LA COMMANDE D'UN SYSTEME AUTOMATISE DE PRODUCTION : APPORT DES GRAPHES ET DE L'ORDONNANCEMENT CYCLIQUE.

Résumé :
Les travaux exposés dans ce mémoire portent sur la conception préliminaire de la commande des systèmes automatisés de production, l'objectif étant d'en améliorer la productivité et la flexibilité. La stratégie retenue consiste à définir une commande idéale et spécifique pc chaque type de produit fabriqué par le système automatisé. Nous définissons un formalis noté GMT (Graphe Marqué et Temporisé) dans lequel nous représentons la gamme opératoire de chaque production. Nous développons ensuite une méthode capable d'ordonnancer cycliquement des gammes non linéaires, prenant par exemple en compte des opérations d'assemblage ou de désassemblage. Cette méthode est basée sur un simulateur qui exécute le plan de production en arbitrant les conflits d'affectation selon le comportement social des colonies de fourmis. Il résulte de cette simulation un diagramme de GANTT duquel nous extrayons le graphe GMT de commande de l'application. Les outils conventionnels de la théorie de graphes permettent alors d'analyser l'architecture et de vérifier les propriétés de célérité et de vivacité du graphe obtenu. Finalement, une passerelle GMT/Grafcet produit le Grafcet de commande de façon systématique.
Ces travaux se terminent par l'étude d'un cas de MPS variable (Minima Part Set), montrant ainsi le bien fondé de notre démarche.

Mots-clés : Système automatisé de production, productivité, flexibilité, conception de la commande, graphe, Grafcet, ordonnancement cyclique, algorithme des fourmis.

CONTROL DESIGN OF AUTOMATED PRODUCTION EQUIPMENT: CONTRIBUTION OF THE GRAPH AND THE CYCLIC SCHEDULING.

Abstract:
The results presented in this work are focussed on the preliminary control design of the automated production equipment, with the main objective to improve the productivity and flexibility. The selected strategy consists in defining a specific and ideal command for each type of product manufactured by the automated system. We define a formalism noted GMT (Marked and Temporised Graph) in which we represent the operational steps of each production. We then develop a method for cyclic scheduling of non-linear operating sequence, by considering for example assembling and dismantling operations. This method is based on a simulator, which executes the production plan, by arbitrating the conflicts in the attribution following the social behaviour of ant colonies. It results from this simulation a GANTT diagram, from which we extract the GMT application command graph. The conventional graph theory tools then allow to analyse the architecture and to verify the celerity properties and the vivacity of the graph obtained. Finally, a GMT/Grafcet platform (link) generates systematically the Grafcet command.
This work is completed by a case study of MPS variable (Minimal Part Set), thus pointing out the validity of our approach.

Key words: Automated production system, productivity, flexibility, control design, graph, Grafcet, cyclic scheduling, ant algorithm (system).