Analyse et conception des systèmes temps-réel : translation d'une approche fonctionnelle à une approche orientée objet

Adel Benzina

N° ordre: 2815

Année 1997

Thèse

Présentée devant
L'Université Paul Sabatier (Sciences)

en vue de l'obtention du titre de
Docteur de l'Université Paul Sabatier de Toulouse

Spécialité : **Informatique Industrielle**

par
Adel BENZINA
Ingénieur Génie Électrique ENIT - Tunis

Analyse et Conception des Systèmes Temps Réel : Translation d'une Approche Fonctionnelle à une Approche Orientée Objet

Soutenue le 16 décembre 1997, devant le jury :

Président :	**R. VALETTE**	Directeur de recherche CNRS
Rapporteurs :	**J. P. BOUREY**	Professeur des universités
	P. ESTRAILLIER	Professeur des universités
	J. J. SCHWARZ	Professeur des universités
Examinateurs :	**M. COURVOISIER**	Professeur des universités
	J. PROUST	Chef de service, PSA Peugeot-Citroën
Directeur de thèse :	**M. PALUDETTO**	Maître de conférences

Rapport LAAS N° 97501
Thèse préparée au Laboratoire d'Analyse
et d'Architecture des Systèmes du CNRS.

7, avenue du Colonel Roche
31 077 Toulouse Cedex 4.

Avant-Propos

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Analyse et d'Architecture des Systèmes (L.A.A.S) du C.N.R.S au sein du groupe Décision et Conduite dans les Systèmes de Production (D.C.S.P). A ce titre, je tiens à remercier Messieurs Alain Costes et Jean Claude Laprie, directeurs successifs du L.A.A.S, ainsi que Monsieur Jean Claude Hennet responsable du groupe D.C.S.P pour m'avoir accueilli et m'avoir ainsi permis d'effectuer mon travail de recherche.

Je remercie également Monsieur Robert Valette, Directeur de Recherche au C.N.R.S, pour ses remarques, ses conseils et pour avoir accepté de présider mon jury de soutenance.

Je tiens tout particulièrement à exprimer ma profonde gratitude à mon directeur de thèse, Monsieur Mario Paludetto, Maître de conférences à l'Université Paul Sabatier de Toulouse, pour sa confiance, son soutien et ses idées. Que ce mémoire soit le témoignage de ma profonde reconnaissance.

Je suis également reconnaissant à Messieurs Jean Pierre Bourey, Professeur à l'École Centrale de Lille, Pascal Estraillier Professeur à l'Université de Paris VI et Jean Jacques Schwarz Professeur à l'IUT A de l'Université Claude Bernard de Lyon, pour l'intérêt qu'ils ont accordé à mon travail et pour avoir accepté la lourde tâche d'en être les rapporteurs.

Je tiens également à exprimer ma gratitude à Messieurs Marc Courvoisier Professeur à l'Université Paul Sabatier de Toulouse et Directeur de l'Institut National des Sciences Appliquées de Toulouse ainsi que Monsieur Jacques Proust Chef du Service Mécatronique et Sûreté de Fonctionnement à la société PSA Peugeot-Citroën pour leurs remarques, leurs conseils et pour avoir accepté de prendre part à mon jury de soutenance.

Je remercie chaleureusement l'ensemble des membres du groupe D.C.S.P pour l'aide et la sympathie dont j'ai pu bénéficier tout au long de mon séjour au LAAS ainsi que Madame Eliane Dufour pour son efficacité et sa gentillesse.

Mes remerciements s'adressent également à tous les membres des services informatique, administratif, documentation, édition..., ceux qui, par leur efficacité, font que le LAAS est ce qu'il est.

Je ne saurais oublier tous mes amis, du LAAS et d'ailleurs, qui m'ont aidé et supporté tout au long de mon travail. Je pense particulièrement à Eric Zamaï, Nabil Ben Khalifa, Ahmed Braham, Dhia Ben Letaifa, Jerôme Delatour, Isabelle Bazet, Christian Berty, Cyril Briand, Gilles Moncelet, Laurent Gallon, Isabelle Blum, Thierry Gayraud, Slim Abdellatif, Mohamed Jarraya, Loubna Yacoubi, Meriem Chater, Slim Khalbous, Mohamed El Yacobi, Khalil Drira, François Vernadat...

Enfin, je dédie ce travail à mes parents pour leur dévouement et pour leur soutien permanent.

Table des matières

Abréviations

ASER	ASynchronous Execution Request
CASE Tool	Computer Aided Software Engineering Tool
CFD	Control Flow Diagram
Cspec	Control Specification
COO	Conception Orientée Objet
DFD	Data Flow Diagram ou Diagramme de Flots de Données
DFC	Diagramme de flots de Contrôle
FDFD	Formal Data Flow Diagram
HOOD	Hierarchical Object Oriented Design
HOOD/PNO	Hierarchical Object Oriented Design with Petri Net Objects
HSER	Highly Synchronous Execution Request
LOO	Langages Orientés Objet
LSER	Loosely Synchronous Execution Request
OCS	Object Class Specification (dans HOOD/PNO)
ODS	Object Description Skeleton (dans HOOD/PNO)
OMT	Object Modeling Technique
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OODA	Object Oriented Domain Analysis
OOSE	Object Oriented Software Engineering
PNOBCS	Petri Net OBject Control Structure
PNOBCS	Petri Net OPeration Control Structure
Pspec	Process Specification
RdP	Réseaux de Petri
RdPTS	Réseaux de Petri Temporisés Stochastiques
RTOS	Real Time Operating Systems
SA	Structured Analysis
SADT	Structured Analysis and Design Technique
SA-RT	Structured Analysis - Real Time
SASS	Structured Analysis and Systems Specification
SD	Structured Design
SDL	Specification and Description Language
TOER	Time-Out Execution Request
UML	Unified Modeling Language

Introduction

Le logiciel apparaît aujourd'hui comme un composant fondamental des systèmes automatiques. Il est à la base de la coordination des activités de tout le système, pour l'amener à accomplir ses fonctions (production, conduite, surveillance...). Pour les systèmes temps réel, la tâche de ce composant est d'autant plus critique qu'il est contraint par le procédé contrôlé tout en étant obligé de satisfaire les exigences de l'utilisateur.

Ainsi, les tâches de plus en plus complexes dévolues au logiciel exigent des méthodes et des outils appropriés. La recherche dans le domaine du génie logiciel doit apporter les solutions adéquates aux problèmes posés par les systèmes temps réel. Faisant partie des axes principaux de cette recherche, la mise au point de méthodologies pour l'analyse et la conception des systèmes temps réel constitue un domaine où l'évolution a été très rapide.

En effet, les méthodologies d'analyse et de conception des systèmes temps réel ont été marquées par plusieurs évolutions. L'une des plus significatives a conduit à l'apparition des concepts orientés objet, issus des recherches sur les langages orientés objet. Parmi ces concepts, notons particulièrement l'encapsulation, le masquage de l'information et la localisation. Plus qu'une évolution dans les outils, ces concepts constituent une évolution dans le mode de pensée de l'analyste. Ils amènent plusieurs avantages tant sur le plan de la qualité du logiciel réalisé que sur le processus de développement. D'une part, ils améliorent les critères de qualité du logiciel comme la réutilisabilité, l'indépendance spatiale et temporelle. D'autre part, au niveau du processus de développement, ces concepts améliorent la trace des besoins et facilitent la maintenance du logiciel et ses éventuelles modifications et extensions. Plusieurs méthodologies basées sur ces concepts ont été mises au point. Elles couvrent plus ou moins toutes les étapes du cycle de vie du logiciel et sont différemment adaptées aux divers problèmes du génie logiciel (gestion de bases de données, systèmes temps réel...).

Cependant, comme pour la plupart des évolutions, l'apparition des méthodologies orientées objet pose un problème de transition pour les personnes habituées aux méthodologies fonctionnelles, plus traditionnelles. Fortes de leur passé, les méthodologies fonctionnelles (ou structurées) ne sont pas toujours dénuées d'intérêt. Elles adoptent des principes de structuration intuitifs, basés sur la décomposition des fonctions, qui leur ont valu une large diffusion auprès des concepteurs de logiciels. Parmi ces méthodologies, la méthode SA-RT est particulièrement estimée par un grand nombre d'industriels. Le rapport [Wood et Wood 89] indique qu'en 1989, plusieurs années après l'émergence des approches orientées objet, plus du tiers des industriels américains utilisent la méthode SA-RT.

Ainsi, malgré les avantages qu'elle offre, l'adoption des approches orientées objet n'est pas systématique. Cette réticence est due principalement à l'expérience acquise dans le domaine des méthodes fonctionnelles et aux réalisations accumulées grâce à ces méthodes. Il est difficile d'abandonner des acquis. D'autre part, la complexité technique des approches orientées objet, encore du domaine de la recherche pour le temps réel, leur diversité et les investissements nécessaires à une éventuelle reconversion (formation, outils...) sont perçus comme des handicaps supplémentaires.

Il y a donc une phase de transition à franchir au cours de laquelle il serait intéressant de donner aux concepteurs le moyen d'effectuer une analyse des besoins selon une approche fonctionnelle et de continuer les autres étapes du cycle de vie selon une approche orientée objet. Une méthode de translation d'une approche fonctionnelle vers une approche orientée objet permettrait de résoudre les problèmes qui se posent lors de cette phase de transition.

La possibilité de passer d'une approche à l'autre, notamment en phase d'analyse des besoins, a fait l'objet de plusieurs recherches. Nous nous associons à ceux qui ont cherché des solutions à ce problème en proposant une méthode de passage d'une spécification fonctionnelle vers une spécification orientée objet, basée sur l'utilisation des propriétés structurelles des réseaux de Petri. Ce mémoire est consacré à la description de cette méthode et des concepts qu'elle utilise, il est composé de cinq chapitres.

Le premier chapitre expose brièvement les concepts fondamentaux du génie logiciel pour les systèmes temps réel, les caractéristiques des méthodologies utilisées ainsi que deux méthodologies particulièrement adaptées aux systèmes temps réel. La première est une méthodologie fonctionnelle associant SA-RT et les réseaux de Petri. La seconde est une méthodologie orientée objet HOOD/PNO. Enfin, l'utilité d'une migration entre les deux types de méthodologies est mise en avant.

La polémique que suscite ce passage est évoquée dans le second chapitre. Sont présentées également les incompatibilités entre les deux types d'approches et une étude bibliographique sur les solutions proposées pour ce problème. L'analyse de ces solutions permet de préciser la nature des problèmes posés et les propositions permettant de les résoudre.

La solution adoptée consiste à utiliser les réseaux de Petri pour effectuer le passage entre SA-RT et HOOD/PNO. Le troisième chapitre propose les règles de description

des spécifications SA-RT par les réseaux de Petri, ainsi que les divers problèmes que pose cette tâche.

L'exploitation des réseaux de Petri obtenus pour la décomposition des spécifications fonctionnelles en spécifications orientées objets fait l'objet du quatrième chapitre. Trois types d'approches sont proposées. La première exploite la description statique, la seconde s'appuie sur la description dynamique et la troisième est une approche mixte. Les divers aspects de la méthode ainsi que ses limites sont également étudiés dans ce chapitre.

Enfin, la représentation des spécification par réseaux de Petri nous a amené à étudier l'apport de ce modèle pour la validation et l'évaluation des spécifications. Cet aspect est présenté dans le cinquième chapitre.

Tout au long du mémoire, les divers aspects étudiés sont illustrés par des exemples dont le choix a été particulièrement soigné afin qu'ils soient assez complets pour être représentatifs tout en relativisant leur volume et leur complexité.

Chapitre I

Le génie logiciel pour les systèmes temps réel

Le génie logiciel temps réel est le résultat de la combinaison des impératifs découlant du domaine du génie logiciel avec les contraintes imposées par les systèmes temps réel. Plusieurs méthodologies ont été mises au point pour assister les développeurs de logiciels. Dans ce chapitre introductif, nous exposons brièvement les concepts de base du génie logiciel, les particularités des systèmes temps réel. Nous donnons également un aperçu sur les méthodologies d'analyse et de conception pour les systèmes temps réel et nous exposons deux méthodologies particulières : SA-RT (Structured Analysis and Real-Time) et HOOD/PNO (Hierarchical Object Oriented Design with Petri Nets Object). Enfin, nous exposons les motivations de notre travail sur le passage entre analyse fonctionnelle et analyse orientée objet.

I.1 Le génie logiciel

Le génie logiciel est l'art de réaliser des logiciels. Jusqu'au début des années soixante-dix, la réalisation de logiciel se limitait au codage (ou programmation). Les activités d'analyse et de conception se faisaient au fur et à mesure des besoins du codage. Elles dérivaient souvent et inconsciemment vers une solution plus basée sur les possibilités ou les performances de la machine que sur les besoins réels du client. Cette situation a abouti à une crise dans le domaine du génie logiciel. A la fin des années soixante-dix, un constat d'échec a été publié par un groupement de chercheurs [GAO 79] : 45% des dépenses en génie logiciel allaient à des logiciels jamais utilisés avec succès. En

outre, étant donné la complexité croissante des tâches allouées aux logiciels, il n'était plus concevable de continuer à les développer d'une façon artisanale. Les activités nécessaires à la réalisation des logiciels ont été restructurées et des méthodologies de conception ont commencé à voir le jour.

I.1.1 Le cycle de vie du logiciel

La réalisation d'un logiciel ne se résume pas uniquement à son codage. Elle passe par un ensemble d'étapes qui constituent son cycle de vie. Ce cycle commence par les étapes d'établissement du cahier des charges et s'étend jusqu'à l'étape de maintenance du logiciel. Les étapes d'analyse des besoins, de conception, d'implémentation et de tests sont particulièrement importantes. Elles ne sont pas parfaitement délimitées dans le temps et dans la nature des tâches qu'elles couvrent. Il peut exister un certain empiètement des unes sur les autres. Par ailleurs, l'effort que nécessite chacune de ces étapes diffère d'un projet à l'autre.

Globalement, les quatre étapes principales se succèdent dans le temps. Cette succession et le recouvrement qui peut en découler sont illustrés par le graphique de la figure *I.1* [Hatley et Pirbhai 87].

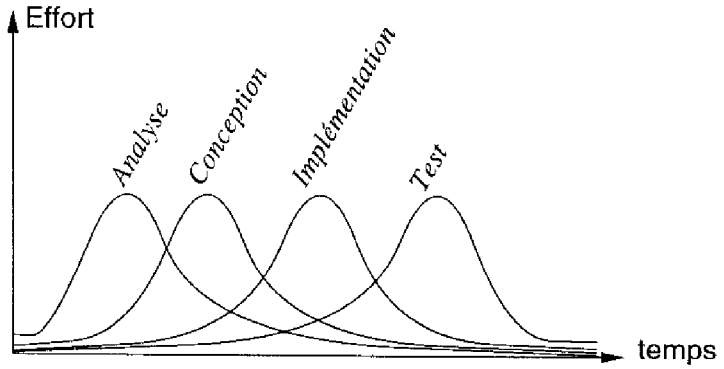


fig I.1 – Distribution de l'effort pendant le cycle de vie du logiciel

Plusieurs modèles ont été établis pour positionner les différentes étapes du cycle de vie du logiciel et pour clarifier les interactions entre elles. Ces modèles peuvent être classés en modèles séquentiels, itératifs et récursifs. Parmi les plus connus, citons le modèle en cascade itérative [Royce 70], le cycle en V (qui intègre le développement du logiciel dans un cycle plus global qui est le développement du système), le modèle en spirale [Boehm *et al.* 84], le standard américain 2167A du département de la défense [DoD 88]. Notons aussi le modèle récursif, *analyser un peu, concevoir un peu, implémenter un peu, tester un peu*, qui s'applique bien aux approches par abstraction. Par exemple, le modèle en V de la figure *I.2* permet de bien situer les différentes étapes du cycle de vie.

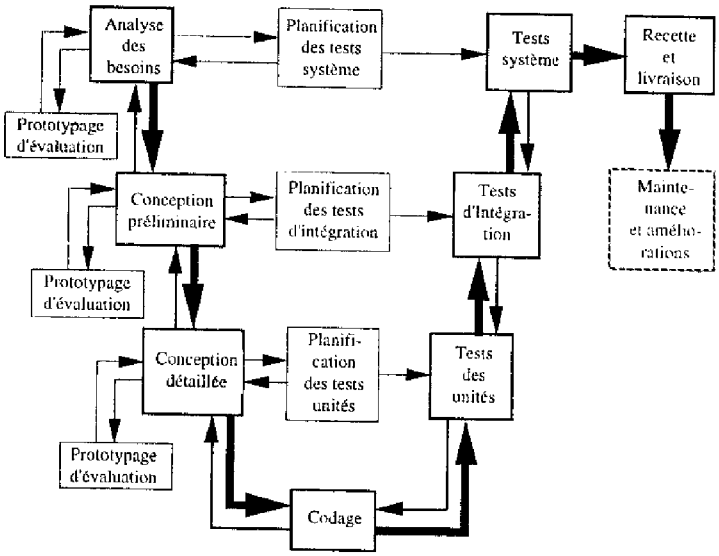


fig I.2 – Modèle en V du cycle de vie du logiciel

I.1.2 Importance de l'étape d'analyse des besoins dans le cycle de vie du logiciel

L'étape d'analyse des besoins a des conséquences directes et cruciales sur la qualité du logiciel développé. Il est difficile de délimiter exactement la tâche de l'analyste lors de cette étape, mais il est commun de dire que, lors de cette phase, l'analyste doit spécifier *ce que doit faire le logiciel* (par opposition au "*comment le faire*" qui est traité dans la phase de conception). Cette étape doit être menée en collaboration entre l'analyste et le demandeur de logiciel. A l'issue de cette étape, l'analyste doit avoir une compréhension maximale du problème et des besoins émis par le demandeur du logiciel.

Une bonne analyse des besoins permet de faciliter le travail du concepteur, d'éviter les malentendus et les mauvaises surprises à l'utilisateur, de réduire la complexité du projet et d'améliorer la qualité du logiciel. La phase d'analyse des besoins aboutit à l'établissement de plusieurs documents contractuels entre le demandeur et le réalisateur. Parmi ces documents, ceux qui intéressent l'analyste-concepteur sont les documents de spécification du logiciel demandé.

L'influence de l'étape d'analyse des besoins sur le reste du cycle de vie est d'autant plus grande qu'elle engage le concepteur et le client sur tout le reste du projet. Par ailleurs, c'est aux spécifications auxquelles aboutit cette phase que se réfèrent tous les intervenants sur le logiciel.

Ainsi, il est primordial de veiller à effectuer une bonne analyse des besoins. Plusieurs méthodologies ont été développées pour assister l'analyste lors de cette étape. Parmi ces méthodologies certaines conviennent mieux aux spécificités des systèmes qui nous intéressent : les systèmes temps réel.

I.2 Les systèmes temps réel

I.2.1 Définitions

Il est difficile de caractériser les systèmes temps réel en faisant abstraction du contexte de travail. Le qualificatif *temps réel* est utilisé, à tort ou à raison, dans divers domaines et dans de nombreuses applications.

La définition donnée par le groupe de réflexion sur le temps réel du CNRS semble être la plus proche des systèmes temps réel tels que nous les concevons : *Peut être qualifiée de «temps réel» toute application mettant en œuvre un système informatique dont le fonctionnement est assujetti à l'évolution dynamique de l'état d'un environnement (appelé aussi procédé) qui lui est connecté et dont il doit contrôler le comportement.*

Pour suivre et piloter l'état du procédé, le système informatique de contrôle doit alors généralement contenir un ensemble d'actions (tâches) dont les instants de début d'exécution sont, directement ou indirectement, assujettis aux signaux émis (événements) ou prélevés (mesures) sur ce procédé, et dont les instants de terminaison d'exécution sont soumis à des contraintes temporelles imposées par le procédé [Elloy *et al.* 88].

Pour certains, le «label» temps réel n'est accordé à un système que si ses exigences temporelles sont très contraignantes (inférieures à la milliseconde). Pour d'autres, il suffit qu'un système soit temporellement contraint (même de l'ordre de la minute) pour être qualifié de temps réel. Ainsi, une classification communément utilisée parle de systèmes temps réel «dur» (*hard real-time systems*) et de systèmes temps réels «mou» (*soft real-time systems*).

I.2.2 Spécificités des systèmes temps réel

Trois spécificités des systèmes temps réel découlent de ces définitions : la réactivité, les problèmes de parallélisme et de synchronisation et la prise en compte du temps.

Un système temps réel met en interaction le système informatique de commande et le procédé au moyens de capteurs et d'actionneurs. Il peut être qualifié de réactif au sens de Harel [Harel et Pnueli 85] : il doit réagir aux stimuli de l'environnement pouvant arriver à des instants aléatoires et il doit être capable de les prendre tous en compte.

La réactivité est une caractéristique qui oblige le système à traiter parallèlement (au moins au niveau de la spécification) les différents stimuli de l'environnement, tout en respectant les contraintes de temps. Ce parallélisme s'accompagne de la nécessité de communications (synchrones et/ou asynchrones) entre les tâches. Cependant, les systèmes temps réels sont plus contraints que les systèmes présentant uniquement du parallélisme. En effet, un système purement parallèle peut être satisfaisant indépendamment de la cadence d'exécution (horloge) du logiciel alors que la cohérence d'un système temps réel l'est fortement. Toutefois, notons qu'il est parfois possible de satisfaire des besoins de parallélisme par des implantations séquentielles. Ceci est le fonde-

ment même d'un des domaines de recherche sur le temps réel : l'ordonnancement des tâches.

Enfin, le problème le plus important posé par les systèmes temps réel est la prise en compte du temps. Le système de commande doit réagir aux événements du procédé dans les temps impartis. Ce temps est relatif à la dynamique du procédé : le temps de réponse exigé pour une variation de température dans un système de surveillance de température sur un site nucléaire peut être de l'ordre du millième de seconde, alors que pour un système de surveillance de température d'un four sidérurgique il peut être de l'ordre de la seconde [Benzina et Paludetto 96a].

Dans un système temps réel, le temps ne doit pas être vu comme un facteur de qualité ou de performance. Par exemple, dans un système bancaire la rapidité augmente le confort d'utilisation. Alors que dans un système temps réel, le temps est un facteur critique dont dépend la survie du système et même celle des personnes environnantes (systèmes avioniques, contrôle de procédés chimiques ou nucléaires...).

Cette notion du temps peut être appréhendée par le système de contrôle de deux façons différentes [Elloy *et al.* 88] :

- Le système de commande observe le procédé à des instants discrets pré-définis. C'est une perception du temps qui permet de définir rigoureusement les concepts de simultanéité, de synchronisation, d'exclusion, etc; approche rigoureuse qui permet de prévoir un ordonnancement des actions de façon tout aussi rigoureuse. Dans ce cas, il s'agit d'une maîtrise du *temps logique*.

- Le système de commande observe de façon continue le procédé. Par exemple, le concept de simultanéité repose sur une datation des événements (fournie par des horloges) ou doit être remplacé par des relations d'ordre ou de causalité entre les événements. Les actions de pilotage sont alors exécutées en fonction de leurs durées et de leurs échéances. Il s'agit ici d'une maîtrise du *temps physique*.

I.2.3 Domaines de recherche sur le temps réel

Les spécificités des systèmes temps réel doivent être prises en compte dans toutes les phases du cycle de vie du logiciel temps réel. L'implication de ces spécificités sur les différentes phases de développement du logiciel définit plusieurs axes de recherche :

- Spécification des applications temps réel.
- Évaluation des caractéristiques temporelles.
- Tolérance aux fautes matérielles et logicielles.
- Évaluation des performances.
- Élaboration d'architectures matérielles.

- Mise en œuvre dans des environnements centralisés ou distribués.
- Problèmes de communications (réseaux, protocoles).
- Systèmes d'exploitation temps réel (RTOS Real Time Operating Systems).
- Ordonnancement de tâches.
- Langages d'implémentation.
- Bases de données distribuées temps réel.

L'importance relative des domaines de recherche cités ci-dessus est différente d'une classe à l'autre. Par exemple, pour les systèmes temps réel dur, la recherche sur les architectures matérielles prend une importance capitale, alors que pour les systèmes temps réel mou, cet axe est relativement moins important.

Une introduction plus complète aux divers problèmes posés par les systèmes temps réel peut être trouvée dans [Elloy *et al.* 88], [Stankovic *et al.* 96], [Laplante 92], [Stankovic et Ramamritham 88].

Parmi les divers domaines de recherche concernant les systèmes temps réel, notre travail concerne principalement le problème de spécification. En fait, il s'agit de trouver des méthodologies efficaces pour l'analyse des besoins et des outils de spécification capables de prendre en compte les particularités de ces systèmes. Les méthodes de génie logiciel ne sont pas toujours adaptées à leurs exigences : représentation du parallélisme, de la synchronisation, des contraintes temporelles implicites et explicites, structuration efficace des modules logiciels, disponibilité d'outils d'analyse des spécifications, etc.

I.3 Méthodologies d'analyse et de conception des systèmes temps réel

Étant donné la complexité croissante des systèmes et l'apparition de contraintes temps réel de plus en plus sophistiquées, plusieurs méthodologies ont été mises au point pour l'analyse et la conception des systèmes temps réel. Ces méthodologies constituent des guides pour l'analyste-concepteur à travers les phases du cycle de vie du logiciel. Les diverses méthodologies peuvent être différenciées selon plusieurs critères. Parmi ces critères citons, les étapes qu'elles couvrent, l'étude d'aspects particuliers à certains domaines, la difficulté d'utilisation, la disponibilité d'outils informatiques d'aide à la conception (Case tools) supportant ces méthodologies ou l'importance accordée par les organismes gros consommateurs de logiciels temps réel (agences nationales de défense, d'aéronautique, programmes spatiaux, programmes de recherche internationaux,...). Ces critères font partie également des causes de succès ou d'échec des diverses méthodologies.

Le développement de méthodologies a pris un essor considérable ces dernières années. Elles se comptent par centaines. Cependant, les concepts de base utilisés ne sont

pas aussi nombreux. Parmi les concepts qui ont influencé la mise au point des méthodologies temps réel citons les concepts de structuration en modules [Parnas 72], les diagrammes de flots de données [DeMarco 78], les travaux du DoD (département de défense des USA) sur l'organisation du domaine génie logiciel temps réel et sur le langage de programmation ADA, les concepts orientés objet [Booch 86], les use-cases [Jacobson *et al.* 92], l'utilisation des réseaux de Petri en génie logiciel et tous les travaux de validation en résultant, etc. Les nombreuses méthodologies existantes adoptent des symbolismes différents mais, au fond, elles n'apportent pas toutes des innovations au niveau des concepts de base. Plusieurs méthodologies adaptent des concepts déjà connus à un domaine de recherche particulier ou à un type de projet particulier.

Les divers outils, méthodes et méthodologies peuvent être classés de diverses manières :

- **Méthodologies formelles ou informelles :**

  Les méthodologies formelles sont accompagnées d'outils d'analyse et de validation formels, cependant, elles sont généralement rigides à l'utilisation et manquent de techniques de structuration qui leur permettent de fournir une vue globale nécessaire pour les projets volumineux. Ex : Z, B, VDM, SDL, etc.

  Les méthodologies informelles (ou semi-formelles pour certaines d'entre elles) sont généralement accompagnées d'outils graphiques qui permettent de faciliter leur utilisation. Elles peuvent fournir des techniques de structuration capables de faciliter les projets complexes et ne nécessitent pas une grande qualification technique pour leur utilisation. En revanche, la validation des résultats n'est généralement pas aisée. Dans certains cas, les techniques de simulation permettent d'avoir certaines possibilités de validation. Ex : SADT, SA-RT, OMT, etc.

- **Méthodologies orientées donnée ou orientées contrôle :**

  Certaines méthodologies, comme celles utilisées dans les systèmes d'information, sont totalement orientées donnée. Elles permettent de structurer les données utilisées et les relations entre elles (comme les diagrammes entités-associations [Chen 76]). D'autres, appliquées aux systèmes à événements discrets, par exemple, sont plutôt orientées contrôle, elles représentent principalement la dynamique des systèmes (comme les réseaux de Petri et les state charts).

  Cependant, pour les systèmes temps réel il est nécessaire d'avoir une méthodologie capable de représenter les aspects dynamiques liés à l'évolution de l'état du procédé et de l'état interne du système de commande et, en même temps, de structurer la partie logicielle et de représenter les données relatives au système étudié.

  Généralement, les méthodologies utilisées pour les systèmes temps réel réalisent un compromis entre la part consacrée à la description du contrôle et celle consacrée à la description des données. Un outil qui privilégie un aspect au détriment de l'autre est nécessairement limité face à la complexité des systèmes temps réel. Ainsi, il est nécessaire de combiner, au moins, deux types de techniques pour aboutir à une méthodologie efficace.

- **Méthodologies fonctionnelles ou orientées objet :**

  Parmi les plus importantes évolutions du génie logiciel nous avons noté l'adoption des concepts orientés objet. En fait, les approches fonctionnelles (ex : SADT, SA-RT), traditionnellement utilisées, sont issues de la pensée impérative (*que doit faire le système?*), alors que les approches orientées objet (ex : OOD, HOOD) privilégient la pensée applicative (*sur quoi le système doit-il agir?*). Cette évolution est due au fait que les fonctions demandées au système changent tout le long du cycle de vie (maintenance évolutive et adaptative), alors que les objets auxquels s'appliquent les fonctions demeurent plus stables.

  Le passage des approches fonctionnelles aux approches orientées objet a bouleversé les habitudes des concepteurs. Les méthodologies fonctionnelles gardent un succès certain grâce notamment à leur accessibilité et leur grande diffusion. Les méthodologies orientées objet, nées de cette évolution des concepts, amènent des améliorations importantes tant au niveau des étapes de conception du logiciel qu'au niveau de la qualité du produit final. Étant donné ces avantages, une grande partie des concepteurs de logiciels y ont adhéré. Ainsi, deux types de méthodologies se côtoient actuellement : les méthodologies fonctionnelles (ou structurées) et les méthodologies orientées objet.

Étant donné la diversité des méthodologies existantes, le choix n'est pas aisé. Il est nécessaire de prendre en compte la nature des systèmes étudiés pour choisir celle qui convient le mieux. Pour l'analyse et la conception des systèmes temps réel, nous travaillons avec deux types de méthodologies. La première est une méthodologie fonctionnelle associant SA-RT et les réseaux de Petri, et la seconde est une méthodologie orientée objet, développée au LAAS [Paludetto 91], associant les objets Hood et les réseaux de Petri. Les deux parties suivantes sont consacrés à la présentation des principales caractéristiques de ces deux méthodologies.

I.4 Une méthodologie fonctionnelle SA-RT et réseaux de Petri

Parmi les approches fonctionnelles, la méthode SA-RT (Structued Analysis, Real Time) est l'une des méthodes les plus utilisées, autant dans le monde universitaire que dans le monde industriel. Elle offre les outils nécessaires pour gérer la complexité des systèmes à travers les phases d'analyse des besoins logiciel et de conception préliminaire. Elle apporte également une aide appréciable dans les phases de conception détaillée et d'implémentation, en fournissant une documentation riche et cohérente sur les différents aspects des systèmes étudiés. La partie suivante, adaptée de [Benzina et Paludetto 97], est consacrée à la description de cette méthode et plus particulièrement de la version que nous utilisons conjointement avec les réseaux de Petri. Des exemples d'applications peuvent être consultés dans [Benzina et Paludetto 96b]. D'autres travaux ont également associé les réseaux de Petri à la méthode SA-RT, citons par exemple [Mrhailaf et Sahraoui 93], [Ben Ahmed *et al.* 96].

I.4.1 Historique

SA-RT est l'aboutissement d'une suite de recherches visant à obtenir une méthode d'analyse et de structuration de logiciels. Les premiers travaux sont ceux de DeMarco sur les diagrammes de flots de données [DeMarco 78]. Plusieurs chercheurs ont repris ces travaux pour adapter cette méthode de structuration (SA) aux problèmes temps réel (RT). Plusieurs variantes de SA-RT ont ainsi vu le jour. Elles obéissent aux mêmes principes généraux mais diffèrent sur certains aspects secondaires. Parmi ces variantes, celle de Ward et Mellor (que nous citerons sous l'abréviation WM) [Ward et Mellor 85] et celle de Hatley et Pirbhai (HP) [Hatley et Pirbhai 87] sont les plus diffusées. Nous utilisons une version de SA-RT (mixte, WM et HP) enrichie par les réseaux de Petri pour la description du contrôle, plus adaptée aux systèmes temps réel complexes.

I.4.2 Description générale

Comme dans toute méthode d'analyse, la première question à se poser est : *que doit faire le système?* Plus tard se posera la question : *comment le faire?* (Le quoi et le comment). La démarche adoptée par SA-RT est une démarche descendante utilisant l'abstraction et la décomposition. La réponse à la première question se résume en deux mots qui définissent en même temps toute l'utilité du système : sa fonction globale. Cette fonction sera décomposée en sous-fonctions un peu plus détaillées qui seront à leur tour décomposées en sous-fonctions encore plus détaillées. Une analyse SA-RT obéit toujours à cette structuration hiérarchique, c'est incontestablement l'un des aspects les plus importants de la méthode.

La méthode est basée sur trois aspects indissociables et menés en parallèle :

- Les diagrammes : aspect graphique.
- Les mini-spécifications : formalisme textuel.
- Le dictionnaire de données : vision données.

Les diagrammes servent à illustrer les différentes étapes de la démarche. La clarté, la simplicité et le pouvoir d'expression de ces diagrammes contribuent largement au succès de la méthode. Deux types de diagrammes sont principalement utilisés : les diagrammes de flots de données (DFD) qui représentent l'aspect statique et les diagrammes de flots de contrôle (DFC) qui représentent l'aspect dynamique. Malgré la distinction faite, ces deux types de diagrammes interagissent et coexistent sur le même graphique (Vision WM). Les symboles graphiques utilisés pour les représenter sont exposés ci-dessous (figure *I.3*).

Les versions WM et HP ont leurs spécificités en ce qui concerne les symboles utilisés (cf. [Ward et Mellor 85] et [Hatley et Pirbhai 87]). En particulier la version WM propose de distinguer les flots de données continus (continûment disponibles) et discrets (non continûment disponibles). Nous pensons qu'une telle distinction n'est pas toujours

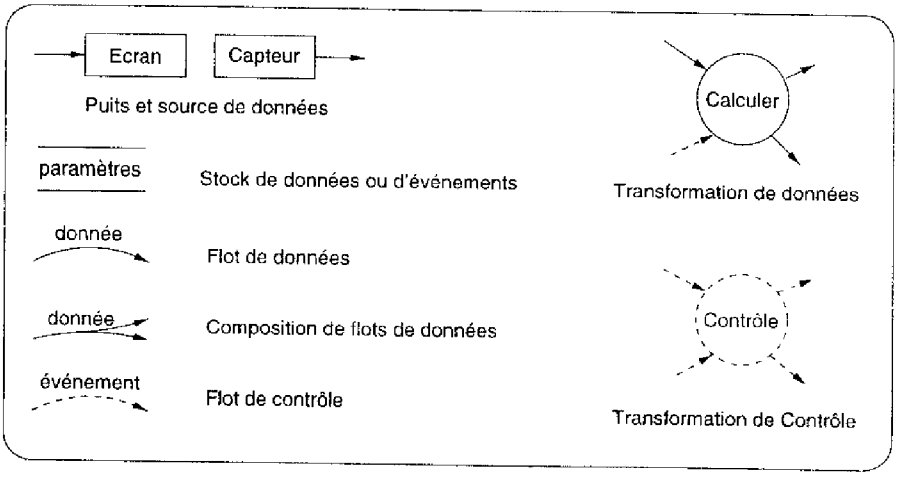


fig I.3 – Symboles graphiques utilisés dans SA-RT

indispensable en phase d'analyse. Si le caractère continûment ou non continûment disponible d'une information intervient dans le comportement du système, cet aspect doit être modélisé lors de l'analyse des besoins et non simplement signalé par un symbole graphique particulier. En général cet aspect influence surtout l'architecture de la solution, donc il intervient plutôt en phase de conception. En phase d'analyse des besoins il se traduit par des contraintes temporelles qui doivent être analysées globalement et au niveau de chaque transformation.

L'étude commence par délimiter l'environnement du système étudié et produit ainsi le **diagramme de contexte**. Par la suite, il faut préciser les fonctions réalisées par le système, ces fonctions constitueront les **transformations de données** du DFD. Il faut aussi rechercher les liens entre les transformations de données par **flots de données** interposés.

Les **transformations de contrôle** du DFC permettent de coordonner le travail des transformations de données. Elles sont spécifiées elles-mêmes à l'aide des structures de contrôle **Cspecs** (Control specifications) dans lesquelles interviennent les événements issus de l'environnement.

Chaque transformation de données est décomposée à un niveau hiérarchique plus bas, ce qui donne un nouveau DFD avec éventuellement le DFC correspondant. Quand une transformation est jugée primitive, elle est appelée **transformation terminale**. Alors, une description appelée **mini-spécification** (ou Pspec pour Process specification) lui est associée. Ces étapes sont itérées plusieurs fois (pour chaque transformation de données) en respectant la logique de la structuration hiérarchique décrite plus haut. Toutes ces notions seront détaillées plus loin. La figure *I.4* donne une vue globale des divers diagrammes et documents de la méthode SA-RT.

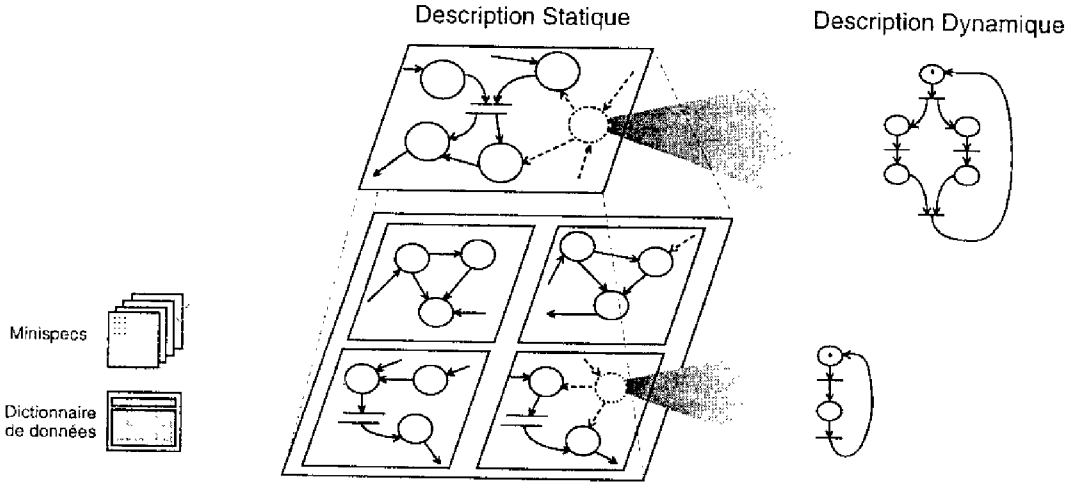


fig I.4 – Vue d'ensemble de la méthode

Ainsi, la démarche peut être résumée par les étapes suivantes :

1. Effectuer la liste des entrées/sorties (données et événements).
2. Effectuer la liste des fonctions principales réalisées par le système.
3. Établir le diagramme de contexte (à la première itération seulement).
4. Pour le niveau d'abstraction considéré, grouper éventuellement les fonctions et définir les transformations de données.
5. Établir le diagramme SA-RT (DFD et DFC).
6. Associer à chaque transformation de contrôle la description correspondante (Cspec).
7. Mettre à jour le dictionnaire de données.
8. Mettre à jour la liste des transformations.
9. Expliciter les mini-spécifications des transformations terminales.
10. Pour chaque transformation non terminale, reprendre en 1.

I.4.3 Les éléments de représentation

I.4.3.1 Le diagramme de contexte

Le diagramme de contexte sert à préciser l'environnement du système étudié. Dans ce diagramme, le système est représenté au centre par une transformation de données et, tout autour, sont disposées les entités qui sont amenées à interagir avec le système (capteurs, opérateurs, autres systèmes,...). Ces entités sont représentées par des sources et des puits d'informations. Des flots de données et de contrôle matérialisent les échanges entre le système et son environnement.

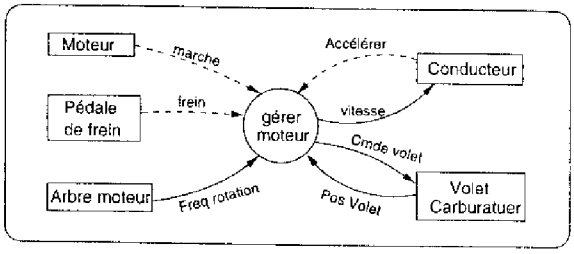


fig I.5 – Exemple de diagramme de contexte pour un système de régulation de vitesse

I.4.3.2 Les diagrammes de flots de données

Un diagramme de flot de données (DFD) d'un niveau hiérarchique *i*, résulte de la décomposition d'une transformation de données se situant au niveau *i-1*, immédiatement supérieur. Le DFD résultant doit donc prendre en entrée (respectivement, produire en sortie) les mêmes flots de données d'entrée (resp. de sortie) de cette transformation du niveau *i-1*.

Un diagramme flots de données est un diagramme transformationnel. Il montre les transformations, plus ou moins abstraites, appliquées aux données d'entrée pour produire les données de sortie.

Les transformations de données abstraites doivent être décomposées dans d'autres DFD de niveau hiérarchique *i+1*. La décomposition est arrêtée lorsque la complexité est faible ou lorsque les fonctions obtenues sont bien connues (bibliothèque).

I.4.3.3 Les mini-spécifications

Appelées aussi Pspecs (Process specifications), les mini-spécifications servent à expliciter les traitements effectués par les transformations de données terminales : *Comment obtenir les données de sortie à partir des données d'entrée?*. La description d'une mini-spécification est généralement textuelle et peut prendre la forme d'un algorithme. Lors de cette spécification, il faut veiller à préciser les pré-conditions nécessaires à l'exécution de la transformation en question et les précautions à prendre lors du passage à l'implémentation. Les mini-spécifications sont complétées au fur et à mesure de l'analyse et constituent un support appréciable pour la conception et l'implémentation.

I.4.3.4 Les diagrammes de flots de contrôle

Les diagrammes de flots de contrôle (DFC) apportent une dynamique à l'étude, ils sont constitués d'une ou plusieurs transformations de contrôle qui représentent les centres de décision pour le niveau hiérarchique considéré. Les flots de contrôle entrants et sortants des transformations de contrôle véhiculent :

- les événements provenant des sources d'information,
- les ordres d'activation destinés aux puits d'information,

- les événements provenant des transformations de données (information sur l'évolution du système),
- les ordres ou signaux d'activation destinés aux transformations de données.

En fonction des informations qu'elles reçoivent, les transformations de contrôle coordonnent l'activité des transformations de données et leurs interactions avec l'environnement.

La spécification du contrôle se fait dans le diagramme Cspec (Control specification). Ce diagramme peut prendre plusieurs formes selon la complexité du contrôle à spécifier et selon la version de SA-RT adoptée. Il peut comporter des tables d'activation de processus, des machines à états finis avec les conventions de Moore, de Mealey ou sous une forme hybride [Hatley et Pirbhai 87], ou encore des réseaux de Petri. Ces derniers ne font pas partie de la méthode SA-RT au sens WM ou HP. Toutefois, étant donné leurs avantages, ils ont été associés à la méthode [Benzina et Paludetto 96b]. En effet, ils offrent un très fort pouvoir d'expression et possèdent des outils de simulation et de validation fort pratiques.

I.4.3.5 Spécification du contrôle par réseaux de Petri

Utilisés conjointement avec SA-RT, les réseaux de Petri ont pour rôle de contrôler le système ou le sous-système en engendrant les commandes relatives à leur niveau d'abstraction en cohérence avec les informations fournies par les transformations de données. Pour cela, ils prennent en compte les événements qui se produisent dans le système ainsi que ceux produits par les transformations de données du même niveau.

Afin de le mettre en relation avec son environnement, un réseau de Petri doit être un réseau étiqueté (figure *I.6*). Les mentions portées sur les étiquettes correspondent aux flots de contrôle entrants et sortants de la transformation de contrôle. Les notations utilisées et leurs significations sont explicitées ci-dessous.

q_i est une fonction booléenne de données. Elle traite les données internes et/ou les entrées du système et les événements externes, exprimés sur les flots de contrôle entrants.

O_i est une liste de transformations (ou d'opérateurs) à activer par l'intermédiaire des flots de contrôle sortants ou une liste d'événements sortant du processus de contrôle.

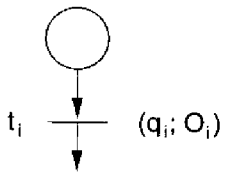


fig I.6 – Réseau de Petri étiqueté

Les données sont définis dans le dictionnaire des données et les opérateurs dans les mini-spécifications.

Règles opératoires : t_i est tirable si elle est sensibilisée et si q_i est vraie. Le tir de t_i déclenche les transformations (ou les événements de sortie) O_i. La figure *I.7* montre les cas particuliers de cette représentation.

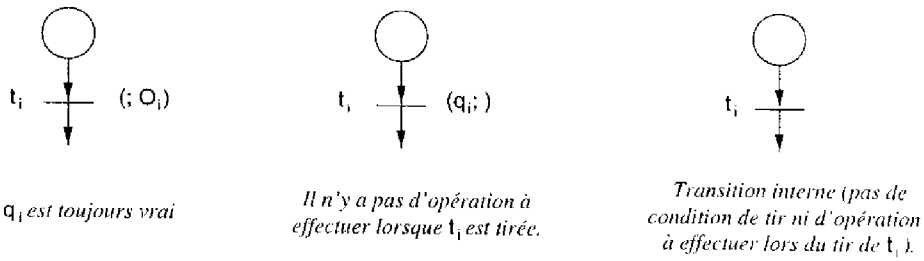


fig I.7 - Cas particuliers de réseaux de Petri interprétés

Lorsque la complexité du système ou les objectifs de validation le justifient, d'autres classes de réseaux de Petri peuvent être utilisés: les réseaux de Petri temporels [Merlin et Farber 76], les réseaux de Petri à prédicats-transitions [Genrich 86], les réseaux de Petri à objets [Sibertin-Blanc 85], les réseaux de Petri colorés [Jensen 81], etc. Néanmoins, l'expérience montre qu'il est rarement nécessaire d'utiliser les réseaux de Petri de haut niveau. En fait, ces derniers, intègrent sur un même support les aspects structuration, contrôle, et données. Ce regroupement est intéressant pour analyser tous ces aspects ensemble. En revanche, ces modèles perdent de leur clarté et deviennent d'un abord plus difficile. En séparant les aspects statiques et dynamiques d'une part et en permettant une structuration hiérarchique d'autre part, SA-RT permet, dans la plus part des cas, de se contenter des réseaux de Petri classiques pour l'expression du contrôle.

I.4.3.6 Le dictionnaire de données

Le dictionnaire de données répertorie et détaille toutes les informations véhiculées par les flots de données et de contrôle. Il utilise une syntaxe précise du type **Backus-Naur** [Hatley et Pirbhai 87] et sert de support pour les phases ultérieures de développement ainsi que pour la documentation finale de l'étude. Comme les mini-spécifications, le dictionnaire de données fait partie du formalisme textuel accompagnant les diagrammes SA-RT.

I.4.4 Aspects temporels

Le temps est l'un des aspects les plus critiques des systèmes temps réels. Une méthodologie d'analyse de besoins utilisée pour les systèmes temps réel doit pouvoir considérer les contraintes de temps de l'application (réponse à un événement, temps d'exécution) comme un besoin au même titre que les autres besoins fonctionnels.

Lors de la phase d'analyse des besoins, une étude par la méthodologie SA-RT spécifie simplement (mentionne) les contraintes temporelles. A ce stade de l'étude, il ne s'agit pas d'amener des réponses quant à la manière de satisfaire ces contraintes. La spécification des aspects temporels est réalisée à l'aide de tables définissant les intervalles de temps tolérés entre l'occurrence des stimuli de l'environnement et la réalisation

des actions associées. Sont spécifiées également les fréquences d'occurrence de signaux périodiques ou encore la fréquence minimale de prise en compte de signaux continus (échantillonnage).

Étant donné la multiplicité des modèles à réseaux de Petri (temporels, temporisés, stochastiques, à objets....), la spécification du contrôle par des réseaux de Petri permet, en plus, de prendre en compte certains aspects temporels explicites comme, par exemple, les *chiens de garde (watch-dog)* qui permettent de définir des fenêtres temporelles associées à la prise en compte de certains événements.

Lors de la phase de conception, les versions WM et HP de la méthodologie SA-RT ne donnent pas de directives précises pour satisfaire les contraintes temporelles. Les solutions sont laissées à la charge du concepteur et la prise en compte des contraintes temporelles logiques est généralement réalisée par des choix d'architecture de système : utilisation de structures mono-processeur ou multi-processeurs, choix de la fréquence de l'horloge, scrutation des signaux du procédé ou utilisation d'interruptions matérielles, caractéristiques des convertisseurs analogiques-numériques et numériques-analogiques...

I.4.5 Conclusion

Par ses capacités de structuration, par la cohérence des outils qu'elle adopte et par la richesse de ses documents, la méthodologie SA-RT apporte de nombreux avantages au génie logiciel temps réel. Son utilisation conjointe avec les réseaux de Petri la rend très adaptée à la nature des systèmes temps réel. Bien qu'il existe plusieurs variantes de SA-RT, les concepts de base utilisés sont les mêmes, ce qui fait de SA-RT une méthodologie abordable par une grande partie des analystes-concepteurs.

I.5 Une méthodologie orientée objet HOOD et réseaux de Petri

A l'orée du vingt et unième siècle, la pensée orientée objet apparaît comme la meilleure approche de développement de logiciels. Cela semble se confirmer de plus en plus, y compris dans le domaine du Temps Réel. Il existe de nombreuses méthodes. Près de trois cents ont été recensées [Rumbaugh 96]! Les plus connues sont OOD [Booch 86], [Berard 89], [Coad et Yourdon 89], OMT [Rumbaugh *et al.* 91], HOOD [ESA 89], ... Mais, rares sont celles qui permettent de bien intégrer les problèmes temps réel de façon satisfaisante. Cela est essentiellement dû au fait que ces problèmes sont spécifiques et donc difficiles à résoudre globalement. Pour une même application, il est souvent nécessaire de faire appel à plusieurs techniques, outils ou formalismes de description. De plus, la démarche doit être uniforme et homogène le long du cycle de vie. Elle doit également intégrer un aspect validation très important, validation sans laquelle le produit logiciel est rejeté (aéronautique, transport, militaire, ...).

Ainsi, il serait illusoire de parler d'une méthode de génie logiciel pour le temps réel. La réalité de la problématique veut qu'un ensemble de méthodes soit nécessaire. C'est la raison pour laquelle HOOD/PNO (Hierarchical Object Oriented Design with Petri Net Objects) se présente comme une **méthodologie** pour la définition de logiciels temps réel. Elle couvre les phases d'analyse, de conception, de mise en œuvre, de validation et de maintenance. La présentation complète de la méthodologie nécessiterait un ouvrage à elle seule. Aussi dans ce qui suit, nous décrirons uniquement la démarche qu'elle préconise en phase d'analyse des besoins. Cette description est adaptée de la présentation de la méthodologie faite dans [Paludetto et Benzina 97]. Pour de plus amples détails concernant la méthodologie elle-même et les méthodes classiques sur lesquelles elle s'est appuyée, il faut se référer à [Paludetto 91], [Paludetto *et al.* 90], [Raymond et Paludetto 92].

Historiquement, HOOD/PNO est née de HOOD [ESA 89] et des réseaux de Petri. L'objectif était de combler certaines lacunes de HOOD relatives à la définition des objets de l'application, à la description des comportements et à la modélisation des caractéristiques intrinsèques au temps réel. HOOD est une méthode de conception et de développement des grands projets industriels. Elle fut définie par un consortium d'industriels en 1986 à la demande de l'ESA (European Space Agency). HOOD/PNO améliore notablement HOOD en intégrant une méthode d'analyse des besoins orientée objet, une aide semi-formelle à la définition des objets, un outil formel de description des comportements des objets et des règles d'implémentation pour un langage cible donné. D'autres travaux ont également associés les réseaux de Petri à des approches orientées objet, citons par exemple [Bachatene 94], [Sibertin-Blanc 96], [Di Giovanni 91], etc.

I.5.1 HOOD/PNO dans le cycle de vie

Les facilités d'implémentation et, par voie de conséquence, de vérification et de maintenance des objets définis à partir des objets réels de l'espace du problème est un des aspects les plus attrayants des approches orientées objet. Cette approche appliquée aux systèmes temps réel soulève la remarque évidente que les objets réels sont rarement détruits. Ils sont donc persistants et impliquent ainsi la persistance des objets logiciels associés. HOOD/PNO part de ce constat et préconise, en phase préliminaire, de recenser les objets réels de l'environnement du problème et de leur associer des classes et/ou des objets logiciels. Ce travail peut être réalisé pour les besoins propres du projet et, par la même occasion, il permet d'enrichir le domaine du problème, ce que [Berard 89] nomme *OODA* (Object Oriented Domain Analysis).

Le processus d'analyse des besoins orientée objet s'accommode des deux types d'approches, descendante (par décomposition ou *top down*) et ascendante (par composition ou *bottom-up*). L'étude débute généralement par une démarche descendante qui permet de déterminer les objets abstraits de haut niveau (ou parents) susceptibles de composer le système. Chaque objet parent est décomposé en objets enfants. Ensuite, chaque objet enfant est isolé et analysé lui-même suivant le même processus (voir figure *I.8*).

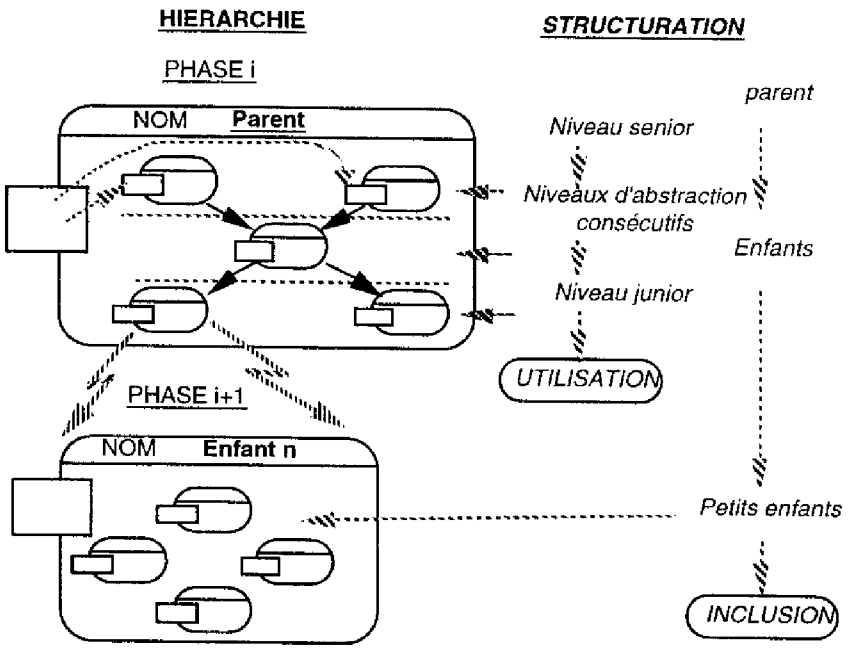


fig I.8 - La hiérarchie de structuration HOOD et HOOD/PNO

La hiérarchie ainsi obtenue fait apparaître deux types de relations. La *relation d'inclusion*: un objet parent de niveau *i inclut* des objets enfants de niveau *i+1*. Les objets frères coopèrent entre eux dans une *hiérarchie d'utilisation* senior-junior: un enfant senior *utilise* un enfant junior. Cette relation d'utilisation signifie que le senior effectue une demande d'opération au junior. En phase d'analyse, les objets ainsi définis peuvent être des objets en cours de définition, des classes ou des instances de classes. (Une classe peut être définie en parallèle pour le besoin de l'analyse). En phase de conception, la structuration fera référence à des objets ou à des instances de classes créés statiquement ou dynamiquement.

Si la phase de pré-analyse des besoins a été complètement menée, la démarche globalement descendante peut être partiellement suspendue afin de bâtir l'objet par composition à l'aide d'objets ou d'instances de classes appartenant au domaine de l'application. Ensuite, un retour en arrière sur l'objet parent est effectué pour l'affiner davantage. Des corrections, vérifications et prototypages sont également réalisés au fur et à mesure. Ce mécanisme de va-et-vient entre la définition du parent et des enfants est entretenu tant que l'objet parent ne donne pas satisfaction vis-à-vis du rôle qu'il doit assurer. En réalité, une telle démarche conduit à analyser *un peu* (un objet), concevoir *un peu* (toujours le même objet), implémenter *un peu*, vérifier *un peu*, et ensuite à recommencer pour chaque objet. Elle constitue ainsi un mini-cycle récursif. Par ailleurs, étant donné que chaque objet est isolé et étudié séparément, le travail peut être mené en parallèle par des équipes différentes. Ainsi, les objets sont étudiés suivant des *mini-cycles de vie récursifs parallèles.*

Après avoir avancé quelques idées de base de la méthodologie, les principales étapes sont résumées ci-dessous :

1. Apprentissage du problème et des besoins utilisateur. Un premier recensement des objets physiques de l'environnement et des relations qu'ils doivent avoir avec le système est effectué.

2. A l'aide des connaissances rassemblées précédemment, déterminer les objets réels abstraits. Par exemple, dans le cas d'un atelier de production manufacturière, une cellule d'usinage ou une machine sont de bons objets réels abstraits suivant que le problème posé concerne le niveau coordination ou le niveau commande locale de l'atelier.

3. Étant donné un niveau d'abstraction, il est possible de regrouper des objets pour en définir une classe. A ce stade, nous manipulons encore des objets réels.

4. Décrire chaque objet et classe d'objet. Trois points de vue sont importants : que fait réellement l'objet? Pourquoi fait-il cela et comment? La réponse au *quoi* donne le comportement externe de l'objet, celle au *comment* définit ses besoins internes (structure, comportement et opérations). Enfin, la réponse au *pourquoi* définit la position de l'objet dans la hiérarchie ainsi que les relations avec ses frères.

5. A l'aide des classes et objets physiques définis à l'étape précédente, déduire les classes et objets logiciels. La première question qui se pose est : faut-il bâtir des classes ou des objets *fidèles* à ceux définis à l'étape précédente? La réponse est non car les descriptions physiques n'intègrent pas toutes les responsabilités requises par les besoins logiciels propres.

   La description de l'objet est formalisée dans un document de spécification, l'ODS (Object Description Skeleton), celle de la classe dans l'OCS (Object Class Specification).

6. Définir les objectifs (les fonctions) du système à concevoir et procéder à une première structuration en objets du système. C'est ici que débute la structuration proprement dite en niveaux d'abstraction. A chaque niveau, il faut se poser la question : *que doit faire l'objet (ou sous-système)* (le *quoi*). D'autres objets abstraits, ensembles d'objets ou d'instances de classes déjà analysés, seront peut-être créés suivant les niveaux d'abstraction (réponse à la question «*sur quoi agit l'objet père*»). Un squelette de description de chaque objet impliqué dans le système est également décrit.

   A partir d'ici la procédure peut être différente suivant que l'approche menée est globalement ascendante ou descendante.

   Cas d'une approche ascendante :

7. Pour chaque objectif à atteindre, déterminer les objets nécessaires (objets ou instances de classes d'objets spécifiés aux étapes 5 et 6).

8. Mettre les objets ainsi déterminés en relation entre eux de façon à atteindre chaque objectif (description par Réseaux de Petri lorsque cela est nécessaire). Procéder à la vérification et à la validation de chacun d'eux.

   Cas d'une approche descendante:

7. Pour un niveau envisagé, définir les objectifs et les responsabilités de l'objet (parent) vu de ce niveau.

8. Décrire le comportement (PNOBCS ou Petri Net OBject Control Structure) du parent à l'aide de réseaux de Petri. Procéder à la vérification des bonnes propriétés et calculer les composantes conservatives. Compte tenu des objets définis à l'étape 6, affecter une composante conservative à un objet enfant (ou inversement). Ainsi, sont obtenus les objets enfants, et les composantes conservatives correspondantes représentant leurs comportements externes vus du corps du parent. L'ensemble des composantes conservatives associées aux objets enfants doit couvrir le PNOBCS du parent.

   Enfin,

9. Le prototypage et l'implémentation de chaque objet (ou classe d'objet) sont effectués en s'appuyant sur l'ODS (ou l'OCS) et sur les règles établies par la méthode pour un langage cible donné. Les tests et vérifications partiels ou complets sont conduits au fur et à mesure.

10. Pour chaque objet abstrait d'un niveau donné, reprendre en 1 ou en 2 suivant que l'objet abstrait et le niveau d'abstraction lui-même sont bien définis ou non.

11. Les tests unitaires d'intégration et système sont conduits suivant le cycle en V (voir figure *I.2*).

I.5.2 Description des comportements

La méthode HOOD apporte un bon pouvoir d'abstraction et de hiérarchisation. En revanche, elle souffre du manque de moyens de description des comportements notamment dans le cadre des systèmes temps réel. Dans HOOD/PNO, les réseaux de Petri ont été introduits comme outil de description des comportements, comme aide à la définition des objets enfants en les associant à certaines composantes conservatives et comme moyen de validation. Nous abordons ici le principe de représentation des comportements des objets.

HOOD/PNO définit des objets passifs et des objets actifs. Un **objet passif** ne possède pas de structure de contrôle propre. Il ne comporte que des données et opérations sur ces données. Les seules structures de contrôles sont celles des opérations elles-mêmes (OPCS dans HOOD pour OPeration Control Structure). Il n'y a pas d'interaction dynamique avec les autres objets. Les structures de contrôle des opérations décrites à l'aide des réseaux de Petri deviennent des PNOPCS (Petri Net OPCS) dans la méthodologie HOOD/PNO.

L'**objet actif** possède une structure de contrôle (OBCS dans HOOD pour OBject Control Structure) qui lui confère une dynamique (parallélisme, relation d'ordre dans les exécutions, ...). En particulier, les objets actifs sont en relation entre eux et peuvent également demander des opérations à des objets passifs. Les communications entre objets sont modélisées à l'aide des réseaux de Petri. L'OBCS décrit à l'aide des réseaux de Petri devient PNOBCS (Petri Net Object Control Structure) dans la méthodologie HOOD/PNO.

La structuration hiérarchique induite par les relations d'inclusion et d'utilisation fait que pour de nombreuses applications les réseaux de Petri de haut niveau ne sont pas nécessaires. Bien sûr, les classes de réseaux de Petri utilisées pour la description des comportements dépendent essentiellement de la complexité de l'application et des objets définis pour cette application. L'expérience prouve que les graphes obtenus sont relativement simples même dans le cas d'applications complexes nécessitant par moment des réseaux de Petri de haut niveau. Ceci constitue un atout supplémentaire pour la validation et la sécurité.

Au niveau du PNOBCS, une opération fournie aux utilisateurs est décrite par une séquence formée d'une transition, d'une place et d'une transition. La description détaillée de l'opération elle-même est effectuée ultérieurement, éventuellement par un réseau de Petri (PNOPCS) si elle comporte du parallélisme.

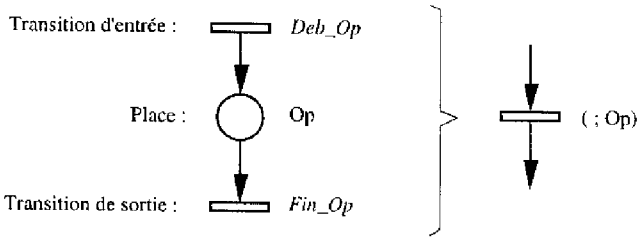


fig I.9 – Agrégation d'une opération représentée par réseau de Petri

La place représente le fait que l'opération est *en train d'être fournie*, sa transition d'entrée étant associée à l'événement *début d'opération* et sa transition de sortie à l'événement *fin d'opération* (figure *I.9*, partie gauche). Le jeton est un type de données, par exemple, attributs d'une pièce, caractéristiques d'usinage, gamme de fabrication, ...

Dans tous les cas où la durée de l'opération n'a aucune conséquence sur la description du comportement de l'objet vis-à-vis de ses relations avec ses frères, les événements *début et fin* d'opération peuvent être confondus et la séquence peut être abrégée en une seule transition (figure *I.9*, partie droite).

A tout instant, l'état du PNOBCS doit montrer les opérations pouvant être exécutées et les opérations ne pouvant l'être qu'après l'occurrence d'un ou de plusieurs événements. Les opérations fournies sont mises en évidence par les transitions de début d'opération qui sont franchissables. A un instant donné, l'état de l'objet est donc décrit par le marquage courant de son PNOBCS. L'avantage de cette approche réside dans le fait que la description d'évolutions simultanées est naturelle.

Le PNOBCS doit également montrer les opérations requises par l'objet à un instant donné. Ces demandes sont décrites par des transitions et sont mémorisées par des places de communication dans les objets offrant l'opération. La figure *I.10* synthétise les représentations non agrégées des PNOBCS.

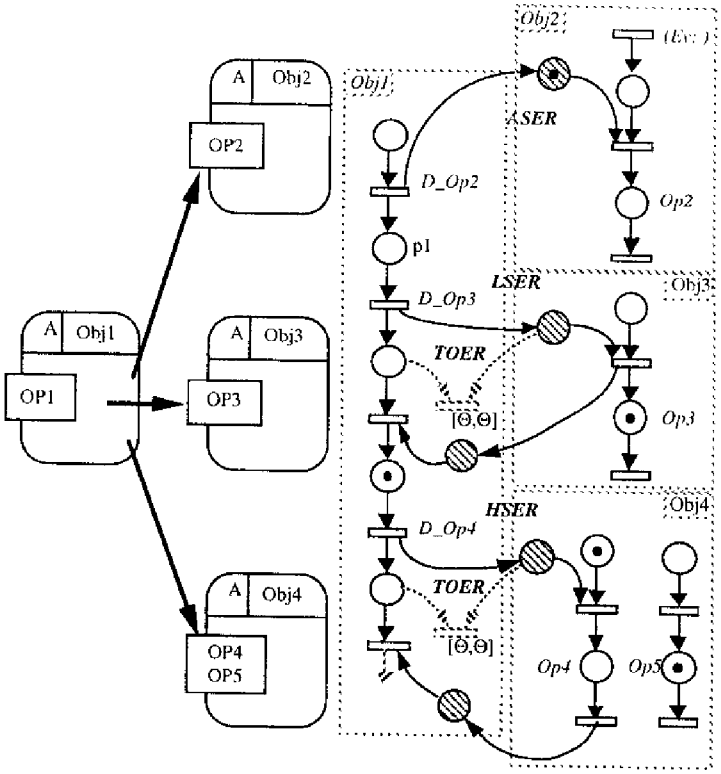


fig I.10 - Quelques moyens de communication décrits par réseaux de Petri

Ainsi, l'objet *Obj1* requiert en séquence les opérations *Op2*, *Op3* et *Op4*. L'objet *Obj2* offrira l'opération *Op2* dès la réception de l'événement *Ev*, l'objet *Obj3* est en cours d'exécution de l'opération *Op3*, alors que *Obj4* offre l'opération *Op4* et effectue simultanément l'opération *Op5*.

La méthodologie met en œuvre des protocoles de communication entre les objets. La figure *I.10* en illustre quelques uns. La demande *D_Op2* est asynchrone, la *D_Op3* est synchrone (rendez-vous) et la *D_Op4* correspond à un appel de procédure distante. Les requêtes *D_Op2*, *D_Op3* et *D_Op4* correspondent respectivement aux communications de type ASER, LSER et HSER définies dans HOOD (ASynchronous Execution Request, Loosely Synchronous Execution Request et Highly Synchronous Execution Request). Les demandes d'opérations de type LSER ou HSER avec une durée d'attente limitée peuvent être représentées aussi simplement. Ces requêtes correspondent aux communications de type TOER de HOOD (Time Out Execution Request).

Dans une description qui spécifie le comportement des objets les uns par rapport aux autres, il est possible de faire abstraction de la durée des opérations. Il est possible alors

d'adopter la notation abrégée (agrégation de l'opération, voir figure *I.9*). En phase de conception, cette représentation est suffisante. Ce n'est qu'en implémentation détaillée qu'il faudra éventuellement redéployer le réseau, suivant que la communication se fait entre objets appartenant à un même site ou à des sites distants.

La requête de l'opération par un *objet-i* et l'offre de cette opération par l'*objet-j* est représentée alors par une transition commune aux deux objets, fusion entre la transition de requête et la transition de service (figure *I.11*).

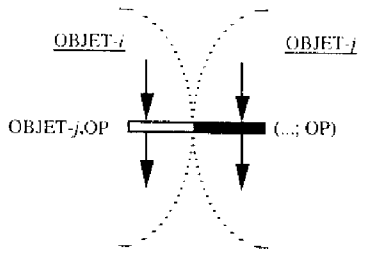


fig I.11 - Transition, agrégation du rendez-vous

Par ailleurs, il est indispensable de représenter séparément chaque objet ou classe d'objet et donc le PNOBCS correspondant. Ceci implique la scission des transitions exprimant les communications entre les objets. Lors de cette séparation, le concepteur décide quel sera l'objet demandeur de l'opération et quel sera l'objet qui offre cette opération (le choix est effectué en fonction des contraintes, notamment des contraintes temporelles).

L'opération offerte est représentée graphiquement par une flèche entrante épaisse et l'opération requise par une flèche sortante de même type. Les flèches entrantes ou sortantes vont se situer au niveau des transitions représentant respectivement l'offre et la demande d'opérations (figure *I.12*). La flèche sortante dénote une requête et peut être accompagnée de la notation pointée des langages orientés objet (LOO). La flèche entrante signifie une opération offerte. Cette dernière est notée avec une syntaxe classique rencontrée dans les réseaux de Petri interprétés.

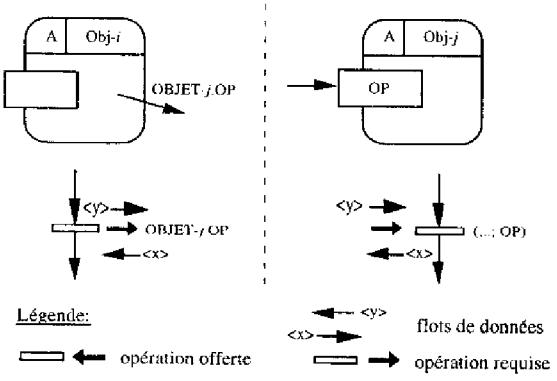


fig I.12 – Représentations séparées des opérations offertes et requises

De même, la communication peut transmettre une structure de données dans un sens ou dans l'autre (passage de jetons). Elle est représentée avec une flèche mince. Le sens indique si les données sont de type entrée ou sortie vues de l'objet considéré.

En conception architecturale, les structures de données et les opérations sont notées avec une certaine abstraction car la lisibilité et la compréhension du comportement sont meilleures. En fait, c'est la structure du réseau qui est importante pour la décomposition ou la composition de l'objet abstrait.

En conception architecturale détaillée, les mêmes réseaux pourront ensuite être transformés en réseaux de haut niveau suivant la représentation désirée. (Lorsque l'implémentation qui suit se fait avec un LOO, éventuellement, les réseaux à objets [Sibertin-Blanc 85] sont utilisés, ils permettent un codage quasi immédiat).

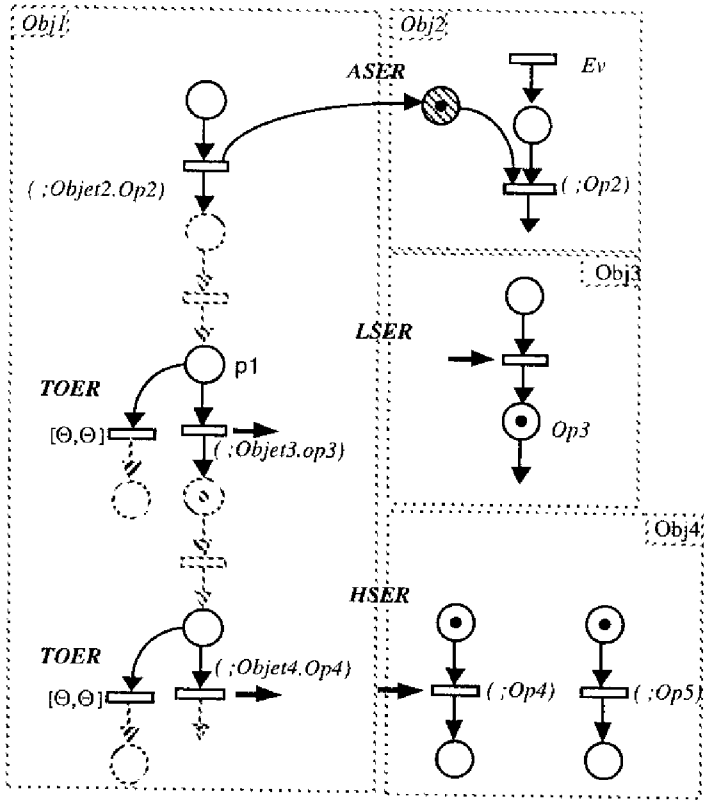


fig I.13 – Description abrégée des OBCS

Avec l'hypothèse de durée opératoire négligeable, dans une représentation globale, la transition décrivant la demande fusionnera avec la transition correspondant à l'offre de service, sauf dans le cas de requêtes asynchrones. Dans une représentation des objets séparés, cette transition est scindée entre l'objet client et l'objet serveur. Le réseau de la figure *I.10* est alors simplement représenté par celui de la figure *I.13* dans laquelle il faut considérer que les deux transitions relatives à la même opération côté appelant et côté appelé doivent être tirées de façon synchrone.

La scission est réalisée pour un simple confort de représentation du PNOBCS de chaque objet. Il faut souligner, toujours avec la même hypothèse, que les réseaux de Petri des figures *I.10* et *I.13* sont équivalents, c'est-à-dire qu'il est possible de passer de l'un à l'autre en appliquant les règles de réduction classiques (place substituable, place implicite, etc... voir [Courvoisier et Valette 80], [Brams 83], [David et Alla 89]).

Nous limitons cet exposé sur HOOD/PNO à l'étape d'analyse des besoins, ce qui nous intéresse le plus dans ce mémoire. Toutefois, rappelons que HOOD/PNO couvre tout le cycle de vie du logiciel. Ainsi, les réseaux de Petri permettent d'assister le concepteur pour la composition et décomposition d'objets ainsi que pour la réalisation de validations structurelles. Au niveau de l'implémentation, HOOD/PNO fournit des propositions pour aider à l'implémentation (en Ada et C++) des objets dont le comportement est défini à l'aide des réseaux de Petri.

I.5.3 Conclusion

L'approche orientée objet est constituée d'une panoplie de concepts susceptibles d'aider l'ingénieur à surmonter des difficultés dues à la complexité de l'application. Elle le conduit à dégager des objets au niveau global, c'est-à-dire dès l'analyse des besoins, et à appliquer le concept de localisation afin de permettre d'avoir une trace simple des besoins tout le long du cycle de vie, c'est-à-dire au cours des différentes transformations du cahier des charges. En bref, elle l'oblige à analyser, analyser encore et analyser toujours. Ce n'est qu'après cet effort d'analyse que la structuration en hiérarchies d'abstraction devient naturelle.

HOOD/PNO intègre une méthode d'analyse orientée objet, préliminaire indispensable à la phase de conception orientée objet. Elle est présentée sous forme d'étapes guidés s'appuyant sur l'aspect matériel du système à commander et sur le réseau de Petri de comportement. Il est montré que le problème principal de l'analyse et de la conception orientées objet n'est pas la définition des objets, mais bien celle des classes d'objets éléments clés de la réutilisabilité. La démarche proposés dans HOOD/PNO est une démarche pragmatique de définition des classes d'objets par une approche alternativement ascendante et descendante en étroite liaison avec les objets réels du problème. L'analyse des besoins est un domaine dans lequel la définition d'une méthode formelle est très difficile, voire impossible.

La description du comportement dynamique des objets est réalisée à l'aide des réseaux de Petri. Ces derniers permettent une validation formelle des comportements. Une des originalités de la méthode de conception HOOD/PNO réside dans l'exploitation des résultats d'analyse des réseaux de Petri au cours de la conception.

HOOD/PNO est une méthodologie encore bien jeune. Des travaux de mise à jour et de définition d'un outil support sont nécessaires. Il est à noter que STOOD [TNI 96], l'outil HOOD, permet d'atténuer partiellement cet inconvénient.

L'utilisation de cette approche orientée objet amène de nombreux avantage au développement du logiciel. Citons notamment la réutilisabilité des composants logiciels,

la facilité de la maintenance et de l'extension du logiciel, une bonne structuration pour les projets de grandes tailles et l'amélioration de la trace des besoins. Cependant, les concepts utilisés dans cette approche ne permettent pas de l'utiliser avec une spécification fonctionnelle des besoins (du type SA-RT), à laquelle de nombreux analystes-concepteurs sont attachés. Pour cela, et pour permettre aux utilisateurs des approches d'analyse fonctionnelles de bénéficier de ces avantages, nous avons pensé à rechercher un moyen de passage entre spécifications fonctionnelles et spécifications orientées objet.

I.6 Utilité d'un passage entre analyse fonctionnelle et analyse orientée objet

Comme nous l'avons déjà souligné, l'adoption d'une approche orientée objet comme HOOD/PNO à travers tout le cycle de vie du logiciel présente des avantages non négligeables (réutilisabilité des composants logiciels, facilité du partage des tâches, suivi de la trace des besoins, localisation...). Ainsi, une grande partie des concepteurs se sont reconvertis à l'approche orientée objet (les méthodes utilisées restent toutefois variées). Toutefois, le changement de méthode de travail n'est pas aussi aisée particulièrement pour les industriels ayant une longue expérience. Ils constituent une grande partie des utilisateurs de méthodologies et des demandeurs de logiciels. Cette difficulté peut être justifiée par les raisons suivantes :

- L'expérience acquise avec les méthodes fonctionnelles ne peut pas être délaissée. En outre, une grande partie de ces industriels pensent que ces méthodes suffisent dans la plupart des applications et ils ne changeront d'avis que quand les méthodes fonctionnelles ne répondront plus à leurs besoins (Yourdon dans [de Champeaux *et al.* 90]).

- Les systèmes déjà développés avec des méthodes fonctionnelles sont trop utiles pour être abandonnés, ces systèmes doivent être maintenus, étendus et utilisés parfois comme noyaux pour des développements futurs.

- La grande diffusion de certaines techniques fonctionnelles comme les diagrammes SA-RT et leur adoption par la majorité en font un outil universel. Par ailleurs, il existe dans ce domaine une littérature importante qui expose les méthodes. Les plus connues sont [Ward et Mellor 85] et [Hatley et Pirbhai 87]. Une étude comparative (datant toutefois de 1989) nous apprend qu'aux États-Unis chacune des deux versions de SA-RT est utilisée par 1/6 des analystes en systèmes temps réel [Wood et Wood 89].

- La disponibilité d'outils informatiques conviviaux qui supportent les méthodes fonctionnelles (CASE Tools) dont certains sont du domaine public.

- La complexité technique des méthodes orientées objet encore du domaine de la recherche et donc posant un problème de rentabilité. Par ailleurs, il n'y a pas

encore d'unanimité sur une méthode ou un groupe de méthodes orientées objet. La grande diversité des méthodes dans ce domaine rend la formation et le choix difficiles. Il y a bien des tentatives d'unification (UML [Rational 97]), mais elles sont encore en cours d'étude.

- Le passage à de nouvelles techniques nécessiterait une étape de formation dont le coût n'est pas à négliger. Ce problème intéresse particulièrement certains chercheurs qui proposent des méthodes de passage progressives pour des analystes-concepteurs habitués aux méthodologies fonctionnelles [Shlaer et Mellor 96].

Pour toutes ces raisons, l'étape d'analyse des besoins reste un domaine où les méthodologies fonctionnelles sont encore les plus connues et les plus utilisées.

Il y a donc une transition à franchir au cours de laquelle il serait intéressant de donner aux analystes-concepteurs les moyens de travailler selon une approche fonctionnelle pendant la phase d'analyse des besoins et selon une approche orientée objet pendant les phases ultérieures. Pour cela, nous nous proposons dans ce mémoire, de rechercher des règles de passage entre ces deux types d'approches.

La solution à un tel problème aiderait à faire évoluer les logiciels existants vers des techniques plus modernes et faciliterait le dialogue avec les non spécialistes, plutôt enclin à travailler avec des méthodes fonctionnelles. La figure *I.14* montre le positionnement de cette étape de passage. Ce dernier permettrait aux industriels et aux analystes-concepteurs de procéder à une phase d'analyse des besoins à l'aide d'outils traditionnels, de type fonctionnel. Les spécifications qui résultent de cette phase d'analyse pourront ensuite être "traduites" en spécifications orientées objet. Ainsi, le développement pourrait être poursuivi selon une approche orientée objet.

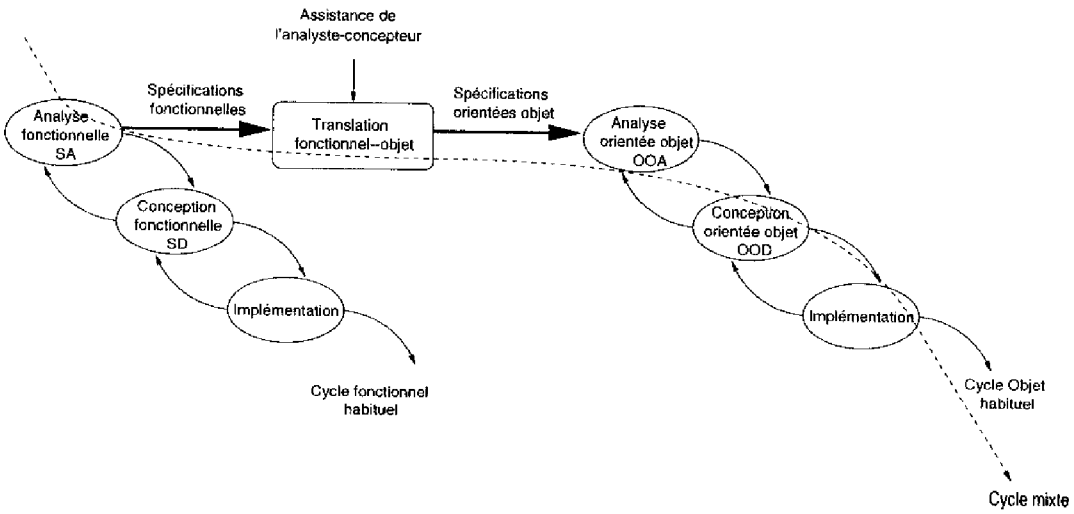


fig I.14 – Positionnement du passage fonctionnel-objet

L'idéal serait de proposer une méthode automatique qui permettrait d'effectuer ce passage. Le caractère automatique rendrait cette étape transparente au concepteur.

Conscients qu'il est impossible, dans un premier temps, d'aboutir à cette automatisation, nous essayerons de chercher des règles de passage qui permettront au concepteur d'effectuer la transition. Son intervention sera toujours nécessaire notamment lors de l'apparition d'ambiguïtés et de problèmes spécifiques au cas étudié.

Par ailleurs, dans un cycle de vie «classique» il est parfois nécessaire de remonter les étapes pour assurer la possibilité de retrouver la trace des besoins, pour pouvoir assurer le dialogue avec le demandeur de logiciel en cas de problèmes de conception, et pour prévoir les évolutions possibles du produit. Ainsi, il est également utile d'étudier le passage dans le sens inverse (objet - fonctionnel).

I.7 Conclusion

Dans ce chapitre, nous avons exposé quelques concepts de base du domaine du génie logiciel temps réel. Nous avons particulièrement souligné l'importance de l'étape d'analyse des besoins dans le cycle de vie du logiciel. Par ailleurs, un exposé sur les spécificités des systèmes temps réel, nous a permis de situer notre travail par rapport aux domaines de recherche sur le temps réel. En effet, pour ces systèmes, le temps doit être pris en compte dès la phase d'analyse des besoins au même titre que les besoins fonctionnels. Ainsi, une méthodologie d'analyse et de conception pour les systèmes temps réel doit utiliser les outils adéquats pour prendre en compte les spécificités de ces systèmes.

Dans le domaine des méthodologies d'analyse et de conception des systèmes temps réel, nous avons exposé deux méthodologies qui nous semblent bien adaptées à ce type de systèmes. La première est une méthodologie fonctionnelle qui utilise conjointement SA-RT et les réseaux de Petri. La seconde, HOOD/PNO, est une méthodologie orientée objet qui associe les réseaux de Petri aux objets HOOD. Cette dernière, couvre tout le cycle de vie du logiciel et apporte de nombreux avantages tant sur le processus de développement que sur la qualité des logiciels réalisés.

Cependant, pour les diverses raisons exposées plus haut, l'utilisation des méthodes orientées objet en général et de HOOD/PNO en particulier, n'est pas aussi répandue que l'utilisation de SA-RT, particulièrement pour la phase d'analyse des besoins. Nous avons donc pensé à rechercher un moyen de passage entre analyse fonctionnelle et analyse orientée objet. Nous avons montré, dans ce chapitre, l'utilité de ce passage et les motivations de notre travail. Dans le chapitre suivant, nous analysons les études qui ont traité ce sujet et les résultats proposés.

Chapitre II

Méthodologies fonctionnelles et méthodologies orientées objet

A priori, la réalisation d'un logiciel peut être satisfaisante aussi bien en utilisant une méthodologie fonctionnelle qu'une méthodologie orientée objet. Chacun choisit le type de méthodologie qui semble convenir le mieux à son application. Pour les raisons explicitées dans le chapitre précédent, certains analystes-concepteurs peuvent ressentir le besoin de trouver un passage entre les deux types d'approches. Dans ce chapitre, nous présentons la polémique que suscite ce sujet, les incompatibilités qui existent entre les deux approches et nous analysons les travaux réalisés dans ce domaine. Par ailleurs, nous exposons également certaines propositions complémentaires permettant d'améliorer les approches étudiées.

II.1 Position de la communauté scientifique

Dès l'émergence des approches orientées objet, plusieurs chercheurs se sont penchés sur le problème du passage entre spécifications fonctionnelles et spécifications orientées objet. Certains continuent encore à étudier le problème. Cependant, les publications ne sont pas toutes unanimes sur la pertinence et l'efficacité de ce passage.

II.1.1 Débat préliminaire

En 1990, au cours de la conférence OOPSALA/ECOOP un débat a été animé autour du sujet : *Quelle est la relation entre l'analyse structurée (SA) et l'analyse orientée*

objet (OOA)? [de Champeaux *et al.* 90]. Trois questions précises ont été traitées par des chercheurs de renom (L. Constantine, I. Jacobson, S. Mellor, P. Ward, E. Yourdon et D. DeChampeaux) :

- L'analyse structurée [1] peut-elle être utilisée dans l'étape d'analyse des besoins pour un système qui va être conçu et implémenté d'une façon orientée objet?
- Si ce n'est pas le cas, est-il possible d'adapter l'analyse structurée, que faut-il lui ajouter? Si elle ne convient pas du tout, quel est l'obstacle principal à son utilisation?
- Dans le cas où l'analyse structurée et l'analyse orientée objet ont des domaines d'application différents, comment délimiter ces domaines? Y a-t-il un recouvrement entre les deux?

Ces questions résument bien le débat qui s'est installé bien avant cette conférence et qui continue à susciter un intérêt sur la question de la compatibilité entre approches fonctionnelles et orientées objet.

Les avis étaient très partagés, certains pensaient qu'il était possible d'allier les deux approches alors que d'autres trouvaient qu'il fallait abandonner l'une au profit de l'autre.

Pour conclure ce débat l'animateur résume les réponses ainsi :

- L'analyse structurée ne peut pas être utilisée pour spécifier les besoins d'un système qui va être développé et implémenté à l'aide d'une méthode orientée objet.
- L'analyse structurée ne peut pas être adaptée à la conception orientée objet. Cependant, certains (Constantine et Ward) suggèrent qu'en ajoutant des extensions et en utilisant les diagrammes entités-associations il est possible de passer de l'analyse structurée à l'analyse orientée objet.
- Les domaines d'application de l'analyse structurée et de l'analyse orientée objet sont les mêmes (pour le moment). Cependant, Constantine soutient qu'une représentation fonctionnelle convient mieux pour certains aspects ou fragments de problèmes.

Au pessimisme dégagée lors de ce débat s'opposent les nombreuses études qui se sont intéressées à trouver les moyens de passer d'une approche à l'autre: [McMenamin et Palmer 84], [Seidewitz et Stark 87], [Alabiso 88], [Gray 88], [Bailin 89], [Khalsa 89], [Ward 89], [Shumate 91], [Shumate 93], [George 96], [Shlaer et Mellor 96], [Vazquez 96]. Dans la partie suivante, nous résumons les positions des différents chercheurs en ce qui concerne le passage entre approches fonctionnelles et orientées objet.

1. Pour des raisons de fidélité au texte original, nous avons préféré utiliser une traduction mot à mot du terme anglo-saxon désignant l'analyse fonctionnelle *structured analysis*.

II.1.2 Discussion des différentes positions

Les avis concernant la compatibilité des approches fonctionnelles et orientées objet sont très variés. Ils s'échelonnent entre le rejet total de l'idée de mixer les approches jusqu'à la proposition de méthodes de passage. Nous résumons dans ce qui suit les différentes positions des chercheurs qui se sont penchés sur le problème :

a- Les approches sont complètement incompatibles.

C'est la position la plus radicale. Elle est justifiée par l'incompatibilité entre les abstractions utilisées dans chacune des approches. Certains, modèrent leur position en disant que, bien évidemment, si les ressources (matérielles et temporelles) sont illimitées, il est possible de tout faire ([Firesmith 91], Yourdon dans [de Champeaux *et al.* 90]). Yourdon traite les démarches qui proposent un passage entre approches fonctionnelles et approches orientées objet de "démarches brutales" qui ne peuvent pas être efficaces.

Après une étude sur les tentatives de passage entre analyse structurée et analyse orientée objet, [Firesmith 91] conclu que les risques encourus sont très grands et que les techniques fonctionnelles doivent être abandonnées au profit des techniques orientées objet.

Cette position radicale est quand même rare parmi les chercheurs. Il y a aujourd'hui un consensus sur la possibilité d'allier les deux approches même si certains jugent cette alliance préjudiciable à la qualité du logiciel.

b- L'analyse orientée objet est tellement proche du problème réel qu'il faut plutôt se demander si elle ne doit pas remplacer l'analyse fonctionnelle dans une démarche fonctionnelle.

C'est l'idée de Mellor (dans [de Champeaux *et al.* 90]) qui est très convaincu de l'adéquation de l'analyse orientée objet aux problèmes de génie logiciel. Cependant, cette démarche ne présente pas d'intérêt pour notre étude et ne semble pas être dans le sens de l'évolution des méthodes. Plus tard, Mellor abandonna cette idée et proposa une démarche progressive pour *migrer* des méthodologies structurées à celles orientées objet [Shlaer et Mellor 96].

c- Une analyse orientée objet convient beaucoup mieux. Le passage entre les approches est possible mais doit être évité.

Cette position plus nuancée est prise par une grande majorité de chercheurs pensant que l'analyse orientée objet convient mieux à un système qui va être conçu d'une façon orientée objet, particulièrement pour les systèmes de grande taille. L'utilisation des mêmes principes, lors des différentes phases du cycle de vie, facilite la tâche du concepteur. Cependant, ces chercheurs n'excluent pas la possibilité de commencer l'analyse avec une démarche fonctionnelle quand cela est inévitable.

Jacobson précise que si une analyse fonctionnelle est disponible il est préférable de l'exploiter plutôt que de refaire le travail [de Champeaux *et al.* 90]. Il avertit que le

passage d'un modèle à un autre est une activité complexe, qui doit être entreprise manuellement. Il recommande que ce passage soit effectué par les personnes qui ont fait l'analyse fonctionnelle initiale, car il connaissent mieux que quiconque la nature du système étudié.

Tout en ne conseillant pas de le faire, Booch comprend qu'il est parfois nécessaire de substituer l'analyse structurée à l'analyse orientée objet. Il suggère alors de bien délimiter les phases d'analyse et de conception. Par ailleurs, Booch avance que l'utilisation d'une telle démarche ne doit pas être justifiée par des raisons politiques (non honorables selon lui) [Booch 94].

d- Il vaut mieux investir dans la formation des analystes que dans la recherche de méthodes de transitions.

Shlaer et Mellor, auteurs d'une méthode d'analyse orientée objet [Shlaer et Mellor 92], misent sur la rééducation des analystes-concepteurs qui ont une formation basée sur les méthodes fonctionnelles. Dans [Shlaer et Mellor 96], ils proposent une stratégie de "migration" entre méthodologies structurées et méthodologies orientées objet. Ils utilisent des notions familières aux habitués de SA-RT (DFDs, machines à états finis) pour passer progressivement à l'analyse orientée objet OOA.

A long terme, le passage par une étape de formation est nécessaire pour mettre à jour les connaissances des analystes-concepteurs. Cela n'empêche pas la recherche d'une démarche de passage entre approches fonctionnelles et orientées objet afin de rendre cette étape de formation plus souple.

e- Il est possible de faire évoluer l'analyse structurée pour englober des considérations orientées objet.

Cette idée est à la base des travaux de Ward dans [Ward 89]. Il pense qu'en rajoutant certains principes orientées objet à la méthodologie SA-RT (dont il fut l'un des coauteurs [Ward et Mellor 85]), il est possible de l'utiliser pour exprimer des spécifications en termes de classes d'objets avec des propriétés d'héritage. Cette analyse peut alors être continuée par une conception orientée objet. Outre les DFDs, la méthodologie SA-RT s'appuie sur plusieurs autres diagrammes comme les diagrammes entités-associations. Tous ces diagrammes sont généralement utilisées dans des approches plutôt fonctionnelles mais, selon Ward, ils peuvent aussi bien être utilisés dans des approches orientées objet. Par ailleurs, il existe plusieurs outils de CAO basés sur les diagrammes SA-RT qui pourraient alors être utilisés pour créer des spécifications orientées objet.

Cette idée pourrait introduire beaucoup de confusions dans l'application des méthodologies qui sont déjà assez complexes. D'ailleurs, Ward n'a pas été suivi dans cette voie. Néanmoins, dans sa démarche il présente certaines idées intéressantes que nous examinerons plus loin.

f- Il est intéressant d'allier les deux types d'approches.

Certains trouvent des avantages à allier les approches. Dans [de Champeaux *et al.* 90], Constantine pense que le fait d'utiliser des concepts différents dans les phases d'analyse et de conception confère au moins l'avantage d'accentuer les limites entre ces activités différentes mais difficilement différentiables.

Shumate conseille d'équilibrer les aspects fonctionnels et orientées objet lors de l'analyse des besoins [Shumate 91]. Il rappelle que généralement les besoins sont exprimés d'une manière fonctionnelle, c'est le point de départ de l'analyse et le seul point de vue qui intéresse l'utilisateur.

Il faut tout de même relativiser cette position. Nous ne pensons pas que le fait d'allier systématiquement les deux approches soit toujours bénéfique. Le suivi des concepts orientées objet dans toutes les étapes du cycle de vie est beaucoup plus bénéfique à la qualité du logiciel. Il est possible d'utiliser un formalisme fonctionnel dans une approche orientée objet (pour décrire les opération d'un objet, par exemple) mais cette utilisation doit être bien circonscrite et sa place bien définie.

D'ailleurs, plusieurs méthodologies actuellement très répandues utilisent, à un niveau ou un autre de leurs démarches, une forme de spécifications fonctionnelles à l'aide des diagrammes DFDs. Dans OMT [Rumbaugh *et al.* 91], la démarche d'analyse s'appuie, entre autres, sur une *vue fonctionnelle* modélisée avec des DFDs. Dans la méthode OOA [Shlaer et Mellor 92] les auteurs associent à chacun des états des objets du modèle d'information, un DFD qui décrit les activités associées à cet état. Ceci constitue une reconnaissance de la grande diffusion et de la facilité de compréhension des diagrammes fonctionnels, particulièrement pour les utilisateurs.

g- L'utilisation d'une notation unifiée facilite la conversion.

Dans [Wasserman *et al.* 90], les auteurs proposent un "modèle de notation en génie logiciel" appelé *OOSD notation* (Object-Oriented Structured Design notation) qui peut être adopté indifféremment pour exprimer un processus de développement fonctionnel ou orienté objet. Dans cet article, les auteurs ne proposent pas de démarche de conception. Ils proposent uniquement un système de notation (un ensemble de symboles graphiques) neutre par rapport à la méthode de conception utilisée. Ce système de notation peut aussi bien servir pour exprimer une démarche de conception orientée objet qu'une démarche de conception fonctionnelle.

Les auteurs de cet article soutiennent que l'adoption d'un système de notation unifié permet une conversion facile des concepteurs, habitués aux approches fonctionnelles, vers des concepts orientés objet. Toutefois, [Wasserman *et al.* 90] ne propose pas de démarche de translation entre les deux types d'approches.

Comme nous le montrerons plus loin, les incompatibilités entre les deux types d'approches ne se situent pas simplement au niveaux des notations. C'est plutôt une différence de mode de pensée qui se reflète également dans les méthodes de travail.

h- Le passage est faisable et intéressant.

Plusieurs chercheurs sont convaincus de l'utilité de disposer d'une méthode de passage entre analyse fonctionnelle et analyse orientée objet. Les diverses publications sur ce sujet montrent son intérêt. L'étude critique des différentes techniques proposées permet de dégager des idées intéressantes sur ce passage.

Enfin, comme le précise D. Firesmith, le succès du mariage des approches ne doit pas être considéré dans l'absolu. C'est une équation globale qui prend en compte la réalisation d'un logiciel conforme aux besoins, la maximisation des qualités de ce logiciel (réutilisabilité, maintenabilité ...) et la minimisation du coût du cycle de vie [Firesmith 91]. Une approche de transition peut être jugée efficace selon certains critères et médiocre selon d'autres.

Dans la suite du chapitre, nous examinerons d'abord les incompatibilités entre approches fonctionnelles et approches orientées objet et les difficultés qui en découlent. Ensuite, à travers les études disponibles, nous présenterons les possibilités offertes pour passer de l'une à l'autre.

II.2 Incompatibilités entre les deux approches

Les incompatibilités entre approches fonctionnelles et approches orientées objet résultent de la manière dont les systèmes sont étudiés et des points de vue considérés. Ces incompatibilités sont dues aux différences entre les abstractions utilisées, l'ordre d'utilisation de ces abstractions, les cycles de développement utilisés, les problèmes posés par chaque approche et les qualifications techniques nécessaires pour appliquer chacune des méthodes.

II.2.1 Les abstractions utilisées

L'analyse structurée exploite plutôt l'abstraction de fonctions avec une légère abstraction de données. En revanche, l'analyse orientée objet utilise plusieurs abstractions pour modéliser le monde réel [Firesmith 91] :

- Abstraction d'objets pour identifier les objets et les classes.
- Abstraction de données pour identifier les attributs de ces objets.
- Abstraction de fonctions pour identifier les opérations des objets.
- Abstraction de processus pour modéliser les opérations concourantes réalisées par chaque objet.
- Abstraction d'exceptions pour prévoir les pannes et les solutions à ces pannes.

En outre, il y a une différence fondamentale dans l'ordre d'utilisation de ces abstractions. En effet, l'analyse fonctionnelle utilise l'abstraction de fonctions en premier lieu, puis l'abstraction de données. Certaines versions de l'analyse structurée permettent de commencer par l'abstraction de processus dans le cas de problèmes de parallélisme. En revanche, les méthodes orientées objet utilisent l'abstraction dans un ordre différent. Elles commencent par identifier les objets et les classes, puis utilisent l'abstraction de données pour modéliser les attributs des objets, ensuite l'abstraction des fonctions. Pour les problèmes temps réel, l'abstraction de processus est utilisée en même temps que l'abstraction d'objets car les objets réalisent souvent des activités parallèles.

Le fait de débuter l'analyse selon différents types d'abstractions est à la base des incompatibilités qui existent entre les deux approches. Rappelons que ce que l'une regroupe dans une entité, l'autre le disperse dans des entités différentes. Par exemple, en décomposant une fonction en sous-fonctions, ces dernières ne sont pas toujours des opérations relatives à un même objet ou à une même classe. De même, les fonctions dans une analyse fonctionnelle ne sont pas contraintes à agir sur un même objet ou sur des données appartenant à un même objet. Souvent ces fonctions agissent simultanément sur plusieurs objets.

II.2.2 Les problèmes spécifiques à chaque type de représentation

Lors de l'étape d'analyse des besoins, la représentation par objets fait apparaître des problèmes que la représentation fonctionnelle ne voit pas. Il s'agit des problèmes de communications et de synchronisations entre objets. En effet, les méthodes orientées objets qui se basent sur le concept de l'objet actif, permettent d'allouer un comportement propre à chaque objet. Ainsi les objets évoluent d'une façon parallèle et communiquent entre eux pour assurer la fonctionnalité globale du système. Ces problèmes de communication et de synchronisation n'apparaissent pas dès l'étape d'analyse des besoins dans les approches fonctionnelles, ils apparaissent ultérieurement.

II.2.3 La qualification technique nécessaire

L'utilisation d'une méthode orientée objet nécessite une qualification technique assez importante pour pouvoir assimiler les concepts apportés par cette méthode. La qualification technique nécessaire est supérieure à celle exigée par l'emploi d'une approche fonctionnelle. Ceci se traduit par un coût plus important, en temps et en investissements, pour la formation des ingénieurs.

II.2.4 Les risques associés au mixage des approches

Étant donné les incompatibilités présentées plus haut, le mixage des approches devient une opération risquée. D'une part, le suivi de la trace des besoins devient plus difficile car ces derniers sont exprimés différemment dans les deux approches.

D'autre part, si la conception orientée objet est effectuée par la même équipe qui a procédé à l'analyse des besoins selon une démarche fonctionnelle, il y a des risques de confusion. En effet, si le passage n'est pas bien effectué, il n'est pas improbable que, lors de l'étape de conception, des considérations fonctionnelles continuent à être prises en compte ce qui biaise le travail des concepteurs. Il n'est pas facile de changer de façon de penser instantanément.

II.2.5 Conclusion

Il en ressort que le problème principal qui s'impose à toute démarche qui veut passer d'une analyse fonctionnelle à une analyse orientée objet est la possibilité de faire le lien entre deux méthodes différentes, qui utilisent des abstractions différentes, dans un ordre différent, et qui traitent des problèmes parfois différents.

Ainsi, une éventuelle démarche doit principalement trouver la réponse à la question : *A partir du résultat d'une analyse fonctionnelle, comment identifier les objets qui interviennent dans la spécification, les reconstituer et aboutir à une analyse orientée objet correcte?*. Dans la partie suivante, nous examinons comment les différentes techniques proposées pour le passage entre spécifications fonctionnelles et spécifications orientées objet ont répondu à cette question.

II.3 État de l'art

Les travaux publiés sur le passage entre spécifications fonctionnelles et spécifications orientées objet s'appuient généralement sur les DFDs pour identifier les objets ou les définir. Les différentes techniques utilisées pour guider ce passage sont exposées dans cette partie. Les travaux explorés concernent, entre autres, ceux de Booch [Booch 86] qui, dans sa méthodologie de développement orientée objet, commence par une phase d'analyse fonctionnelle à l'aide des DFDs. Nous avons aussi étudié les travaux de B. Alabiso sur le passage des DFDs à la conception orientée objet [Alabiso 88]. Certains chercheurs sont persuadés de la possibilité d'allier les deux approches. Parmi eux K. Shumate traite de la compatibilité entre l'analyse structurée et la conception orientée objet [Shumate 91], [Shumate 93]. Ward s'est intéressé à l'intégration des concepts orientés objet dans l'analyse structurée [Ward 89]. Plus récemment, une stratégie informelle pour la correspondance entre modèles fonctionnels du type DFD et modèles orientés objet du type OMT a été présentée dans [George 96]. Par ailleurs, la possibilité de transiter par une algèbre formelle a également été explorée dans [Vazquez 96]. D'autres travaux qui se sont penchés plus ou moins sur le problème ont également été explorés [Booch 94], [Bailin 89], [Seidewitz et Stark 87].

II.3.1 Le diagramme de contexte

Dans la méthodologie SA-RT, le diagramme de contexte est un diagramme établi dès les premières étapes d'analyse des besoins pour mettre en évidence les liens entre le système à concevoir et son environnement (exemple figure *II.1*).

Le diagramme de contexte fait apparaître les sources et les puits d'informations. Ces éléments sont généralement relatifs aux entités physiques qui composent l'environnement du système étudié. Ainsi, ils forment un bon point de départ pour identifier une partie des objets qui interviennent dans la spécification de ce système. Cette idée est retrouvée dans presque tous les travaux qui traitent du passage entre spécifications fonctionnelles et spécifications orientées objet.

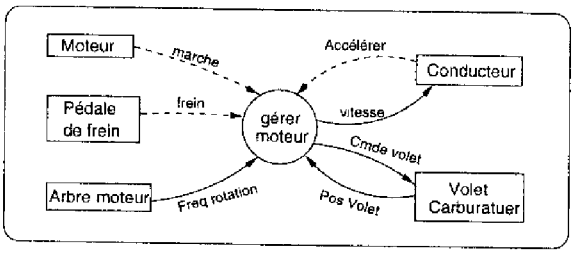


fig II.1 – Exemple de diagramme de contexte pour un système de régulation de vitesse

Bien qu'elle fournisse de bons candidats à une spécification par objets, cette approche reste superficielle et ne permet pas d'identifier les objets conceptuels. Le diagramme de contexte peut être utilisé comme point de départ pour l'identification des objets de niveau supérieur mais l'analyste-concepteur doit continuer sa démarche par d'autres moyens pour procéder à l'identification et la définition de tous les objets.

II.3.2 Les stocks de données

L'idée d'identifier les objets à partir des stocks de données des DFDs a été formulée dans plusieurs travaux mais avec une variation dans l'utilisation [Alabiso 88], [Bailin 89], [Booch 86], [George 96], [Shumate 91].

Booch les utilisent comme un des moyens pour identifier les objets intervenant dans l'activité du système [Booch 86]. Cependant, son utilisation des DFDs s'arrête là, et sa démarche continue plutôt par une analyse des besoins orientée objet que par une exploitation des résultats de l'analyse fonctionnelle. Dans [Booch 94], l'idée d'exploiter les spécifications fonctionnelles a été abandonnée en faveur d'une analyse orientée objet, toutefois l'auteur garde la possibilité d'effectuer cette démarche pour des raisons de compatibilité.

Shumate [Shumate 91] utilise aussi les stocks de données pour identifier les objets, mais il adopte une approche différente pour continuer la définition des objets. Après identification, les objets sont définis en regroupant les transformations de données entourant chaque stock de donnée. Cette démarche sera exposée plus loin.

L'utilisation des stocks de données permet d'identifier un certain nombre d'objets candidats à une spécification par objet. Cependant, les objets trouvés ne sont pas tous pertinents, il est parfois nécessaire d'en ignorer certains ou d'en regrouper d'autres qui traitent de la même entité physique.

En revanche, l'utilisation des stocks de données permet de faire apparaître certains objets conceptuels qui ne sont pas relatifs à une entités physique du système étudié et qui ne peuvent pas être identifiés à partir du diagramme de contexte.

II.3.3 Les données

Dans certains travaux est exploitée l'idée d'identifier les objets à partir des données qui circulent dans les DFDs. Dans [Alabiso 88], l'auteur va encore plus loin en affirmant que *"les données sont des objets et les objets sont des données"*. C'est une idée assez réductrice de l'objet, contradictoire avec le concept d'objet actif pouvant avoir un comportement asynchrone propre. En fait, le type d'objets proposé tend à centraliser le contrôle au niveau d'une spécification orientée objet.

Il existe aussi des variations sur le même thème. Booch propose de s'inspirer des noms figurant dans l'expression écrite des besoins pour identifier les objets du système [Booch 86]. D'autres s'inspirent du dictionnaire de données ou d'une représentation des données du système étudié avec le modèle entités-associations [Bailin 89], [Ward 89], [McMenamin et Palmer 84], [George 96].

Cette approche nous guide vers une représentation plutôt orientée donnée. Elle ne semble pas très efficace même si elle permet d'identifier une partie des objets. En effet, la proportion des objets pertinents par rapport à tous les objets identifiés est faible.

II.3.4 Partitionnements et regroupements dans les DFDs

L'idée de partitionner les éléments d'un DFD en un ensemble de groupes semble être l'idée la plus prometteuse pour l'identification et la définition des objets. Il s'agit de répartir les éléments d'un DFD (transformations de données, sources, puits, stocks) en un ensemble de groupes, judicieusement choisis, qui formeront les objets nécessaires à une spécification par objets. Cette idée a été utilisée différemment dans [Shumate 91], [Shumate 93] ou [Seidewitz et Stark 87] et dans [Ward 89].

Shumate préconise de centrer les objets autour des stocks, des puits et des sources de données, mais également autour des objets physiques et conceptuels (ex. compte, crédit...). Les transformations de données sont réparties sur ces objets selon un processus informel qui favorise la cohésion interne dans les objets, qui minimise le couplage entre eux et qui avantage la séparation des groupes pouvant évoluer parallèlement. La répartition des transformations de données ne se fait pas selon une méthode formelle, le suivi de ces critères, bien fondés du reste, est laissé à l'appréciation de l'analyste-concepteur qui, connaissant le système, est le seul capable de faire les bons choix. Dans la méthodologie orientée objet exposée dans [Seidewitz et Stark 87], la même technique est adoptée pour passer d'une spécification fonctionnelle à une spécification orientée objet.

Par ailleurs, comme les transformations de données qui concernent un objet peuvent être dispersées verticalement dans la hiérarchie, Shumate conseille une remise à plat

afin de faciliter le passage vers une bonne spécification à objets.

En outre, Shumate définit un ensemble d'étapes pour guider l'analyste-concepteur dans le passage entre spécifications fonctionnelles et spécifications orientées objet. Ces étapes passent par un regroupement des objets trouvés en «paquets» puis en «tâches» : c'est une manière de les structurer hiérarchiquement.

Dans [Ward 89], la même idée concernant le regroupement des transformations de données autour des stocks de données sert à définir les objets. Les objets sont préalablement identifiés grâce au modèle entités-associations [Chen 76], une des extensions apporté par Ward et Mellor à SA-RT [Ward et Mellor 85]. Cependant, le but recherché dans l'étude [Ward 89] n'est pas vraiment d'aboutir à une spécification orientée objet mais plutôt d'intégrer certains concepts orientés objet dans l'analyse structurée.

Cette idée de partitionnement et de regroupement dans les DFDs semble être la plus intéressante pour l'identification et la définition des objets. Elle aboutit à un partitionnement dans les DFDs comme le montre la figure *II.5*. Toutefois, telle qu'elle a été utilisée dans [Shumate 91] ou dans [Ward 89], il lui manque un aspect primordial pour les systèmes temps réel, l'aspect dynamique. Dans [Shumate 91] et [Shumate 93], cet aspect n'est pas pris en compte dans la définition des objets puisque le point de départ de la démarche est les DFDs. Il y a bien un diagramme dynamique dans la démarche adoptée, le EAD «Event Action Diagram». Cependant, ce dernier n'est utilisé que pour identifier les objets sources et puits (comme dans le diagramme de contexte). En revanche, dans la démarche adoptée par [Ward 89], l'aspect dynamique est bien pris en compte. Cependant, le contrôle reste centralisé : le passage vers une spécification orientée objet aboutit à la définition d'un objet central qui représente en quelque sorte un objet contrôleur. Ce concept d'objet contrôleur émane d'une vue fonctionnelle plutôt que d'une vue orientée objet puisqu'il limite l'autonomie des objets et engendre un fort couplage entre ces objets.

Cette idée de partitionnement et de regroupement dans les DFDs est donc à retenir. Elle mérite d'être complétée par des considérations dynamiques pour aboutir à une bonne spécification orientée objet.

II.3.5 Passer par une description algébrique

L'idée de passer par une description algébrique pour faire le lien entre DFD et représentation orientée objet a été formulée par F. Vazquez [Vazquez 96]. Le but de ce travail a été de donner un aspect algébrique formel aux DFDs et aux descriptions orientées objet. En fait, Vazquez propose une Σ-algèbre [2] pour décrire chacune des spécifications fonctionnelles et orientées objet et pour valider formellement ces descriptions.

En définissant cette Σ-algèbre pour les DFDs et pour les descriptions orientées objet, l'auteur propose d'utiliser cette algèbre pour guider le passage entre spécifications fonctionnelles et spécifications orientées objet. Cet idée semble fort intéressante car le

2. Une Σ-algèbre est une description algébrique des éléments d'un diagramme et des relations et opérations possibles entre eux. Une définition complète est disponible dans [Vazquez 96].

passage par une description algébrique donnerait à la démarche un aspect formel très intéressant. Cependant, les résultats obtenus montrent que la démarche aboutit à la définition d'un objet pour chaque élément des diagrammes DFD (transformations de données, sources, puits, stocks) :

- A une source correspond un objet dont l'attribut est la donnée générée par la source et une méthode $Get(x)$ (lire).
- A un puits correspond un objet dont l'attribut est la donnée fournie au puits et une méthode $Put(x)$ (écrire).
- A un stock de données correspond un objet dont l'attribut est la donnée conservée par le stock et les méthodes $Get(x)$ et $Put(x)$ (lire et écrire).
- A une transformation de données correspond un objet sans attribut et offrant les méthodes $Process(x)$ (l'opération réalisée par la transformation de données) et $Get(x)$ (fournir le résultat).

La description orientée objet à laquelle conduit cette méthode n'est qu'une réécriture des spécifications fonctionnelles à l'aide d'un formalisme objet. La démarche préconisée n'aboutit pas à un vrai processus d'encapsulation (données, opérations, comportement) qui permettrait d'avoir une meilleure vision objet. Cependant, l'idée est assez originale et devrait être explorée plus en profondeur en vue de rechercher un mécanisme de regroupement propre à l'algèbre utilisée.

II.3.6 Établissement d'une correspondance entre DFD et représentation par objets

Dans [George 96] le problème de passage entre spécifications fonctionnelles et spécifications orientées objet a été posé en termes de recherche de correspondance (mapping) entre les éléments qui constituent une spécification fonctionnelle et ceux qui constituent une spécification orientée objet. Le point de départ est constitué par les DFDs mis à plat, le dictionnaire de données et une description du système par un diagramme entités-associations. L'aspect dynamique (le contrôle) n'a pas été considéré dans cette étude dont le domaine d'application concerne les systèmes d'information, plus statiques par nature. L'objectif de la démarche est d'aboutir à une spécification du type OMT avec une description statique des objets (attributs et opérations) ainsi que des interactions entre objets.

La démarche proposée définit six étapes où l'apport de l'analyste-concepteur est crucial. En effet, la démarche propose les étapes à accomplir et les questions à clarifier mais ne donne pas d'outils pour le faire. Elle s'appuie complètement sur les compétences de l'analyste-concepteur pour conduire le travail. Les étapes proposées sont les suivantes :

1. Identification des objets à partir du diagramme entités-associations, du diagramme de contexte et des stocks de données.

2. Identification des attributs de chaque objet en parcourant exhaustivement le dictionnaire de données et en associant chacune des données à l'objet concerné.

3. Identification et affectation des opérations aux objets à partir du DFD mis à plat. Chaque opération est affectée à un objet en fonction des données qu'elle traite et de son importance relativement aux différents objets.

4. Identification des interactions entre objets à partir des relations du diagramme entités-associations.

5. Identification des messages échangés entre objets à partir des flots de données et des relations relevées dans l'étape précédente.

6. Établissement et affinement du modèle objet : diagramme OSMD - Object Structure and Message Diagram - (diagramme objet de OMT avec des échanges de messages).

L'avantage de cette démarche est son aspect "guide méthodologique" des différentes étapes à accomplir. Certes, aucun outil n'est proposé pour assister l'utilisateur, mais la démarche limite bien le cadre dans lequel doit se faire la correspondance (mapping). Malheureusement, cette démarche n'accorde aucune importance à l'aspect dynamique des systèmes. Elle est destinée à des systèmes à très faible composante dynamique. C'est un inconvénient majeur pour l'étude des systèmes à forte composante dynamique tels que les systèmes temps réel.

II.3.7 Analyse des différentes techniques de passage

II.3.7.1 Classification

Les différentes techniques (principes/idées) de passage, présentées dans ce chapitre, peuvent être classées en deux catégories : d'une part, les techniques d'identification des objets et, d'autre part, les techniques d'identification et de définition des objets. Les techniques d'identification des objets se limitent à rechercher les objets candidats à une représentation par objets du système étudié sans se préoccuper de la définition complète de ces objets. La définition des objets est réalisée par la suite par un travail d'analyse orientée objet (ex : [Booch 86]). La seconde catégorie des techniques présentées s'occupe en même temps d'identifier les objets et de les définir à partir des spécifications fonctionnelles. En fait, les techniques de cette seconde catégorie exploitent mieux le résultat de l'analyse fonctionnelle et tendent à minimiser l'effort d'analyse complémentaire à effectuer.

Les mêmes techniques peuvent également être classées en techniques graphiques et techniques contextuelles. Les techniques graphiques sont plutôt basées sur les diagrammes des spécifications fonctionnelles en faisant abstraction du contexte du problème (ex: identification des objets à partir des sources, des puits et des stocks). En revanche, les techniques contextuelles s'appuient sur les diagrammes, mais accordent

plus d'importance au contexte du problème. Elles se réfèrent aux connaissances de l'analyste-concepteur pour cet aspect contextuel. Les techniques de cette dernière classe sont moins faciles à automatiser car elles sont plus dépendantes d'informations subjectives sujettes à l'appréciation de l'analyste-concepteur. En revanche, elles sont plus proches du problème et donnent de meilleurs résultats.

Le tableau suivant classe les techniques exposées dans ce chapitre selon les critères énoncés ci-dessus :

Classification	Méthode d'identification	Méthode de définition	Méthode graphique	Méthode contextuelle
Diagramme de Contexte	•		•	
Stocks de données	•		•	
Partitionnement des DFDs		•		•
Description algébrique		•	•	
Mapping DFD vers OSMD		•		•

Remarquons aussi que toutes les démarches étudiées ont été unanimes pour choisir les DFDs (ou l'extension SA-RT) comme modèle des spécifications fonctionnelles. En revanche, les modèles objet adoptés sont tous différents, non seulement par le symbolisme graphique mais aussi par le concept même d'objet (assimilation données/objets, centralisation du contrôle...). Ceci montre bien un état de faits : il n'y a pas encore d'unanimité concernant les méthodologies orientées objet.

II.3.7.2 Importance de l'aspect dynamique

Dans les travaux étudiés, l'aspect dynamique n'a pas toujours été pris en compte par les chercheurs. Certains de ces travaux limitent les spécifications fonctionnelles aux diagrammes DFDs ou à des diagrammes statiques similaires. Dans le cas où les transformations de contrôle ont été prises en compte [Alabiso 88] [Ward 89], les auteurs préconisent la création d'un objet «contrôleur» qui s'occupe du contrôle des objets avoisinants. C'est une vision centralisée, plutôt adaptée à une démarche fonctionnelle. Par ailleurs, cette vision augmente le couplage entre cet objet contrôleur et les autres et limite l'autonomie des objets.

Pour les systèmes temps réel il est primordial que l'aspect dynamique soit pris en compte lors du passage des spécifications fonctionnelles vers les spécifications orientées objet. Par ailleurs, la démarche adoptée ne doit pas aboutir à centraliser tout le contrôle au niveau d'un seul objet. Il est nécessaire de veiller à l'autonomie comportementale des objets et à respecter autant que possible le faible couplage entre objets et la forte cohésion au sein des objets. Ceci n'est possible que si le contrôle est réparti entre les objets. La fonctionnalité globale du système serait alors assurée par une coopération entre objets (mécanismes de communication).

II.3.7.3 Les idées à retenir

Les travaux étudiés dans ce chapitre présentent certaines lacunes quant à l'efficacité des démarches exposées pour le passage entre spécifications fonctionnelles et spéci-

fications orientées objet. Cependant, certaines idées relevées sont très pertinentes et doivent être retenues dans une éventuelle démarche pour effectuer le passage. Retenons les moyens d'identification des objets, basés sur le diagramme de contexte, les stocks de données, les entités physiques du système et la possibilité de définir des objets conceptuels (non relatifs à une entité physique).

Pour la définition des objets nous retenons essentiellement la démarche de partitionnement et de regroupement dans les DFDs avec des principes forts tels que minimiser le couplage entre objets, favoriser la cohésion interne dans les objets et rechercher les processus concourants [Shumate 91]. Notons aussi que l'établissement d'un guide détaillé des étapes à entreprendre et des problèmes à résoudre tel qu'il a été présenté dans [George 96] est une façon très intéressante d'aborder la démarche.

II.3.7.4 Les contraintes à respecter

L'étude et l'analyse de plusieurs travaux traitant du passage entre spécifications fonctionnelles et spécifications orientées objet et leur adéquation aux problèmes posés nous permettent de dégager certaines contraintes à respecter pour aboutir à une démarche efficace et applicable aux systèmes temps réel avec un bon taux de réussite. Parmi ces contraintes citons essentiellement :

- La démarche à établir doit permettre d'identifier et de définir les objets en s'aidant le plus possible du travail d'analyse fonctionnelle préalablement entrepris sans refaire systématiquement l'analyse d'une manière orientée objet.
- Cette démarche ne doit pas ignorer l'aspect dynamique des systèmes. Une attention particulière doit être accordée à cet aspect quand il s'agit de systèmes temps réel à forte composante dynamique.
- En outre, elle ne doit pas aboutir à une spécification orientée objet à contrôle centralisé, dans laquelle un seul objet se charge du contrôle de tous les autres. Cette vision est contraire aux principes orientés objet. Il faut veiller à répartir le contrôle sur les objets concernés et allouer le maximum d'autonomie à ces objets (aspect comportemental de l'objet).
- La démarche à retenir ne doit pas s'appuyer exclusivement sur des considérations graphiques. Certes elle serait systématique et facilement automatisable. Or, il est nécessaire de pouvoir traiter les problèmes spécifiques au système étudié. Cette démarche doit permettre de prendre en compte le contexte du problème en s'appuyant sur les connaissances de l'analyste-concepteur.
- Enfin, le modèle fonctionnel et le modèle orienté objet utilisés doivent être adaptés aux systèmes temps réel. Ils doivent permettre d'exprimer des considérations statiques de structuration ainsi que des considérations dynamiques de contrôle du processus et des communications au sein du système et avec son environnement.

II.4 Propositions complémentaires

L'étude des techniques utilisées pour le passage entre spécifications fonctionnelles et spécifications orientées objet nous a fait penser à d'autres techniques complémentaires. D'abord, il s'agit d'une tentative d'application concrète d'un principe repris par toutes les publications : la maximisation de la cohésion interne et la minimisation du couplage entre objets. Ensuite, il s'agit d'essayer de combler une lacune propre à la plupart des techniques étudiées : pas de prise en compte de l'aspect dynamique pour le choix des objets.

II.4.1 Assurer une forte cohésion interne et un faible couplage entre les objets

Nous proposons ici une méthode basée uniquement sur les diagrammes DFDs, qui peut fournir à l'analyste-concepteur le moyen d'évaluer le critère de cohésion lors de la répartition des transformations de données sur les objets.

Lorsqu'un objet est défini en regroupant certains éléments des diagrammes SA-RT, cela revient à établir, au sein du diagramme, une frontière qui sépare les éléments participants à la définition de l'objet du reste des éléments du diagramme. Les éléments d'un diagramme (stocks, sources, transformations, puits) sont considérés, ici, sans aucune distinction. A cette séparation nous associons un paramètre de pénalité qui donne le nombre de liaisons rompues par la sélection de cet objet. Les liaisons rompues sont les flots de données et de contrôles qui sont traversés par la frontière établie. Plus les liaisons rompues sont nombreuses, plus le couplage de l'objet avec son environnement est important et plus la pénalité associée à la définition de cet objet est forte.

La figure *II.2* illustre cette mesure de la pénalité. Si un objet est constitué à partir des éléments e_1 et e_5, la pénalité associée à l'isolation de ces éléments est donnée par le nombre de flots traversés par la ligne de séparation F_2 c'est une pénalité égale à 1. En revanche, la pénalité associée à l'isolation de l'élément e_3 est égale à 5. Ainsi, il est fortement déconseillé d'isoler l'élément e_3.

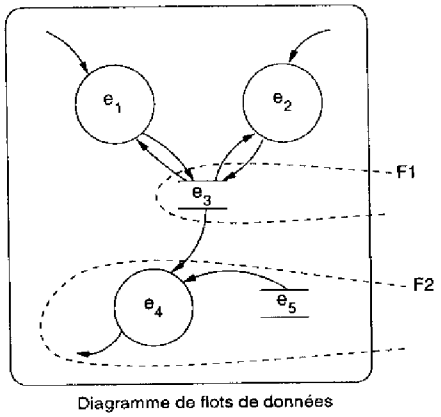


fig II.2 - Mesure de la pénalité

Nous proposons un algorithme permettant de calculer cette pénalité. D'abord, le diagramme étudié est caractérisé par sa **matrice des flots** M : elle donne le nombre de flots entre les différents éléments du diagramme.

Définition

Soit $E = \{e_i,\ i = 1..n\}$ l'ensemble des éléments du diagramme étudié et n le nombre total de ses éléments. La matrice de flots M est définie par :

$$M = [m_{ij}], \qquad i,j = 1..n$$

où m_{ij} est le nombre de flots entre les éléments e_i et e_j du diagramme. M est une matrice carrée, symétrique, le sens des flots n'est pas pris en compte : $m_{ij} = m_{ji}$.

Un objet Obj obtenu par la sélection d'un sous-ensemble d'éléments de E est caractérisé par le vecteur V :

$$V^T = [v_1..v_n], \qquad \begin{array}{l} v_i = 1 \quad si\ e_i \in Obj \\ v_i = 0 \quad si\ e_i \notin Obj \end{array}$$

soit W, le vecteur complémentaire de V qui caractérise l'environnement de l'objet Obj :

$$W^T = [w_1..w_n], \qquad w_i = 1 - v_i$$

■

La pénalité $p(Obj)$ associée à la définition d'un objet Obj caractérisé par le vecteur V, à partir d'un diagramme dont la matrice de flot est M, est donnée par l'expression :

$$p(Obj) = V^T.M.W$$

C'est le nombre de flot rompus par le choix de l'objet Obj. En effet,

$$p(obj) = V^T.M.W = \sum_{i,j}^{n} v_i m_{ij} w_j$$

soit $p_{ij} = v_i m_{ij} w_j$, donc :

$$p(obj) = \sum_{i,j}^{n} p_{ij}$$

p_{ij} ne peut prendre que les valeurs m_{ij} ou 0 :

$$\begin{array}{ll} p_{ij} = m_{ij} & ssi \qquad e_i \in Obj\ et\ e_j \notin Obj \\ p_{ij} = 0 & dans\ tous\ les\ autres\ cas. \end{array}$$

Ainsi, p_{ij} est la pénalité associée à la séparation de l'élément e_i de l'élément e_j, c'est le nombre de flots entre ces éléments. $p(obj)$ est le nombre total de flots rompus, c'est la pénalité associée au choix de l'objet Obj.

Cette méthode de calcul est facilement programmable et peut être intégrée à un environnement de génie logiciel pour avoir une mesure de la pénalité encourue lors de la définition des objets. Elle présente l'inconvénient de ne pas prendre en compte le contexte du problème. Elle peut être améliorée en pondérant chacun des éléments de la matrice des flots par un coefficient qui matérialise l'aspect critique de la rupture du flot

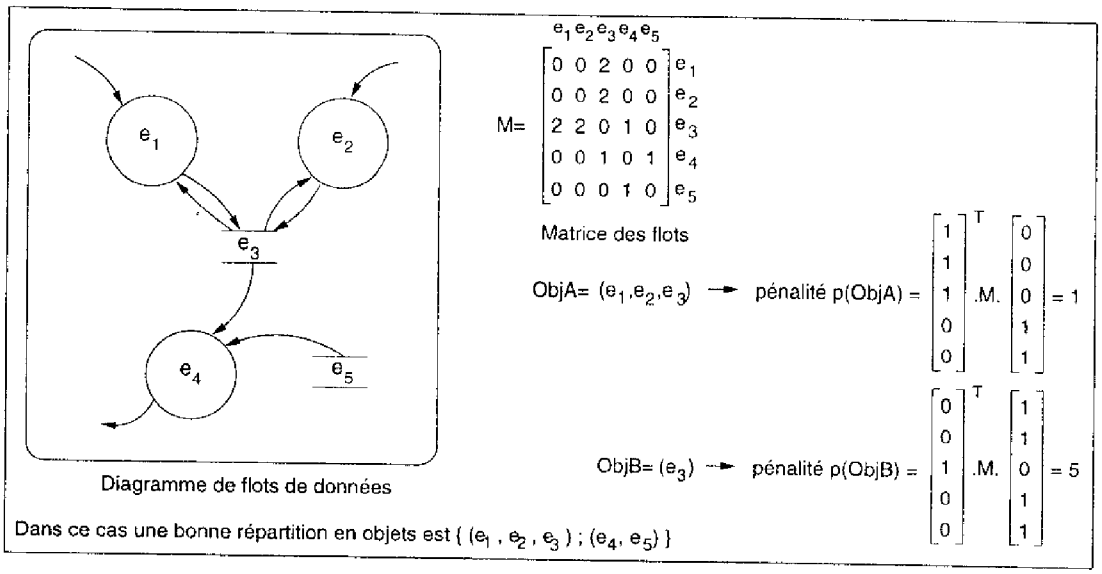


fig II.3 – Exemple de calcul des pénalités

correspondant. Ainsi, nous définissons une application qui associe à chaque flot f du diagramme étudié un coefficient de pondération $c(f)$ compris entre 0 et 1 qui mesure la cohésion entre les éléments du diagramme reliés par le flot f : plus les éléments reliés par le flot f sont jugés fortement reliés l'un à l'autre plus $c(f)$ tend vers 1. Il est évident que c'est une appréciation de l'analyste-concepteur basée sur sa connaissance du système. Dans ce cas les valeurs m_{ij} de la matrice des flots deviennent :

$$m_{ij} = \sum_{f \in F_{ij}} c(f) \ , \quad F_{ij} \textit{ étant l'ensemble des flots entre les éléments } e_i \textit{ et } e_j$$

Ainsi, cette méthode peut être intégrée dans une démarche de partitionnement et de regroupement des éléments d'un DFD. A l'aide de la pondération des flots, elle peut prendre en compte le contexte du problème et peut ainsi être utilisée comme une méthode à part entière pour la définition des objets. Alors, la démarche à suivre est la suivante :

Démarche

- Pour une hiérarchie de spécifications SA-RT donnée, mettre à plat tous les niveaux puis établir la matrice des flots pondérés.
- Calculer la pénalité associée au différents choix d'objets possibles.
- Choisir un objet qui présente une pénalité minimale et l'isoler. C'est un objet à forte cohésion interne faiblement couplé avec son environnement.
- Pour le reste du diagramme reprendre la même démarche (sachant que la nouvelle matrice des flots peut être déduite de la première en éliminant les lignes et les colonnes relatives aux éléments du DFD déjà alloué).
- Arrêter les recherches quand les éléments restants ne doivent pas être séparés (forte pénalité due à une forte cohésion).

■

Remarque: Si la transformation de contrôle est prise en compte comme un des éléments du diagramme à partitionner, tout le contrôle sera associé à un seul objet (objet contrôleur). Il est donc préférable d'appliquer cette méthode sur les DFDs (sans le contrôle) et lui associer une autre technique de répartition du contrôle.

La difficulté principale associée à cette technique est sa complexité algorithmique. En effet, la deuxième étape (recherche de la pénalité associée aux différents objets possibles) est une opération à complexité exponentielle ($\Theta(2^n)$). Il est possible de rechercher des méthodes d'exploration des solutions admissibles plus simples basées sur la technique de relaxation de contrainte *branch and bound*, mais le problème reste entier.

II.4.2 Une technique pour la répartition du contrôle

Parmi les techniques présentées pour le passage entre spécifications fonctionnelles et spécifications orientées objet, les démarches exposées dans [Shumate 91] ou [George 96] semblent bien fondées en ce qui concerne l'aspect statique des problèmes. Elles procèdent différemment mais s'appuient bien sur le contexte du problème étudié. En revanche, comme nous l'avons montré plus haut, les deux démarches sont limitées du point de vue dynamique, particulièrement quand il s'agit de traiter les systèmes temps réel. En fait, soit l'aspect dynamique n'est pas traité, soit le contrôle est centralisé.

Pour essayer de combler ce déficit dans ces démarches, qui sont par ailleurs assez pertinentes, nous proposons dans cette partie une technique simple pour répartir le contrôle entre les objets et favoriser ainsi leur autonomie. Nous considèrerons que l'aspect statique est décrit par un DFD et que l'aspect dynamique est décrit par un réseau de Petri (même si cela est fait par des machines à états finis, le passage vers des réseaux de Petri est assez facile) [Benzina et Paludetto 96b].

II.4.2.1 Présentation

Les techniques présentées plus haut se basent toutes sur les données et les transformations de données pour définir les objets. Supposons que les objets soient définis, il s'agit alors de répartir le contrôle du CFD sur les différents objets (voir exemple fig *II.4*, *II.5*, *II.6*). Il est nécessaire de continuer à se référer aux transformations de données pour effectuer cette répartition. Pour ce faire, nous proposons d'effectuer une "projection" des réseaux de Petri étudiés sur les différents objets : à chaque objet défini sera alloué un modèle réduit du réseau de Petri initial (associé à la transformation de contrôle des spécifications fonctionnelles) qui gère le fonctionnement des transformations de données de cet objet. Pour cela, nous proposons les étapes suivantes (dont la justification relève du bon sens) :

Démarche

- Pour chaque objet, considérer le réseau de Petri de la transformation de contrôle et marquer :
  - Les transitions qui portent des étiquettes relatives aux événements d'activation de l'une des transformations de données allouées à l'objet considéré.

- Les transitions qui portent des étiquettes relatives aux événements émis par l'une des transformations de données allouées à l'objet considéré et à destination de la transformation de contrôle.
- Les places ayant comme transitions d'entrée et de sortie des transitions déjà marquées.

– Distinguer les places de communication de l'objet considéré avec son environnement. Pour cela, dupliquer chaque place ayant en entrée ou en sortie une transition déjà marquée et la relier seulement à la transition marquée (places hachurées de la figure *II.4*).

– Utiliser les règles de réduction des réseaux de Petri [Brams 83], [Peterson 81] pour réduire les places et les transitions qui n'ont pas été marquées. ■

Cette méthode aboutit à la définition d'un réseau de Petri de contrôle pour chacun des objets du système à partir du réseau de Petri initial. Les transitions communes aux réseaux obtenus représentent des communications synchrones entre les objets. Les places communes représentent des communications asynchrones ou une ressource commune. Dans ce cas, il devient intéressant de voir si cette ressource doit servir à définir un nouvel objet (si ce n'est pas déjà fait).

L'exemple suivant montre une application de cette méthode. Par la suite, nous discuterons de ses limites.

II.4.2.2 Exemple d'application

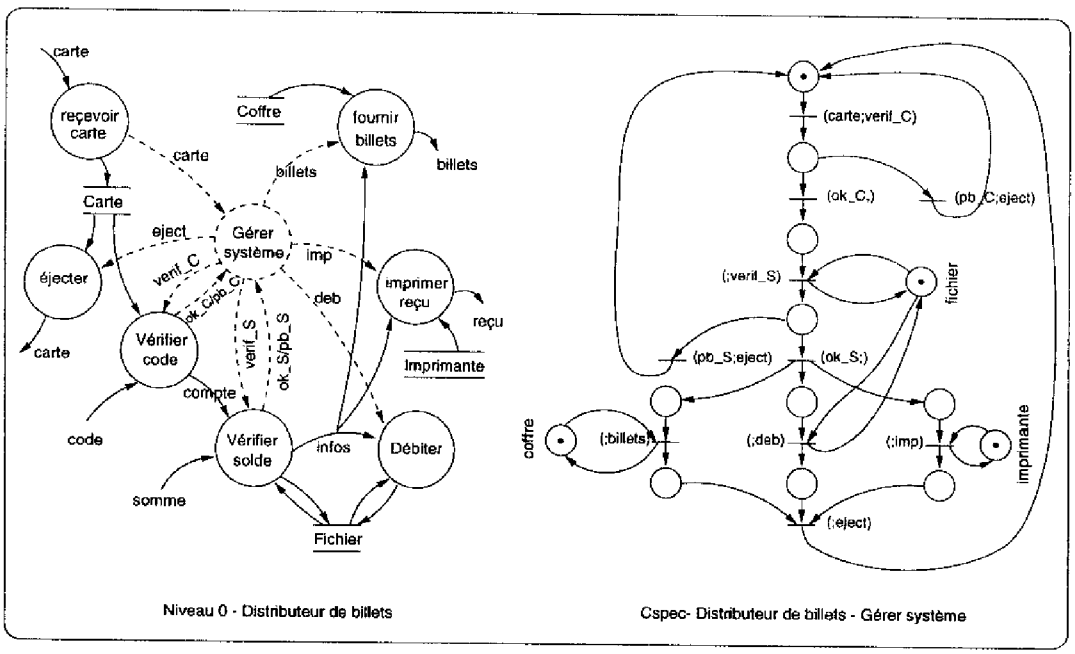


fig II.4 – Exemple

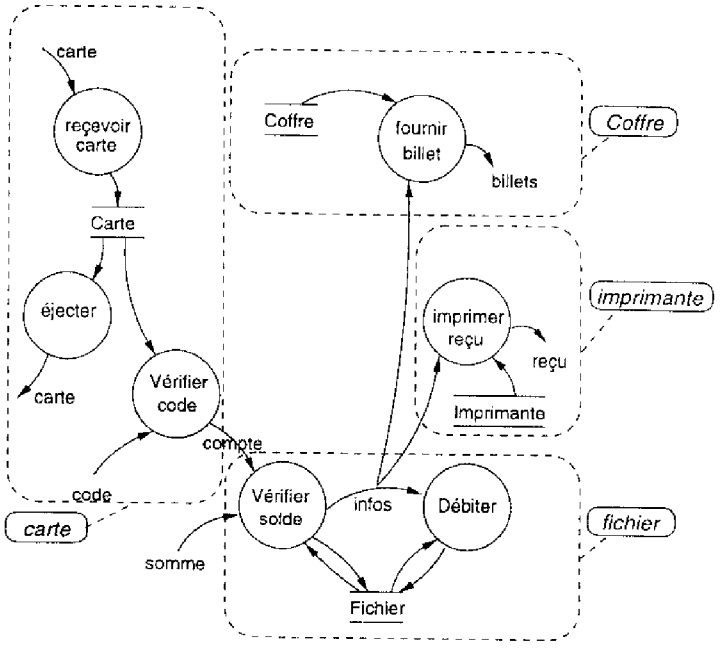


fig II.5 – Exemple - définition des objets

La figure *II.4* donne une partie des spécifications d'un distributeur de billets de banque simplifié. L'utilisateur introduit sa carte et donne son code, le distributeur

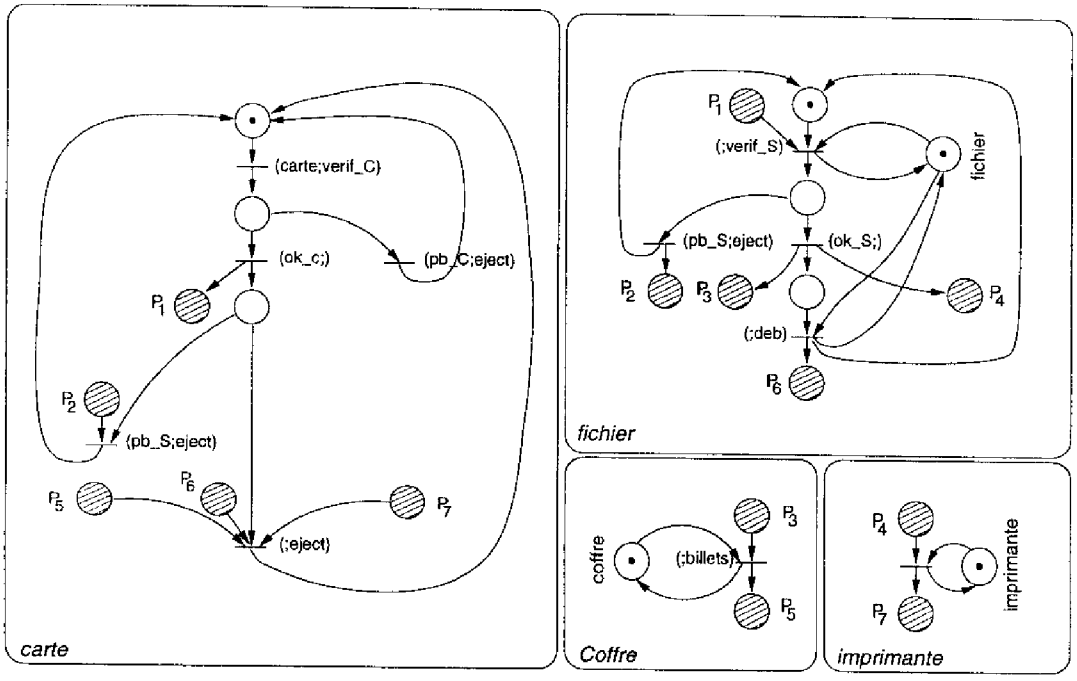


fig II.6 – Exemple - Répartition du contrôle

effectue alors les vérifications nécessaires et éjecte la carte en cas de problème. Si le code est bon, la machine vérifie la disponibilité de la somme demandée à partir du fichier client. En cas de disponibilité, le système fournit les billets, lance l'impression du reçu et met à jour le fichier client. Quand ces opérations sont achevées il éjecte la carte. En cas de non disponibilité de la somme, la carte est éjectée instantanément. La figure *II.4* donne les spécifications par un DFD et le réseau de Petri représentant les activités gérées par la transformation de contrôle.

Supposons que l'une des démarches proposées dans ce chapitre a été utilisée pour identifier et définir les objets à partir du DFD et qu'elle a abouti à la définition des objets : *Carte*, *imprimante*, *fichier* et *coffre*. La figure *II.5* montre la répartition des transformations de données sur ces objets. Procédons maintenant à la répartition du contrôle sur ces objets. En suivant la démarche explicitée plus haut, nous obtenons les réseaux de Petri de la figure *II.6* relatifs à chaque objet. Ainsi définis, les objets adhèrent mieux au concept d'encapsulation (données + opérations + comportement). Ils deviennent plus autonomes et certains, comme l'objet *fichier*, peuvent servir d'autres clients. Par ailleurs, les perspectives de réutilisabilité sont améliorées grâce à cette autonomie.

II.4.2.3 Limites

Cette méthode de répartition du contrôle présente certaines limites. Tout d'abord, le réseau de Petri représentant le comportement de chaque objet peut ne pas présenter une bonne cohésion comportementale a priori et le flux de contrôle peut être interrompus plusieurs fois à cause d'activités de communication. Un tel problème peut s'expliquer par le fait que la définition des objets a été basée sur les transformations de données (l'aspect statique). Cette méthode fonctionne bien quand l'aspect dynamique est assez réduit. Cependant, pour des systèmes temps réel où l'aspect dynamique est prépondérant, il faut rechercher d'autres techniques pour la définition des objets.

Par ailleurs, toujours pour les systèmes temps réel à forte composante dynamique, la méthode n'est plus efficace quand les transitions du réseau de Petri de contrôle du CFD traitent des événements de l'environnement (le procédé) qui ne sont rattachées à aucune des transformations de données. Dans ce cas, ces transitions n'apparaissent dans aucun des réseaux de Petri des différents objets. Ce type de transition dénote la présence d'objets actifs dont l'activité est axée sur les événements du procédé plutôt que sur les transformations de données du DFD. L'étude du DFD ne permet pas à elle seule de distinguer cette catégorie d'objet. Donc, il faut trouver une technique de recherche des objets qui permet de s'appuyer sur l'aspect dynamique quand cela est nécessaire.

II.5 Conclusion

Dans ce chapitre, une étude bibliographique nous a permis de présenter la polémique que suscite le passage entre approches fonctionnelles et approches orientées objet. Nous avons pu préciser les incompatibilités qui existent entre ces approches et présenter les travaux effectués dans ce domaine.

L'étude des techniques déjà utilisées pour résoudre ce problème nous a permis de dégager les idées qui nous semblent intéressantes et de bien cerner les lacunes existantes et les contraintes à respecter pour aboutir à une démarche efficace.

Ainsi, les techniques étudiées présentent certaines idées intéressantes mais ne sont pas toujours satisfaisantes quant à la pertinence des objets identifiés, l'exhautivité de ces objets, l'aspect systématique de la démarche ou la couverture des différents aspects du problème. Plus particulièrement, nous avons remarqué que la totalité des techniques proposées s'appuient exclusivement sur l'aspect statique pour effectuer le passage entre analyse fonctionnelle et analyse orientée objet. Ces techniques ne prennent en compte l'aspect dynamique ni pour l'identification des objets ni pour la répartition du contrôle. Pour pallier cette lacune, nous avons proposé une méthode de répartition du contrôle sur les objets définis à partir de la décomposition de l'aspect statique.

Ces techniques restent néanmoins peu satisfaisantes pour les systèmes temps réel qui possèdent une forte composante dynamique. En effet, pour ces systèmes certains objets sont plutôt axés sur des activités dynamiques et leur identification doit plutôt être basée sur une analyse de cet aspect.

L'environnement scientifique dans lequel nous nous trouvons, nous a permis d'entrevoir une approche basée sur l'utilisation des réseaux de Petri pour effectuer ce passage. En effet, il existe une certaine analogie entre DFDs et réseaux de Petri qui nous permettrait de représenter les aspects statiques et dynamiques à l'aide du même outil. Par la suite, les analyses et les recherches possibles sur les réseaux de Petri pourraient nous aider à identifier les objets intervenant dans la spécification. Dans le chapitre suivant, nous étudions la représentation des diagrammes SA-RT par réseaux de Petri. Ensuite, dans le chapitre IV, nous traiterons de l'utilisation des réseaux de Petri représentant les spécifications pour l'identification et la définition des objets.

Chapitre III

Représentation des diagrammes SA-RT à l'aide des réseaux de Petri

Ce chapitre traite de l'utilisation des réseaux de Petri pour la représentation des spécifications SA-RT. Nous étudions d'abord les travaux publiés dans ce domaine, puis, nous exposons les solutions adoptées pour la représentations des différents éléments des diagrammes SA-RT. Avant d'aborder la présentation, rappelons que les diagrammes SA-RT donnent seulement une partie des informations rassemblées lors d'une analyse des besoins par la méthode SA-RT. En outre, la représentation de ces diagrammes par les réseaux de Petri ne traduit pas tous les points de vue. Ainsi, le résultat obtenu ne sera qu'une vue partielle des spécifications. Il est donc important de veiller à garder les informations les plus pertinentes pour notre objectif final.

III.1 Utilité de la représentation par réseaux de Petri

La méthode SA-RT est très performante comme outil de spécification particulièrement sur l'aspect statique des systèmes. Par ailleurs, les documents fournis à l'issue de l'étude sont aussi clairs qu'explicites. Il est alors légitime de se demander pourquoi vouloir adopter une autre représentation.

En fait, le but recherché n'est pas de substituer les réseaux de Petri aux diagrammes SA-RT mais plutôt de les utiliser comme complément à la méthode. D'abord, les réseaux de Petri sont très utiles pour représenter les évolutions dynamiques dans les spécifications (le contrôle). Nous avons montré l'intérêt de cette utilisation conjointe

dans [Benzina et Paludetto 96b]. Ensuite, en ce qui concerne l'aspect statique, les réseaux de Petri peuvent intervenir dans une étape ultérieure à l'analyse des besoins par SA-RT, ils serviront à exprimer le résultat des spécifications SA-RT. Les possibilités offertes par cette représentation peuvent être résumées ci-dessous :

- Le prototypage des systèmes étudiés.

  Étant donné leur aspect formel et les outils relativement nombreux qui les supportent, il est plus facile et plus intéressant de réaliser un prototype à partir d'une description par réseaux de Petri qu'à partir d'une spécification SA-RT. Plusieurs travaux ont été publiés dans ce sens : [Pulli *et al.* 88], [Fuggetta *et al.* 93] et plus particulièrement les travaux de [Elmstrom *et al.* 93b] poursuivis par [Shi et Nixon 96].

- La formalisation des spécifications.

  L'aspect le plus contesté des spécifications SA-RT est leur aspect informel. En effet, SA-RT est une méthode basée essentiellement sur l'intuition de l'analyste et sur son sens de la logique. Plusieurs travaux ont été entrepris pour doter les spécifications SA-RT d'un aspect formel : [France 92], [Elmstrom *et al.* 93a], [Giron 96], etc. Les réseaux de Petri ont également été considérés pour amener des bases formelles aux documents fournis par les spécifications SA-RT : [Tse et Pong 89], [Lee et Tan 92], [Richter et Maffeo 93], [Elmstrom *et al.* 93b].

- La validation des spécifications.

  L'analyse et la validation des réseaux de Petri ont fait l'objet de plusieurs travaux de recherche, les résultats publiés à ce jour permettent de déceler les défauts de spécification apparaissant lors de la description des systèmes (blocages, parties non vivantes,...). L'exploitation de ces résultats pour la validation des spécifications SA-RT représente une motivation importante. En particulier, cette validation permettrait d'éviter la propagation des erreurs commises lors de l'étape de l'analyse des besoins, vers les étapes ultérieures. Les travaux qui ont étudié cet aspect sont peu nombreux [Tse et Pong 89], [Lee et Tan 92].

- L'aide au passage à des spécifications orientées objet.

  Une dernière application, non explorée encore, qui fait l'objet de notre mémoire, est l'utilisation des réseaux de Petri comme moyen d'aide au passage entre spécifications SA-RT (méthode fonctionnelle) et spécifications orientées objet. En effet, connaissant l'existence de nombreux outils d'analyse structurelle et de recherche de propriétés des réseaux de Petri, il est intéressant de les utiliser pour faciliter ce passage.

La variété de ces objectifs a engendré des différences dans l'utilisation des réseaux de Petri pour la représentation des spécifications SA-RT.

III.2 Les différentes stratégies de représentation par réseaux de Petri

Les travaux publiés dans ce domaine ont principalement pour buts le prototypage et la formalisation. Certains aspects de la validation ont été étudiés sans qu'il y ait eu de résultat probant pour une validation globale des diagrammes SA-RT. Ces travaux peuvent être classés selon le type de représentation par réseaux de Petri. Nous exposons, dans ce paragraphe, trois travaux caractéristiques de ces différentes représentations.

III.2.1 Formalisation des DFDs par l'algèbre des réseaux de Petri

Tse et Pong ont publié un article en 1987 sous le titre *Towards a Formal Foundation for DeMarco Data Flow Diagrams* [Tse et Pong 89] dans lequel les auteurs ont redéfini les diagrammes de flots de données sous formes algébrique et graphique. La forme graphique est celle connue des DFDs et la forme algébrique est empruntée aux réseaux de Petri. La figure *III.1* montre un exemple de cette représentation.

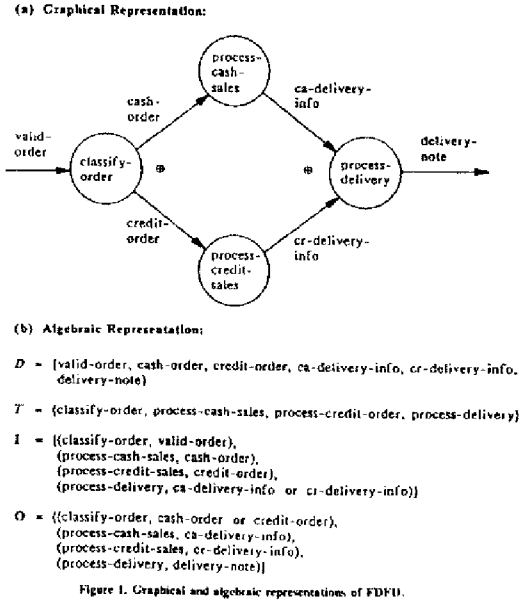


fig III.1 – Exemple d'un FDFD selon [Tse et Pong 89]

Ce modèle est appelé FDFD (Formal Data Flow Diagram). Par ailleurs, les auteurs ont doté les FDFD d'une notion de marquage et de règles d'évolution de ce marquage (la transformation marquée est la transformation active). Les règles d'évolutions traduisent en réseaux de Petri les recommandations faites dans [Ward 86]. Ainsi, dans ce type de représentation la structure du DFD est gardée mais son interprétation est formalisée et limitée par des contraintes provenant de l'utilisation des règles d'établissement et d'évolution des réseaux de Petri.

Pour définir les FDFD, Tse et Pong ont introduit des extensions relativement à la forme classique des réseaux de Petri, principalement des règles de production AND et OR à l'entrée et à la sortie des transitions. Ces extensions rendent problématique l'utilisation des propriétés des réseaux de Petri classiques.

Par ailleurs, les différents éléments pouvant intervenir dans un DFD et les interprétations pouvant en découler n'ont pas tous été examinés. Les auteurs se sont simplement limités aux flots de données et aux transformations de données, ce qui représente une composante importante des DFDs mais ne suffit pas pour étudier la majorité des problèmes. Ainsi, il est possible d'établir de nouveaux FDFD en se limitant aux élément de représentation fournis mais il est difficile de «formaliser» des DFDs déjà établis comprenant des éléments non utilisés par les FDFD. Remarquons aussi que l'aspect dynamique des spécifications SA-RT n'a pas été considéré.

III.2.2 Modélisation du comportement des éléments des diagrammes SA-RT

Dans le cadre du projet IPTES [Pulli et Elmstrom 93], une étude a été faite sur l'utilisation du formalisme des réseaux de Petri pour fournir un support aux spécifications SA-RT : *Giving Semantics to SA-RT by Means of High-Level Timed Petri Nets* [Elmstrom *et al.* 93b]. Dans cette étude, les auteurs étaient motivés par le prototypage des systèmes temps réel. Ils se sont plutôt intéressés à la modélisation du comportement des divers éléments des diagrammes SA-RT, de telle sorte que les réseaux de Petri obtenus modélisent l'état des transformations de données (actives ou au repos) et l'état des flots et des stocks de données (libres ou occupés). La figure *III.2* illustre un exemple de représentation adoptée.

Cette représentation a pour but d'aboutir à des diagrammes SA-RT exécutables. L'aspect réseau de Petri est caché par rapport à l'utilisateur et sert à faire évoluer la simulation des diagrammes SA-RT. Ainsi, les auteurs arrivent à combiner la simplicité de représentation de SA-RT avec le formalisme et la simplicité d'exécution des réseaux de Petri.

Les auteurs de cette étude ont abouti à des résultats très intéressants dont :

- l'adoption de règles de translations locales, de façon qu'à chaque élément des diagrammes SA-RT corresponde un module réseau de Petri. Ainsi, la composition de deux ou plusieurs éléments SA-RT se traduit par la composition (juxtaposition) des modules de réseaux de Petri correspondants.
- La possibilité d'utiliser la même méthode pour effectuer la translation inverse, réseaux de Petri vers SA-RT (l'idée est assez audacieuse vu le manque de formalisme dans SA-RT).
- Par ailleurs, l'utilisation des HLTPN (*High-Level Timed Petri nets* définis dans [Felder *et al.* 93] et [Ghezzi *et al.* 91]) permet d'intégrer la représentation du temps au niveau des informations associées au jeton.

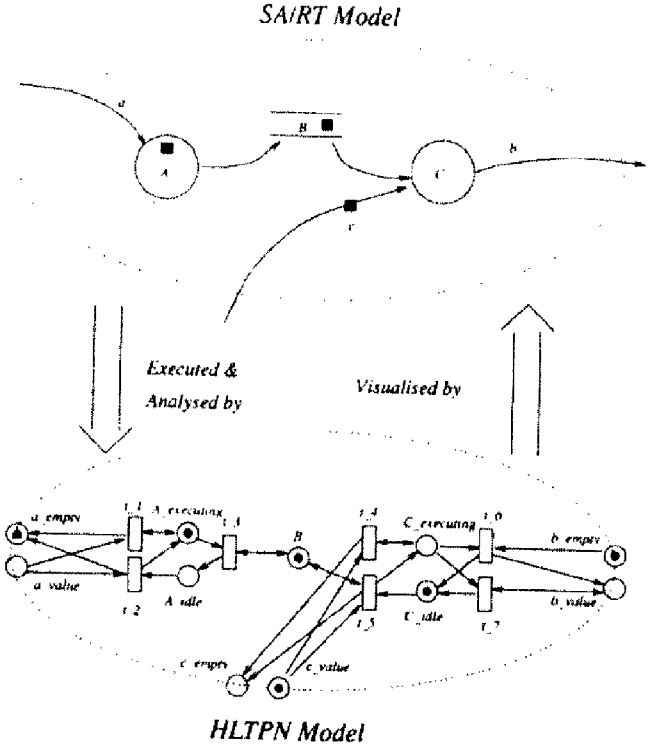


fig III.2 – Modélisation du comportement des éléments des diagrammes SA-RT selon [Elmstrom et al. *93b]*

Cependant, comme nous l'avons mentionné plus haut, les règles de translation présentées modélisent plutôt l'état des éléments du diagrammes SA-RT. Ceci revient à une sorte de description des mécanismes de fonctionnement des DFDs et le résultat obtenu ne tient plus compte du système physique qui a été analysé par SA-RT. En outre, ce type de représentation induit une augmentation rapide de la taille des réseaux obtenus. Un exemple simple est présenté (figure *III.2*) dans lequel un DFD avec deux transformations, cinq flots et un stock de données est représenté par un réseau de Petri ayant onze places et sept transitions! L'étude menée par [Shi et Nixon 96] montre que la taille du réseau de Petri obtenu peut croître exponentiellement dans certains cas.

Par ailleurs, les auteurs ne traitent pas la représentation de la structuration hiérarchique adoptée dans les diagrammes SA-RT; ils partent toujours d'un modèle à plat. Il nous semble que cette simplification constitue une grande perte pour l'analyste car la technique de structuration hiérarchique de la méthode SA-RT est un des aspects les plus intéressants. En outre, la mise à plat augmente considérablement la taille du réseau de Petri obtenu.

Cette étude a été poursuivie par les travaux de [Shi et Nixon 96] qui ont amélioré la représentation par réseau de Petri. Des améliorations ont été notamment portées sur la modularité de la représentation et sur la réduction de la complexité lors de la représentation de l'interaction entre le réseau de Petri représentant la machine à état et

celui représentant le DFD. L'aspect hiérarchique n'est toujours pas pris en compte et l'objectif principal reste la modélisation de l'état des éléments des diagrammes SA-RT.

III.2.3 Représentation de la circulation des données

Dans [Lee et Tan 92], les auteurs se sont intéressés à la modélisation des DFDs par les réseaux de Petri. En fait, ils présentent un environnement de génie logiciel dans lequel les spécifications sont modélisées et visualisées par les DFDs et peuvent être représentées par réseaux de Petri pour des objectifs de formalisation et de validation des spécifications.

Dans cette représentation, les sources, puits, stocks et flots de données sont représentés par des places et les transformations de données par des transitions (simples ou agrégées). Ainsi la circulation du jeton dans les réseaux de Petri représente la circulation des données dans les DFDs. Les auteurs répartissent les places sur quatre sous-ensembles relatifs aux sources, puits, stocks et flots. La validation consiste à faire des vérifications des «contraintes d'intégrité» dans un même niveau et entre les niveaux hiérarchiques successifs.

Ce type de représentation, axé autour de la circulation des données, semble être plus indiqué pour nos objectifs. Cependant, telle qu'elle a été présentée dans [Lee et Tan 92], cette représentation semble assez sommaire. Il faut l'étayer, la consolider et la compléter par les aspects dynamiques qui n'ont pas été considérés.

III.3 Objectifs et difficultés

Pour arriver à représenter les diagrammes SA-RT à l'aide des réseaux de Petri, il est impératif de prendre en considération les deux aspects, statique et dynamique, de la méthode SA-RT.

L'aspect dynamique des diagrammes SA-RT constitue la partie la plus aisée à représenter par réseaux de Petri. Ce travail a fait l'objet d'un article présenté dans la conférence *modélisation des systèmes dynamiques* de l'Afcet [Benzina et Paludetto 96b]. A l'issue de cette étude nous avons abouti à une méthodologie qui utilise conjointement SA-RT et les réseaux de Petri. Il a été possible d'allier la capacité de structuration de SA-RT pour l'aspect statique des systèmes temps réel et le formalisme et la puissance de description des réseaux de Petri pour le contrôle des systèmes.

La représentation de l'aspect statique des diagrammes SA-RT est plus difficile à entreprendre pour deux raisons essentielles :

- Les réseaux de Petri constituent un outil formel dont l'interprétation est unique alors que SA-RT est une méthode semi-formelle (informelle pour les puristes) et dont les notions restent variables selon la version de la méthode et selon les habitudes des utilisateurs.

- La méthode SA-RT permet de prendre en compte les données et les structures de données grâce à des outils autres que les diagrammes alors que les réseaux de Petri sont limités de ce point de vue. Il est certainement possible de recourir aux réseaux de Petri de hauts niveaux (à objets ou colorés), mais cette représentation apporterait des complications supplémentaires. Dans la mesure du possible, nous préférons utiliser les réseaux de Petri sous une forme simple (réseaux de Petri interprétés) pour pouvoir profiter des outils d'analyses disponibles.

Objectifs

Pour représenter les diagrammes SA-RT par des réseaux de Petri, nous nous proposons de respecter, autant que faire se peut, les critères suivants :

- Rester le plus près possible de la structuration SA-RT pour conserver la vue globale que donne les spécifications SA-RT du système étudié.
- Conserver l'aspect hiérarchique des spécifications SA-RT, cet aspect contribuera à atteindre l'objectif précédant et permettra de procéder à l'étude et la validation de chaque niveau à part.
- Associer à chaque élément (ou module) des diagrammes SA-RT un élément (ou module) de réseaux de Petri. Ainsi, le passage pourrait se faire d'une façon incrémentale et les modifications seraient possibles sans remettre en cause la totalité de la représentation, d'autant plus que, lors de l'étape d'analyse des besoins, les modifications sont plutôt fréquentes.
- Considérer la possibilité d'une automatisation du processus de passage entre SA-RT et réseaux de Petri.

■

III.4 Représentation des DFDs par réseaux de Petri

Rappelons que nous ne cherchons pas à modéliser le fonctionnement des DFDs, nous cherchons plutôt à établir une *analogie* entre DFDs et réseaux de Petri. Notre démarche est issue de la constatation de l'existence d'une certaine analogie entre la circulation des données dans les DFDs et la circulation des jetons dans les réseaux de Petri. En effet, en passant aux réseaux de Petri, les transformations de données peuvent être traduites par des opérations sur les transitions et les flots de données par des places par lesquelles transitent les jetons. Ainsi, la circulation des jetons dans le réseau de Petri obtenu sera analogue à la circulation des données dans le DFD de départ.

Alors, nous introduisons aux DFDs une dynamique relative au mouvement des données entre les transformations. Nous supposons que le temps d'exécution d'une transformation est nul. Cette hypothèse écarte momentanément les problèmes dus aux durées d'activité des transitions. Une transformation est activée dès qu'il y a suffisamment de données sur ses flots entrants. Immédiatement, la transformation a lieu et des données sont présentées sur les flots sortants.

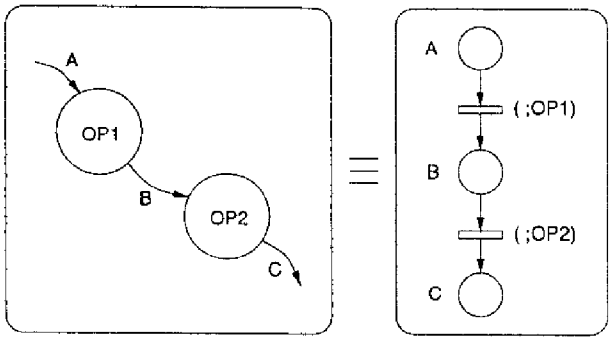


fig III.3 – Dualité entre DFD et RdP

Dans la suite de ce chapitre, sont exposés les détails de représentation de chacun des éléments des DFDs.

III.4.1 Sources et puits de données

Une source de données modélise une entité physique du procédé qui fournit en permanence des informations au système (Ex. un capteur de vitesse). Par opposition, un puits de données est une entité qui consomme en permanence des informations produites par le système.

Proposition

Une source ou un puits de données, est représenté par une ressource toujours disponible [1] pour produire ou consommer les données (voir figure *III.4*). Lorsqu'une entité présente les deux aspects source et puits, il est parfois intéressant de la dupliquer pour mettre en évidence ses deux fonctions. ■

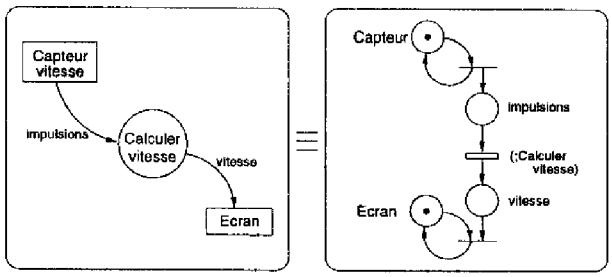


fig III.4 – Représentation des sources et des puits de données

1. Si les besoins imposent une disponibilité conditionnelle ou temporelle, il est toujours possible de prendre en compte cette contrainte au niveau de la transition consommatrice de la ressource.

III.4.2 Stocks de données

Dans les DFDs, les stocks de données permettent de représenter une information disponible en permanence dans le système, c'est la fonction mémoire.

Tout d'abord, il faut clarifier l'utilisation faite du symbole graphique stock de données (les deux segment horizontaux parallèles). Dans certains document SA-RT, cette notation est utilisée indifféremment pour schématiser une mémoire ou un buffer. Cette confusion n'est pas gênante du moment que la notion stock de données est assez générale pour englober les deux cas (c'est la vision HP [Hatley et Pirbhai 87]). Cependant, pour formaliser les notations et pour éviter les confusions, il est préférable de se conformer à la notation Ward et Mellor [Ward et Mellor 85] qui fait la distinction entre stock de données (mémoire) et buffer (pile). Ce dernier cas sera étudié dans le paragraphe suivant.

Ainsi, utilisé comme mémoire, le stock contient une donnée toujours disponible et modifiable. La lecture de cette donnée n'est pas destructrice et l'écriture dans le stock n'engendre pas la création d'une autre donnée, mais simplement la modification de la première. L'écriture est donc destructrice. Dans ce cas, le stock peut être interprété comme une ressource partagée ce qui est facilement représentable par les réseaux de Petri.

Proposition

Un stock de données est représenté par une ressource partagée (voir figure *III.5*). ■

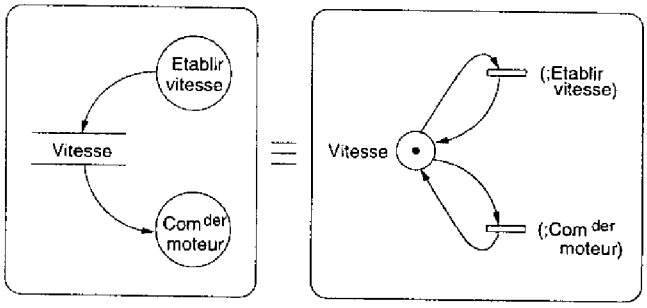


fig III.5 – Lecture non destructrice, écriture destructrice

Dans la figure *III.5*, nous représentons une opération d'écriture et une opération de lecture. Remarquons qu'il ne faut pas dissocier le réseau de Petri du DFD car le DFD représente son interprétation. Sans le DFD, le réseau de Petri ne présente aucune différence entre les opérations de lecture et d'écriture. Par ailleurs, s'il est absolument nécessaire de différencier ces opérations sur le réseau de Petri, il est possible de désagréger la transition représentant l'opération pour préciser le moment auquel la donnée est extraite du stock : au début de l'opération (lecture) ou à la fin de l'opération (écriture).

III.4.3 Les buffers

Les buffers sont utilisés pour représenter des files ou des piles. La lecture dans un buffer est destructrice. En effet, chaque fois qu'il y a une lecture, un élément du buffer est enlevé. En revanche, l'écriture est non destructrice. Un nouveau élément ajouté au buffer n'affecte pas les anciens.

Proposition

Un buffer est représenté simplement par une place. Les jetons se trouvant dans cette place, représentent les éléments du stock (voir figure *III.6*). ■

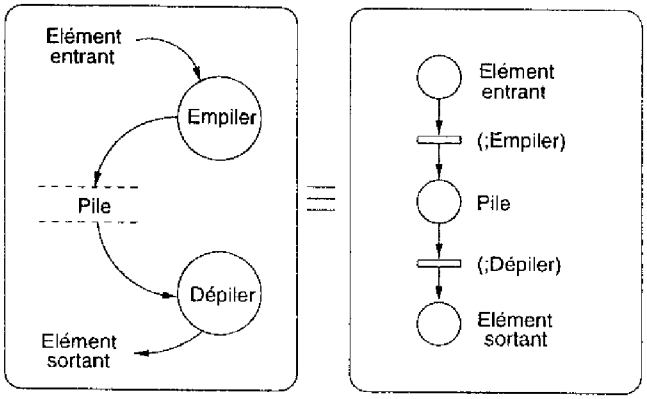


fig III.6 - Lecture destructrice, écriture non destructrice

L'utilisation des buffers n'est pas généralisée dans les spécifications SA-RT. Quand les stocks et les buffers sont représentés avec la même notation, il faut examiner la nature des opérations de lecture et d'écriture (destructrices ou non) pour aboutir à la bonne représentation par réseaux de Petri. Il arrive aussi que la notation *buffer* soit utilisée pour stocker des événements, ce cas est facile à distinguer.

III.5 Exemple préliminaire

Illustrons par un exemple l'utilisation des éléments représentés plus haut. Il s'agit d'un système qui mesure la température interne et externe d'un bâtiment puis calcule et affiche l'écart de température. Seul le niveau 0 est explicité ci-dessous (figure *III.7*). Les sources et puits de données relatifs au diagramme de contexte ont été reportés sur le réseau de Petri correspondant au niveau zéro.

III.6 Les problèmes de représentation

Les paragraphes suivants sont consacrés à la discussion des différents problèmes rencontrés lors de la représentation des diagrammes DFDs par des réseaux de Petri.

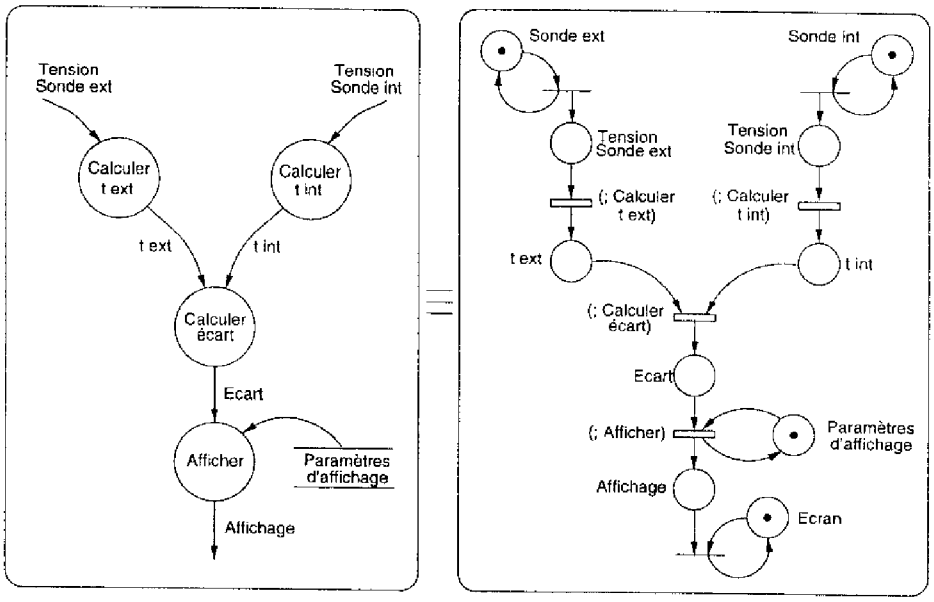


fig III.7 – Exemple

III.6.1 Mécanisme de rafraîchissement des données dans les flots

L'examen approfondi de l'exemple précédent (figure *III.7*) montre que si les deux sondes produisent l'information à des cadences différentes, une accumulation de jetons dans l'une des places de synchronisation est possible. Ce problème est inhérent à la représentation adoptée par réseau de Petri et ne se pose pas avec les DFDs. En effet, pour les DFDs une seule donnée peut être présente dans un flot de données. Dans le cas où une nouvelle donnée est produite à l'entrée du flot alors que l'ancienne n'a pas encore été consommée, le formalisme SA-RT spécifie que la nouvelle donnée doit remplacer l'ancienne : c'est le rafraîchissement des données.

La représentation du flot de données simplement par une place ne prend pas ce mécanisme en compte. En effet, dans un cas pareil une accumulation de jetons se produit dans la place qui représente le flot de données. Il faut alors prévoir un mécanisme qui prenne en compte cet aspect des flots de données.

Le modèle de flot de données présenté dans la figure *III.8* permet de résoudre le problème. Le flot est représenté par deux places selon son état d'occupation (libre ou occupé). Lorsque le flot est occupé et qu'une nouvelle donnée se présente, le jeton est alors réactualisé (mise à jour de la donnée).

Toutefois, cette représentation se rapproche plus de la représentation définie dans [Elmstrom *et al.* 93b] que de la structure initiale du modèle DFD. Elle modélise l'état des différents éléments des DFDs plutôt que la circulation des données entre eux. Avec cette représentation la taille du réseau de Petri résultant augmente rapidement et la lisibilité globale du modèle en souffre.

Un autre moyen d'éviter le problème de l'accumulation des jetons aux sorties des

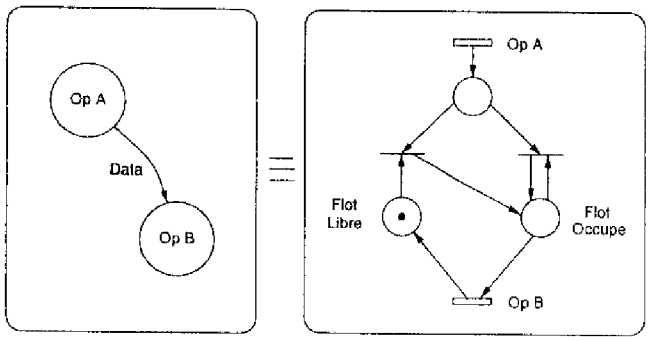


fig III.8 - Mécanisme de rafraîchissement des données

transformations est de limiter à un seul jeton la capacité du flot (figure *III.9*). C'est comme l'utilisation d'un arc inhibiteur entre la place représentant le flot et la transition représentant l'opération. L'existence d'une donnée dans le flot inhibe l'opération en amont.

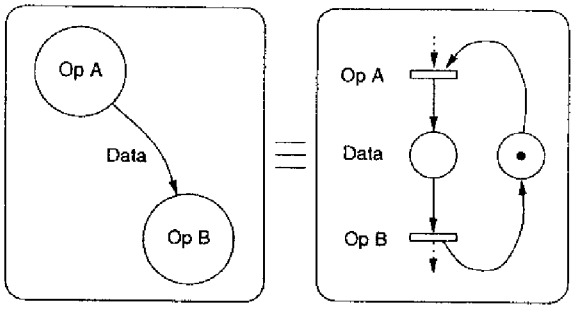


fig III.9 - Limiter la capacité des flots

L'inconvénient de cette représentation est que la transformation de données productrice est bloquée jusqu'à libération du flot, ce qui ne correspond pas exactement à ce qui se passe dans le diagramme SA-RT.

L'expérience montre qu'il n'est pas toujours nécessaire de mettre en place ce mécanisme de rafraîchissement des données. Ce mécanisme est intéressant lorsque le but recherché est la simulation des activités des DFDs ou le prototypage des systèmes. Dans ce cas, l'utilisateur doit faire son choix parmi les possibilités présentées en fonction du but recherché à travers l'utilisation des réseaux de Petri. Pour ce qui est de l'application qui nous intéresse, la représentation de la circulation des données, il n'est pas nécessaire de représenter ce mécanisme de rafraîchissement des données. En général, le risque encouru, si cet aspect des DFDs est ignoré, est l'aboutissement à un réseaux de Petri non borné. C'est au concepteur d'apprécier si cet aspect constitue un problème ou non pour l'utilisation qu'il va faire de la représentation par réseaux de Petri.

III.6.2 Règles de production et de consommation des données

Le non formalisme adopté par les différentes versions de la méthode SA-RT laisse aux analystes une liberté d'utilisation assez large. Particulièrement, l'utilisation des données par les transformations présente certaines ambiguïtés. En effet, les DFDs ne spécifient pas comment les données sont consommées ou produites lorsqu'il y a plusieurs flots entrant ou sortant des transformations. Un même diagramme peut être interprété de manières différentes selon le contexte (voir figure *III.10*).

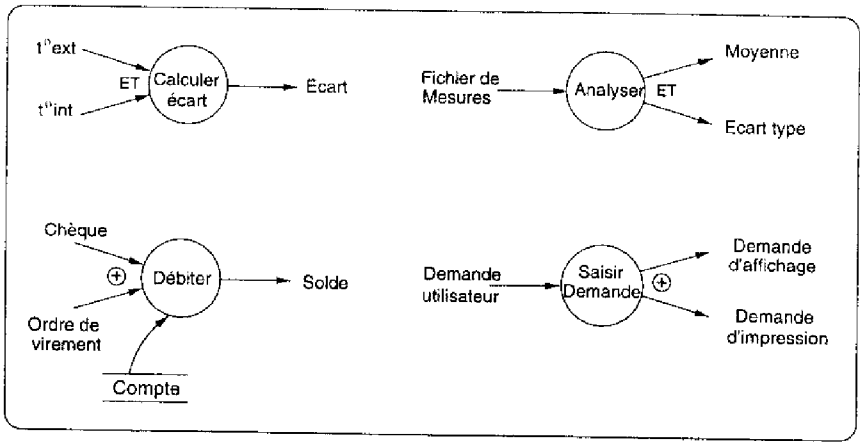
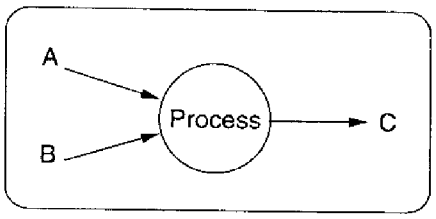


fig III.10 – Différentes interprétations d'un même DFD

Le DFD de la figure *III.11* indique que la transformation de données traite les données A et B. Cependant ce DFD ne précise pas s'il faut absolument que les données A et B soit présentes pour que la donnée C puisse être produite ou si la présence de l'une d'elles suffit. Le même problème peut se poser également à la sortie des transformations. Ceci se traduit par l'utilisation d'une fonction logique des entrées ou des sorties (A ET B), (A OU B) et (A $\oplus$ B). Nous remarquons qu'en pratique les cas les plus utilisés sont le (ET) pour les flots d'entrée et de sortie et le (OU exclusif) pour les flots de sortie.



Pour produire la donnée C, la transformation de données nécessite-t-elle la présence des données A et B simultanément, ou la présence de l'une des deux exclusivement ?

fig III.11 – Ambiguïtés du DFD

Dans leur ouvrage consacré à la méthode d'analyse structurée, Ward et Mellor ont

soulevé ce problème et ont établi des restrictions dans la représentation des diagrammes pour qu'un DFD ait une interprétation unique [Ward et Mellor 85] :

- Chaque transformation doit avoir au maximum une seule entrée discrète.
- Si l'entrée est une entrée composée (jonction), tous ses éléments doivent être présents pour que la transformation puisse opérer.
- Il est possible d'avoir zéro, une ou plusieurs sorties discrètes.
- S'il y a plusieurs sorties, ces sorties doivent être des alternatives et au maximum une seule doit être produite par chaque transformation (OU exclusif).

Les règles de Ward et Mellor s'appliquent aux flots de données discrets sauf aux flots joignant les stocks de données. Ces règles lèvent l'indétermination qui existe mais réduisent le pouvoir d'expression des DFDs. En effet, il n'est plus possible d'avoir le OU exclusif en entrée.

D'autres chercheurs préfèrent plutôt spécifier explicitement, sur les transformations, la nature de l'expression logique utilisée [Tse et Pong 89]. Cette méthode ne réduit pas le pouvoir d'expression des DFDs mais ajoute une préoccupation supplémentaire (simple) aux analystes. Dans la plupart des cas, connaissant le contexte du problème, il est assez facile de comprendre intuitivement (ou en examinant les mini-spécifications des transformations de données) la nature de l'opération requise.

Dans l'approche utilisée dans ce mémoire, nous optons plutôt pour cette deuxième représentation pour trois raisons essentielles :

- L'utilisation courante des DFDs montre que rares sont les diagrammes qui respectent les restrictions faites par Ward et Mellor. Ces restrictions sont assez contraignantes pour une méthode qui se veut généraliste et d'un abord facile.
- Connaissant le contexte du problème, il est assez facile de retrouver la nature des opérations logiques requises.
- L'utilisation des flots de données avec les restrictions de Ward et Mellor devient un cas particulier du modèle, plus général, utilisant la spécification par opérations logiques.

Proposition

Les règles de consommation et de production de données dans les DFDs sont précisées par des opérateurs logiques. Les divers cas sont modélisés par réseaux de Petri comme indiqué dans la figure *III.12*. ■

Remarques : L'opération logique ET est représentée par une jonction, cette représentation est plus proche des symboles utilisés dans les DFDs. Par ailleurs, la représentation du $\oplus$ (ou exclusif) engendre la création d'un conflit structurel. Ce conflit est levé par l'interprétation (DFD).

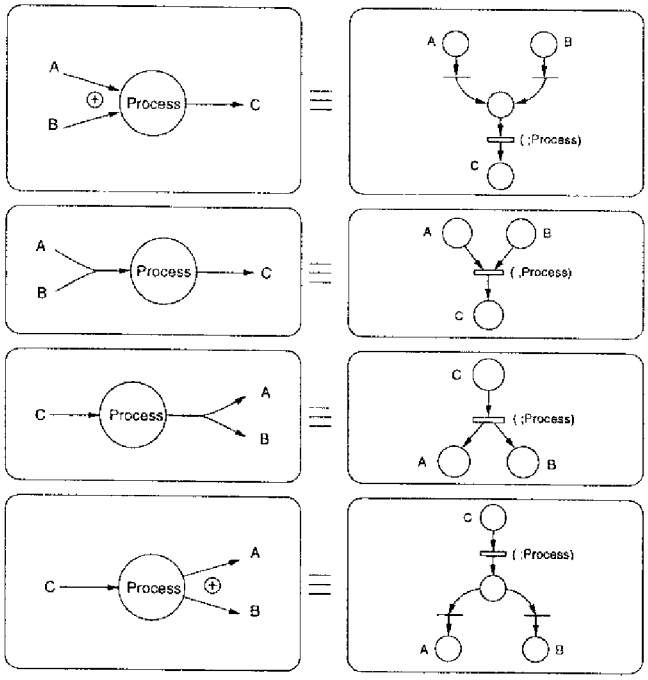


fig III.12 – Règles de consommation et de production des données

III.7 Représentation des différents niveaux hiérarchiques

La hiérarchie adoptée par les DFDs est une hiérarchie d'abstraction. C'est incontestablement l'un des apports majeurs de la méthode SA-RT. Cette hiérarchie permet d'avoir une vision claire du système étudié, une visibilité difficile à obtenir avec les réseaux de Petri.

Pour représenter les différents niveaux hiérarchiques des diagrammes DFDs, il s'avère très intéressant de s'appuyer sur l'idée de substitution de blocs de réseaux de Petri bien formés, décrite dans [Valette 76] et [Valette 79].

En résumé, un bloc de réseau de Petri est un réseau de Petri qui possède une seule transition initiale et une seule transition finale. Un bloc de réseau de Petri bien formé est un bloc pour lequel les propriétés *borné et vivant* sont vérifiées. Un tel bloc peut être remplacé par une transition, sans que la propriété *bien formé* du réseau de Petri global ne soit altérée. Schématiquement, la substitution des blocs bien formés est une agrégation de parties de réseaux de Petri. Cette technique introduit une vision hiérarchique dans les réseaux de Petri (au sens de l'inclusion et non de l'utilisation) pour augmenter leur lisibilité sans pour autant perdre leur formalisme logique.

Ainsi, la transition représentant une transformation de données SA-RT à un niveau i peut être l'agrégation du bloc de réseau de Petri associé au diagramme SA-RT de niveau $i+1$ détaillant cette transformation.

Une seconde possibilité consiste à substituer, dans le réseau de Petri représentant le

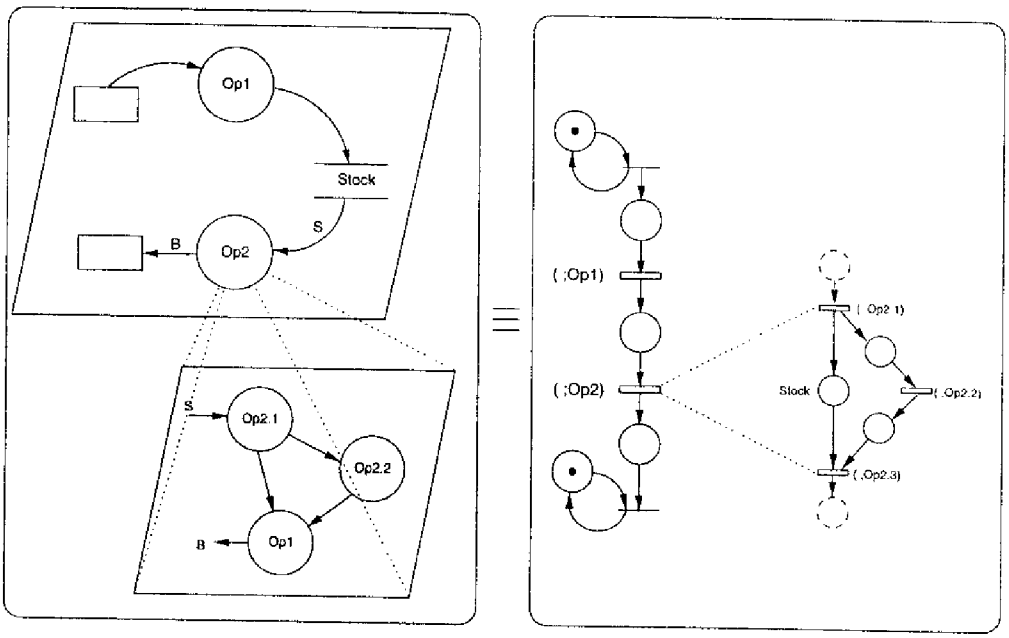


fig III.13 – Décomposition en niveaux hiérarchiques

niveau supérieur, la transition représentant la transformation de données à décomposer ainsi que ses places d'entrées et de sorties, par le réseaux de Petri représentant le sous-niveaux. Ce type de hiérarchisation des réseaux de Petri a été traité par plusieurs études dont [Zuberek et Bluemke 96].

Dans l'exemple exposé par la figure *III.13*, ces deux façons de représenter les niveaux hiérarchiques donnent la même représentation. En effet, dans ce cas le sous-niveau développé est bien un bloc de réseau de Petri (une transition de début et une transition de fin de traitement). Toutefois, il n'est pas toujours aussi facile d'aboutir à un sous-niveau qui se prête à une représentation par bloc, particulièrement quand il y a plusieurs flots d'entrée/sortie. Or, il est très utile de ramener la représentation des sous-niveaux à des blocs de réseaux de Petri pour avoir une représentation homogène d'une part et pour pouvoir procéder à l'analyse des sous-niveaux indépendemment du système global, d'autre part.

Quand il y a plusieurs données à l'entrée d'une transformation de données de niveau supérieur, le réseau de Petri représentant le DFD obtenu en décomposant cette transformation peut être ramené à un bloc en lui ajoutant une transition de début et une transition de fin. La première transition fournit les données à toutes les places représentant les flots de données d'entrée du sous-niveau (répartition des données) et la seconde recueille les données de toutes les places représentant les flots de sortie (encapsulation de données). Cet arrangement permet d'avoir une représentation par des blocs de réseaux de Petri qui se prête à une validation par partie.

Il n'est pas nécessaire de définir une méthode pour décrire la répartition et l'encapsulation des données, ce sont les règles de production et de consommation de données

présentées plus haut (figure *III.12*) qui décrivent les combinaisons (ET,⊕,...) utilisées. La combinaison de données à utiliser pour la répartition des données (ET, OU,...) doit être la même que celle utilisée à l'entrée de la transformation de données à décomposer. De même, la combinaison de données à utiliser pour encapsuler les données doit être la même que celle utilisée à la sortie de la transformation à décomposer.

La figure *III.14* montre une transformation de données et sa décomposition au niveau hiérarchique suivant. La figure *III.15* donne la représentation par réseaux de Petri de cette décomposition. La possibilité de ramener cette décomposition à une représentation par un bloc de réseau de Petri est montrée dans la figure *III.16*. Cette figure illustre l'agrégation du sous-niveau et les étapes de répartition et d'encapsulation des données.

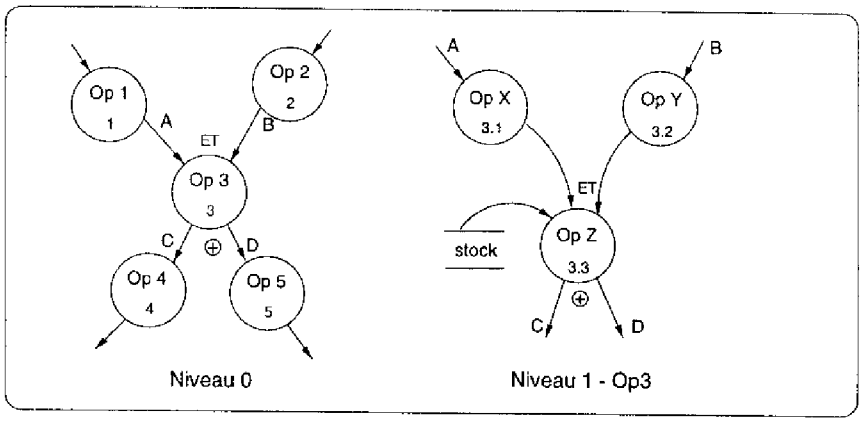


fig III.14 – Représentation hiérarchique de DFDs

Remarquons que cette représentation par un bloc de réseau de Petri n'est pas applicable (ou difficilement applicable) au niveau du diagramme de contexte. En effet, cette opération sous-entend que l'analyste connaît exactement la nature des opérations, les données consommées et produites ainsi que leurs associations dès l'établissement du diagramme de contexte, ce qui n'est pas réaliste étant donné le haut degré d'abstraction de ce diagramme. Ainsi, l'agrégation des différents niveaux de réseaux de Petri ne peut être entreprise qu'à partir du niveau zéro sur lequel il faut reporter les puits et sources de données.

Dans la suite de ce mémoire, nous serons souvent amenés à étudier un niveau hiérarchique intermédiaire des spécifications SA-RT. Pour des raisons de clarté des figures, nous ne représenterons pas systématiquement les blocs de réseaux de Petri. Le réseau de Petri représentant le DFD du niveau étudié aura des places sources et des places puits qui représenteront **les données de bord de diagramme**. Ces places sont les places qui permettront de rattacher le réseau de Petri représentant ce niveau à celui représentant le niveau immédiatement supérieur. Ce sont des places de communications entre niveaux. Elles seront schématisées par des cercles hachurés (comme le montre la figure *III.15*).

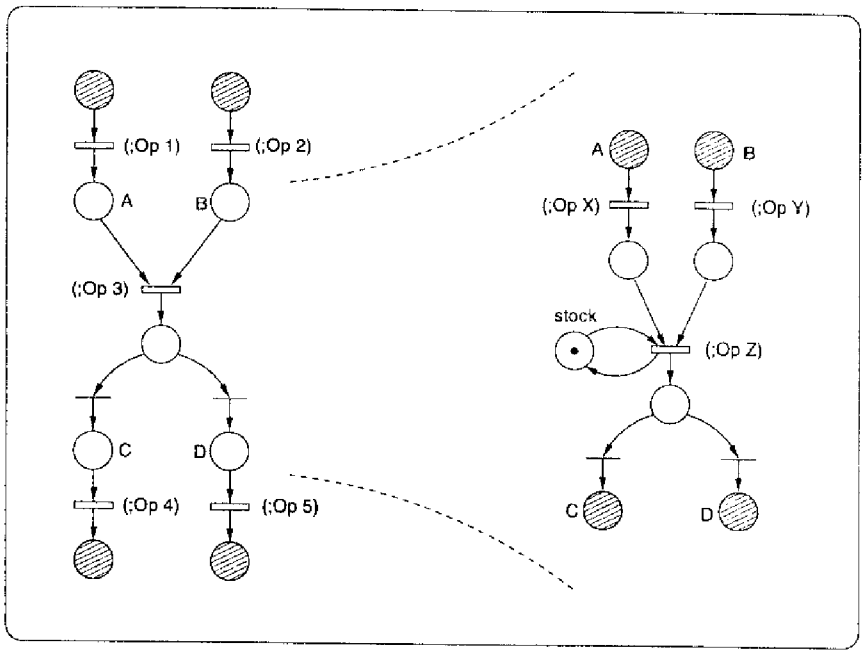


fig III.15 – Représentation de la décomposition par réseaux de Petri

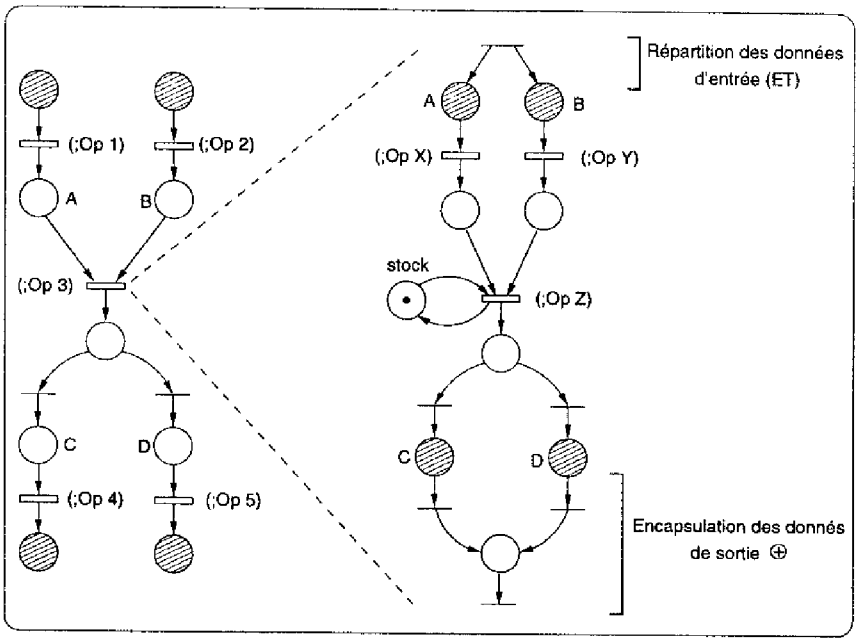


fig III.16 – Représentation par un bloc de réseau de Petri

III.8 Représentation de l'aspect dynamique

La description de l'aspect dynamique constitue un problème crucial pour les systèmes temps réel. Ce problème est au centre des préoccupations des concepteurs de

logiciels temps réel. D'ailleurs, les contributions de Ward et Mellor et de Hatley et Pirbhai pour la définition d'une méthodologie du type SA-RT ont consacré une place assez conséquente à cet aspect.

Dans SA-RT, l'aspect dynamique est décrit par les diagrammes de flots de contrôle (DFC). Ils sont composés essentiellement de flots de contrôle et de transformations de contrôle dont les activités sont spécifiées par des structures de contrôle ou Cspecs (control specifications). Les transformations de contrôle représentent les centres de décision et les flots de contrôle matérialisent les interactions entre ces centres et les autres éléments du diagramme. Les flots de contrôle peuvent être classés en trois groupes :

- Les flots de contrôle provenant de (ou allant vers) les sources (ou puits) d'informations. C'est l'interaction du DFC avec l'environnement du système.
- Les flots de contrôle échangés entre les transformations de contrôle et les transformations de données et composés, eux-mêmes, d'événements (dans le sens données vers contrôle) ou d'ordres d'activation (dans le sens opposé). C'est l'interaction entre DFD et DFC.
- Occasionnellement, les flots de contrôle entre les transformations de données. Ce sont des ordres d'activation et/ou des événements traités localement entre deux transformations de données sans passer par une transformation de contrôle.

Les évolutions dynamiques du système sont gérées par la structure de contrôle ou Cspec. Les différentes versions de SA-RT utilisent des modèles variés pour représenter les Cspecs (Machines à États Finis, tables d'activation des processus,...). Dans [Benzina et Paludetto 96b] nous avons montré l'adéquation des réseaux de Petri pour la modélisation de cet aspect dynamique (voir exemple de la figure *III.17*).

Pour la représentation des DFCs par réseaux de Petri, les Cspecs sont reprises sous la forme de réseaux de Petri et fournissent l'aspect dynamique. Quant aux flots de contrôle proprement dit, ils relèvent plutôt de l'aspect statique. Ils peuvent être représentés par des places, au même titre que les flots de données. En effet, au sens de la circulation des données, les deux types de flots sont porteurs d'informations. Cette représentation est intéressante pour les flots de contrôle échangés entre transformations de données. Cependant, pour les autres types de flots de contrôle elle ne servira qu'à créer des places sources ou puits dont la signification est simplement l'arrivée ou le départ d'une information de contrôle.

Les places représentant les flots de contrôle pourraient être utilisées pour modéliser les interactions entre le réseau de Petri relatif à l'aspect dynamique et celui relatif à l'aspect statique (voir exemple figure *III.18*). Cependant, il ne nous semble pas judicieux de relier structurellement ces deux réseaux. En effet, bien que les deux aspects soient représentés par le même formalisme, les interprétations des deux réseaux de Petri ne sont pas les mêmes. Dans le réseau de Petri modélisant l'aspect dynamique, le marquage des places donne l'état du système alors que dans le réseau de Petri représentant l'aspect statique les jetons représentent les données elles-mêmes. Relier les deux réseaux serait source de confusion pour la représentation que nous avons adoptée. Si

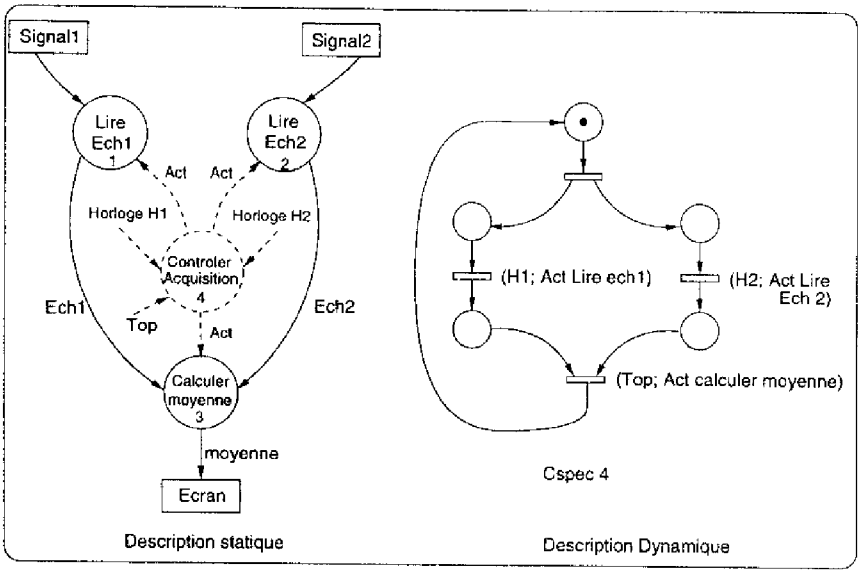


fig III.17 – Représentation par la méthode SA-RT

toutefois nous désirons mettre en relation les deux types de réseaux de Petri, il faudrait adopter une représentation identique à celle proposée par [Elmstrom *et al.* 93b] ou [Shi et Nixon 96] (voir figure *III.2* page 61). Dans ce cas, les deux réseaux de Petri modélisent des états et l'interaction entre les deux est significative. Elle est bien illustrée dans [Shi et Nixon 96].

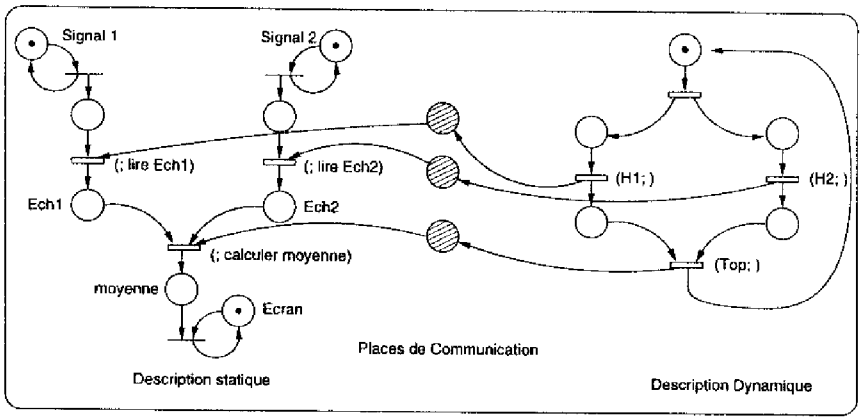


fig III.18 – Interaction structurelle des parties dynamique et statique

Proposition

Il semble plus sain de garder la séparation structurelle entre les réseaux de Petri relatifs aux parties statiques et dynamiques. Les interactions entre les deux seront visibles au niveau des couples (*condition*; *action*) associés aux transitions (voir exemple de la figure *III.19*). Ceci permet d'étudier séparément les deux aspects. ■

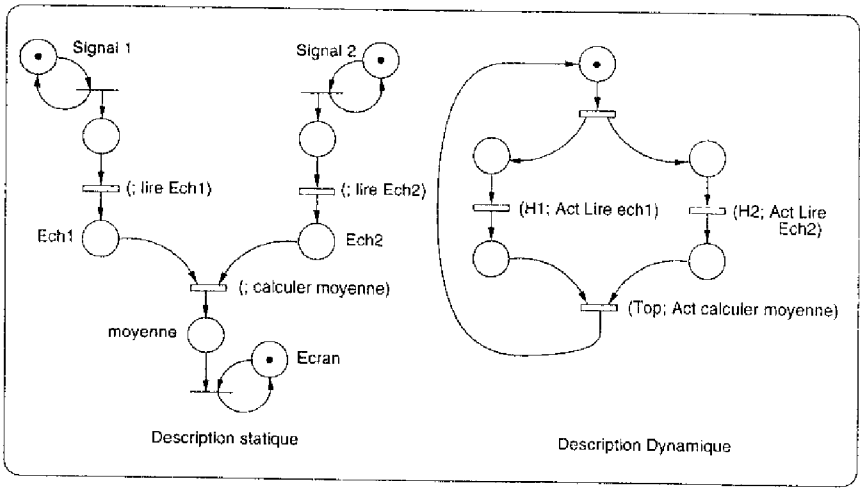


fig III.19 – Représentation par réseaux de Petri des aspects statique et dynamique

III.9 Conventions de représentation

Étant donné la multiplicité des modèles utilisés, certaines conventions de représentation ont été adoptées pour faciliter la compréhension des diagrammes et figures présentés.

Dans notre travail les réseaux de Petri ont une double utilité. D'une part ils représentent le contrôle (Cspec) associé à une transformation de contrôle dans un niveau de spécification SA-RT. Dans ce cas, le réseau de Petri et le DFD qu'il contrôle sont dessinés côte à côte dans le même rectangle (ex figure *III.17*). D'autre part, les réseaux de Petri sont également utilisés pour représenter un DFD. Dans ce cas, le réseau de Petri et le DFD qu'il représente sont dessinés dans des rectangles séparés, joints par le signe de relation $\equiv$ (ex figure *III.7*).

Par ailleurs, au niveau des réseaux de Petri représentant les DFDs, nous avons distingués les transitions relatives aux transformations de données des autres transitions. En effet, elles sont représentées par des rectangles et sont susceptibles d'être désagrégées à un niveau inférieur (voir figure *III.7*).

III.10 Conclusion

La représentation des diagrammes SA-RT, proposée dans ce chapitre, intègre les aspects statiques et dynamiques des spécifications. Elle a été effectuée dans le souci de garder la lisibilité des diagrammes SA-RT. Parmi les objectifs, présentés au début du chapitre, nous avons pu assurer la modularité de la représentation. Cependant, l'automatisation de la démarche ne peut pas être totale. En effet, nous avons relevé plusieurs points où l'utilisateur de la méthode doit intervenir pour résoudre certaines ambiguïtés dues au manque de formalisme des diagrammes SA-RT. Ainsi, il est possible

d'aboutir à une automatisation assistée, où le système proposerait des représentations possibles à l'utilisateur qui retiendrait les plus adéquates.

La représentation de l'aspect statique a été axée autour de la circulation des données dans les diagrammes. Cet aspect est celui qui nous intéresse le plus, puisque cette représentation a été entreprise dans le but de répartir les besoins fonctionnels en besoins orientés objet. Nous aborderons ce sujet dans le chapitre suivant.

La représentation obtenue pourrait également être utilisée pour d'autres objectifs. En particulier, nous examinerons, au chapitre V la possibilité de l'utiliser à des fins de validation et d'évaluation des spécifications.

Chapitre IV

Utilisation des réseaux de Petri en vue de la décomposition en objets

La représentation des diagrammes SA-RT par des réseaux de Petri a été entreprise pour servir de support à la recherche des objets dans une spécification fonctionnelle. Nous exposons, dans ce chapitre, la technique proposée, ses avantages et ses limites.

IV.1 Idée directrice

Dans la représentation des DFDs par réseaux de Petri nous avons axé notre description sur la circulation des données. Cette circulation des données entre les différentes transformations sera l'un des supports de la décomposition en objets. En effet, la circulation d'une donnée dans un ensemble de fonctions peut être considérée comme un mécanisme d'encapsulation aboutissant à la constitution d'un objet comprenant les données considérées (une ou plusieurs), les fonctions agissant sur ces données et le comportement qui leur est associé.

La difficulté réside dans la recherche d'une bonne technique de regroupement (données, fonction, comportement) qui produit des candidats adéquats à une décomposition des spécifications fonctionnelles en objets.

Notre orientation vers les réseaux de Petri n'est pas fortuite. En effet, dans la théorie d'analyse des réseaux de Petri, il existe deux mécanismes de partitionnement qui ont déjà été utilisés dans une méthodologie orientée objet. Il s'agit des techniques de recherche des composantes conservatives et répétitives stationnaires utilisées pour la

composition et la décomposition d'objets dans la méthodologie HOOD/PNO. Leur justification et leur adéquation aux approches à objet a été montrée dans [Paludetto 91] et [Paludetto et Raymond 93].

Il est à signaler ici que l'utilisation des composantes conservatives et répétitives stationnaires ne peut pas donner de résultats avec la représentation des diagrammes SA-RT adoptée dans [Elmstrom *et al.* 93b] et [Shi et Nixon 96]. En effet, ce type de représentation s'appuie sur la modélisation de l'état des éléments des diagrammes SA-RT plutôt que sur la modélisation de la circulation des données. Pour cela, il a été nécessaire d'adopter une technique de représentation basée sur la circulation des données (cf. III)

IV.2 Rappel sur les composantes conservatives et répétitives stationnaires

La recherche des composantes conservatives et répétitives stationnaires est une technique d'analyse structurelle qui permet de caractériser des propriétés particulières d'un réseau de Petri. Il s'agit d'une analyse formelle permettant de prouver, dans des cas particuliers, certaines bonnes propriétés des réseaux de Petri. Cette technique d'analyse permet de s'affranchir des problèmes d'explosion combinatoire rencontrés avec les techniques d'analyse par énumération des marquages accessibles.

Les composantes conservatives et répétitives stationnaires sont issues de la recherche de propriétés liées à la structure du réseau de Petri en faisant abstraction de son marquage.

Soit l'équation fondamentale d'un réseau de Petri dont la matrice d'incidence est C :

$$M' = M + C.\bar{s}$$

En prémultipliant chacun des termes de cette équation par un vecteur f^T représentant une fonction linéaire du marquage des places du réseau de Petri, l'équation obtenue est :

$$f^T.M' = f^T.M + f^T.C.\bar{s}$$

La seule façon d'obtenir des propriétés indépendantes du marquage initial est d'annuler le terme $f^T.C.\bar{s}$ soit en annulant $f^T.C$ ou alors en annulant $C.\bar{s}$. Ce qui donne naissance aux composantes conservatives et répétitives définies dans les paragraphes suivants.

Un exposé plus complet sur les composantes conservatives et répétitives peut être consulté dans [Valette 95], [David et Alla 89], [Silva *et al.* 85], [Krückeberg et Jaxy 87].

IV.2.1 Composantes conservatives et invariants de places

Soit $\mathcal{R}$ un réseau de Petri et C sa matrice d'incidence. Une **composante conservative** est une solution de l'équation :

$$f^T.C = 0$$

Un vecteur f solution de cette équation définit une fonction linéaire du marquage des places du réseau de Petri $\mathcal{R}$ dont la valeur reste constante pour toute séquence tirable :

$$f^T.M = f^T.M' \qquad \forall s / M \xrightarrow{s} M'$$

En particulier, en considérant le marquage initial, la relation

$$f^T.M = f^T.M_0 \qquad \forall M \in A(\mathcal{R}, M_0),$$

définit un **invariant de places** correspondant à la composante conservative f, $A(\mathcal{R}, M_0)$ étant l'ensemble des marquages accessibles à partir du marquage initial. Notons qu'un invariant de places dépend du marquage initial alors qu'une composante conservative n'en dépend pas.

Une composante conservative f est dite **positive** si tous ses éléments sont positifs. Une composante conservative f est dite **minimale** si elle ne peut pas être obtenue en faisant la somme de deux autres composantes conservatives positives.

Graphiquement, une composante conservative délimite un sous-réseau de Petri formé par les places, correspondant aux éléments non nuls du vecteur f, et leurs transitions d'entrées et de sorties. Physiquement, ce sous-réseau de Petri correspond généralement à la conservation d'une ressource (*ex.* machines d'un atelier) ou à la caractérisation d'une propriété du système modélisé (*ex.* gamme d'usinage, file d'attente...).

IV.2.2 Composantes répétitives stationnaires et invariants de transitions

Soit $\mathcal{R}$ un réseau de Petri et C sa matrice d'incidence. Une **composante répétitive stationnaire** est une solution de l'équation :

$$C.\bar{s} = 0$$

Un vecteur $\bar{s}$ solution de cette équation caractérise une séquence s de franchissement de transitions dans le réseau $\mathcal{R}$, qui ne modifie pas son marquage. Si cette séquence est effectivement franchissable à partir d'un marquage accessible alors elle constitue un **invariant de transitions**. Ainsi, comme pour les composantes conservatives, les composantes répétitives stationnaires ne dépendent pas du marquage initial alors que l'existence de l'invariant de transitions en dépend.

Graphiquement, un invariant de transitions délimite un sous-réseau de Petri formé par les transitions de la séquence s, caractérisé par le vecteur $\bar{s}$, et leurs places d'entrées

et de sorties. Physiquement, ce sous-réseau est généralement relatif à un cycle répétitif d'opérations (*ex.* les opérations nécessaires à la préparation d'une pièce dans un atelier).

IV.2.3 Les algorithmes de recherche des composantes conservatives et répétitives

Le problème de la recherche des composantes conservatives et répétitives se ramène à la résolution d'une équation du type :

$$A.x = 0$$

avec $A = C$ (matrice d'incidence) pour les composantes répétitives et $A = C^T$ pour les composantes conservatives.

La résolution de ce problème a fait l'objet de plusieurs travaux. Les algorithmes proposés sont plus ou moins efficaces selon le type de la matrice A. Un algorithme particulièrement accessible est basé sur la triangularisation de Gauss de la matrice A. Il permet de calculer une base de composantes conservatives.

Comme nous nous intéressons plutôt à l'exploitation des composantes répétitives et conservatives, nous n'allons pas nous étendre sur les techniques utilisées pour les calculer. De plus amples études peuvent être trouvées dans [Paludetto 91], [Valette 95], [Silva *et al.* 85], [Alaiwan et Toudic 85], [Krückeberg et Jaxy 87]...

IV.3 Identification des objets

Le problème peut se poser sous deux formes différentes. Il est possible que les objets de la spécification soient déjà identifiés grâce à une étude antérieure (par exemple à partir des entités physiques du système spécifié). Dans ce cas, le problème se limite à répartir les composants des diagrammes de spécifications entre les différents objets déjà connus. D'un autre coté, il est possible que les objets ne soient pas encore connus. Alors, il s'agit, non seulement de répartir les éléments du DFD, mais aussi d'identifier les objets pertinents pour la spécification du système étudié. Nous montrons dans la suite que l'utilisation des réseaux de Petri peut servir dans un cas ou dans l'autre, en parallèle ou en vérification mutuelle.

La représentation des diagrammes SA-RT par réseaux de Petri, entreprise dans le chapitre précédent, aboutit à deux (groupes de) réseaux de Petri correspondant à la partie statique (DFD) et à la partie dynamique (structure de contrôle).

Ainsi, nous avons deux points de départ pour la recherche des composantes conservatives et répétitives : le réseau de Petri décrivant les DFDs et celui décrivant le contrôle. Dans ce qui suit, nous montrons les décompositions possibles à partir de chaque type de réseaux, puis, l'adéquation de chaque démarche à la nature du problème étudié.

IV.3.1 Identification des objets à partir des DFDs

Pour la recherche des objets, considérons d'abord les réseaux de Petri décrivant les DFDs. Rappelons que dans ces réseaux les jetons représentent les données qui circulent dans le système spécifié.

IV.3.1.1 Composantes conservatives ou répétitives?

Une composante répétitive représente un cycle répétitif d'opérations dans un système. Or, dans les réseaux de Petri représentant les DFDs, il est très rare de voir apparaître des cycles. En effet, les diagrammes de flots de données sont des diagrammes transformationnels, ils prennent en entrée des données, les traitent et fournissent le résultat en sortie. L'aspect cyclique est sous entendu, il n'apparaît pas sur les DFDs mais apparaît éventuellement sur la structure de contrôle. Donc, la recherche des composantes répétitives n'est pas pertinente dans le cas des réseaux de Petri représentant des DFDs. Nous nous intéresserons plutôt à la recherche des composantes conservatives.

IV.3.1.2 Les premiers objets

Selon les règles de translation DFDs-réseaux de Petri, les composantes conservatives les plus évidentes sont celles relatives aux stocks, aux puits et sources de données. Ces éléments sont de bons candidats à une représentation par objets des spécifications. Ceci nous conforte dans notre idée d'autant plus que ces éléments représentent généralement des entités physiques du système étudié.

Cependant, ces composantes conservatives ne couvrent pas tout le réseau de Petri. En particulier, dans un RdP représentant un DFD de niveau intermédiaire, il est très commun de ne trouver aucun de ces éléments. Ainsi, il faut chercher des composantes conservatives qui prennent en compte les données à travers tout le DFD. Pour cela, il est nécessaire d'effectuer de légères modifications de mise en forme des réseaux de Petri décrivant les DFDs avant de procéder à la recherche des composantes conservatives.

IV.3.1.3 Mise en forme du réseau de Petri

Lors de la représentation des diagrammes DFDs par des réseaux de Petri, les données de bord d'un DFD étaient représentées par des transitions sources et puits (relatifs aux sources et puits d'information des DFDs, ex: transitions t_1 et t_2 de la figure *IV.1*). La présence de ces transitions rend impossible la recherche des composantes conservatives. Or, pour notre objectif de recherche des objets, nous nous intéressons plutôt à la structure interne du réseau de Petri obtenu et les données de bord ne sont pas d'une importance capitale. Ainsi, il est préférable de les représenter par des places, soit en enlevant les transitions sources et puits pour ne garder que la structure interne du réseau, soit en rajoutant une place en amont de chaque transition source et en aval de chaque transition puits. C'est cette seconde solution qui sera adoptée (ex: places P_1 et P_6 de la figure *IV.2*). Pour notre démarche plutôt qualitative, il n'est pas nécessaire

de fixer un nombre de jetons précis dans les places sources. En effet, les composantes conservatives sont indépendantes du marquage initial (cf. §IV.2). Par ailleurs, notons que ces modifications d'ordre pratique ne modifient pas le comportement du réseau.

De façon imagée, l'addition de ces places sources et puits peut être vue comme le fait d'extraire le DFD étudié de son environnement naturel (où il est confronté à des flux de données non limités) et le mettre sur un «banc d'essai» où les entrées/sorties sont contrôlées, afin d'étudier le comportement de ce DFD.

Mis sous cette forme, le réseau de Petri représentant le DFD comporte des composantes conservatives qui intègrent les données depuis leurs entrées jusqu'à leurs sorties. Ces composantes conservatives se recouvrent entre elles et couvrent, généralement, tout le réseau de Petri. La somme de toutes ces composantes est une composante conservative particulière, elle couvre, généralement, tout le réseau de Petri et traduit l'équation de conservation des données dans tout le DFD (voir plus loin les exceptions pour lesquelles il n'y a pas de couverture totale du réseau de Petri représentant un DFD §IV.3.1.6).

Dans ce réseau, une composante conservative donne le chemin suivi par une (ou plusieurs) donnée d'entrée, sous une forme ou une autre, et les transformations de données impliquées dans son traitement. Il reste donc à déterminer comment ces composantes participeront à la répartition des spécifications fonctionnelles en objets.

IV.3.1.4 Exemple préliminaire

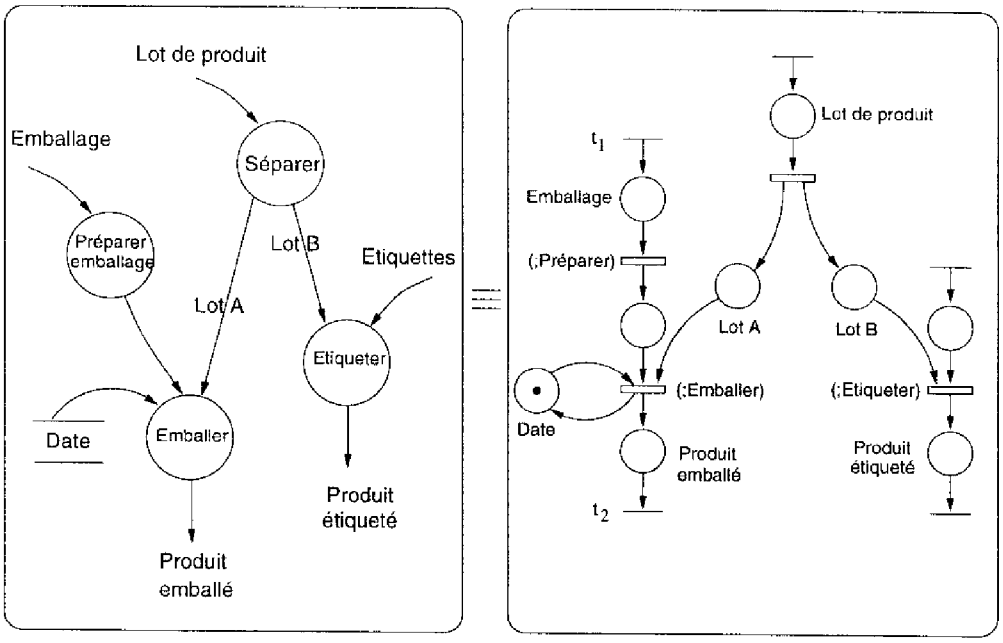


fig IV.1 - Exemple préliminaire

La partie gauche de la figure *IV.1* montre un niveau des spécifications d'un système

d'emballage et d'étiquetage. A droite, le même DFD est représenté par un réseau de Petri. La mise en forme de ce réseau de Petri donne en résultat le réseau de Petri de la figure *IV.2.* (des places sources et puits ont été rajoutées pour représenter les données de bord de diagramme). L'étude de ce réseau de Petri nous révèle l'existence de composantes conservatives minimales caractérisées par les équations suivantes :

$$\begin{aligned}
C_1 &\rightarrow M(p_4) = 1 \\
C_2 &\rightarrow M(p_1) + M(p_2) + M(p_3) + M(p_5) + M(p_6) = C^{te} \\
C_3 &\rightarrow M(p_{11}) + M(p_{12}) + M(p_{13}) + M(p_{14}) = C^{te} \\
C_4 &\rightarrow M(p_5) + M(p_6) + M(p_7) + M(p_8) + M(p_9) = C^{te} \\
C_5 &\rightarrow M(p_7) + M(p_8) + M(p_{10}) + M(p_{13}) + M(p_{14}) = C^{te}
\end{aligned}$$

et une composante conservative particulière, non minimale, qui couvre tout le réseau :

$$\begin{aligned}
M(p_1) + M(p_2) + M(p_3) + M(p_4) + 2M(p_5) + 2M(p_6) + 2M(p_7) + 2M(p_8) \; + \\
+M(p_9) + M(p_{10}) + M(p_{11}) + M(p_{12}) + 2M(p_{13}) + 2M(p_{14}) \; = \; C^{te}
\end{aligned}$$

Cette dernière équation traduit la conservation de toutes les données dans le système (*rien ne se perd, rien ne se crée, tout se transforme* [Lavoisier]).

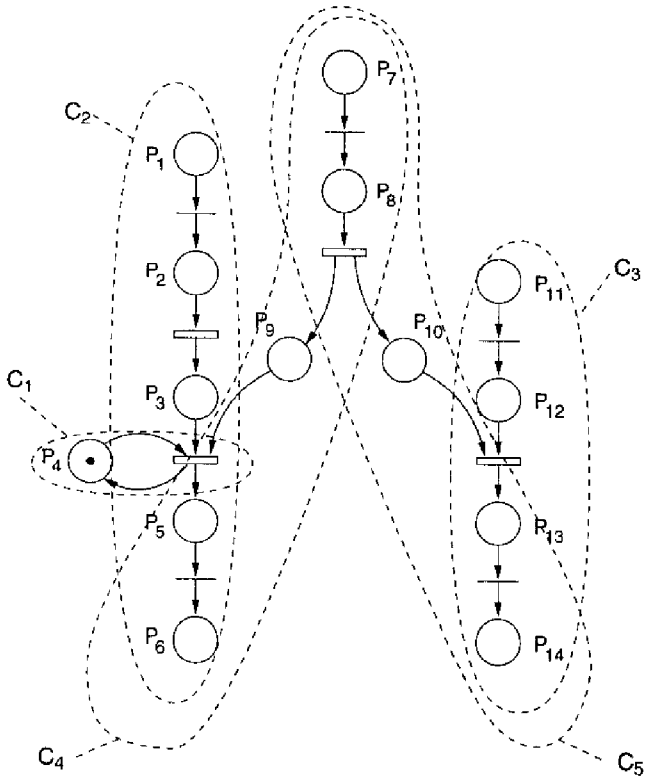


fig IV.2 - Exemple - Mise en forme du réseau de Petri et composantes conservatives

Les composantes conservatives trouvées ($C_1..C_5$) correspondent à des sous-réseaux de Petri (cf. §IV.2) candidats à une éventuelle décomposition en objets. Alors, une possibilité de décomposition peut consister en un ensemble d'objets : **horodateur** (C_1), **Poste d'emballage** (C_2), **Poste d'étiquetage** (C_3), **Lot A** (C_4) et **Lot B** (C_5). Ces objets ne sont que des candidats, il faut les confirmer en fonction du contexte du problème. Il faut également bien répartir les données et les opérations qui les concernent.

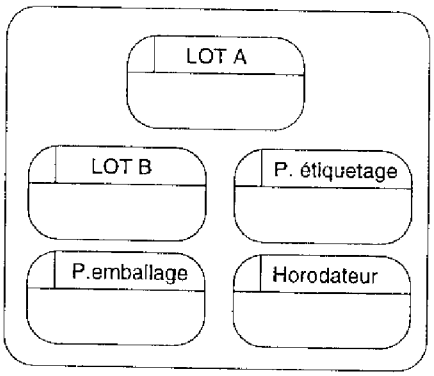


fig IV.3 – Exemple - Objets candidats

IV.3.1.5 Objets candidats à la décomposition

Les composantes conservatives qui couvrent un réseau de Petri représentant un DFD ne sont pas toutes disjointes (cf. figure *IV.2*). Il est donc nécessaire de faire un choix d'affectation lors de la décomposition de façon à éviter les redondances. Rappelons que dans la représentation adoptée, les jetons (donc les places) correspondent aux données et les transitions aux opérations. Une opération commune est affectée à l'un des objets et appelée par l'autre (définition des communications).

Pour illustrer les différentes possibilités de répartition, prenons par exemple le cas d'un réseau de Petri $\mathcal{R}$ couvert par deux composantes conservatives C_i et C_j qui ont quelques places en commun (c'est le cas des composantes C_2 et C_4 dans l'exemple précédent). Ces deux composantes définissent deux sous-réseaux de Pétri, $\mathcal{R}_i$ et $\mathcal{R}_j$, qui se recouvrent partiellement :

$$C_i \longrightarrow \mathcal{R}_i = \mathcal{R}_{i_0} + \mathcal{R}_0$$

$$C_j \longrightarrow \mathcal{R}_j = \mathcal{R}_{j_0} + \mathcal{R}_0$$

$\mathcal{R}_0$ étant le sous-réseaux formé par les places, les transitions et les arcs communs aux deux sous-réseaux $\mathcal{R}_i$ et $\mathcal{R}_j$.

Les différentes possibilités de décomposition en objets des spécifications représentées par le réseau de Petri $\mathcal{R}$ sont :

- Un seul objet défini par la réunion des deux réseaux $\mathcal{R}_i$ et $\mathcal{R}_j$. Cet objet peut éventuellement avoir des objets fils parmi les objets décrits plus bas.

- Deux objets, l'un associé à $\mathcal{R}_i$ et l'autre à $\mathcal{R}_j$. Les transitions de $\mathcal{R}_0$ communes aux deux composantes conservatives donneront lieu à des communications entre ces deux objets. Les opérations associées à ces transitions seront offertes par l'un des objets et appelées par l'autre objet. L'affectation est du ressort de l'analyste-concepteur. Les données appartiendront exclusivement à l'un ou l'autre des objets. C'est le cas le plus fréquent pour la formation des objets.
- Trois objets définis par les sous-réseaux $\mathcal{R}_0$, $\mathcal{R}_{i_0}$ et $\mathcal{R}_{j_0}$. C'est un cas plus rare.

Le choix de l'une ou l'autre de ces décompositions dépend du contexte du problème. En effet, la décomposition ne peut pas être systématique et indépendante du contexte du problème. Un contre exemple évident est le cas de deux DFDs ayant la même structure (mêmes diagrammes) mais qui représentent la spécification de deux systèmes différents. Il est normal et même logique d'aboutir à deux décompositions en objets différentes suivant le type du système.

Dans le cas de l'exemple précédent, le réseau de Petri représentant le DFD peut être couvert par cinq composantes conservatives minimales $(C_1, C_2, C_3, C_4, C_5)$. La décomposition en objets choisie est une répartition en 3 objets : un premier objet relatif aux composantes C_4 et C_5 : c'est un objet relatif aux **Lots** (il pourra éventuellement être décomposé en deux objets fils **Lot A** et **Lot B**). Un second objet relatif à la composante C_3 : c'est le **Poste d'étiquetage**. Un dernier objet relatif aux composantes C_1 et C_2 : c'est le **Poste d'emballage**. La figure *IV.4* détaille la répartition du réseau de Petri sur ces objets et montre les interactions entre eux. Les objets **P emballage** et **Lots** se partagent une transition, c'est une communication dont le protocole est à préciser. Ils se partagent aussi les places P_5 et P_6, c'est une donnée relative au *produit emballé* (voir DFD), donc elle est allouée à l'objet **Lots**.

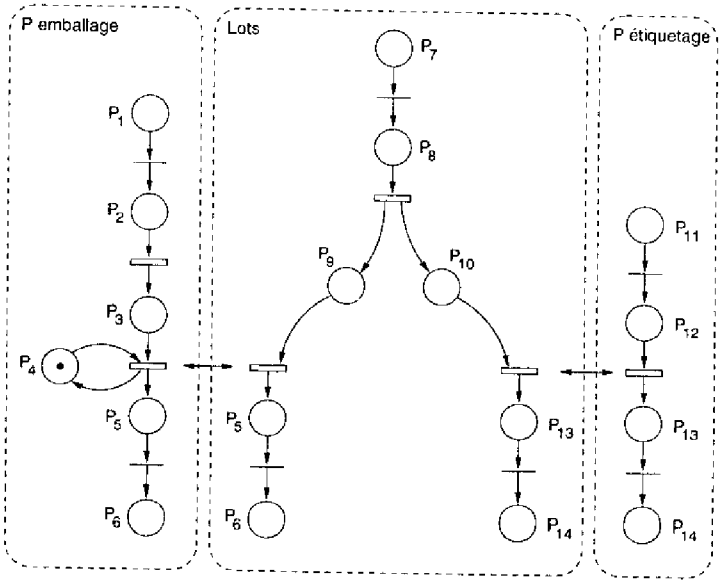


fig IV.4 – Exemple - Décomposition en objet

La proposition suivante permet de généraliser le mécanisme de décomposition :

Proposition

Lors de la décomposition d'un réseau de Petri représentant un DFD, peut être candidat à une représentation par objet,

- Un sous-réseau de Petri relatif à une ou plusieurs composantes conservatives.
- Un sous-réseau de Petri relatif à une composante conservative dont une partie a été réduite (car elle est commune avec une autre composante conservative).
- Un sous-réseau de Petri relatif à une partie commune à deux (ou plusieurs) composantes conservatives (voir $\mathcal{R}_0$ plus haut). C'est un cas à éviter car il peut centraliser le contrôle au sein d'un seul objet.

■

Les places et les transitions de jonctions entre les différents sous-réseaux de Petri, traduisent les activités de communications et d'échanges de données entre objets. Cet aspect sera développé plus loin.

IV.3.1.6 Problème de la couverture globale du réseau de Petri

Dans l'exemple précédent, tout le réseau de Petri représentant le DFD est couvert par un ensemble de composantes conservatives, ce qui nous a permis de répartir toutes les transformations de données du DFD en rapport avec ces composantes conservatives.

Cependant, telles qu'elles ont été énoncées, les règles de représentation des DFDs par des réseaux de Petri ne garantissent pas l'aboutissement à un réseau de Petri pouvant toujours être couvert par un ensemble de composantes conservatives. Il existe deux cas fréquents qui limitent la recherche des composantes conservatives. Dans le premier, la donnée est transformée d'une manière invisible par le DFD (cas d'une opération de test de valeur par exemple, figure *IV.5a*). Le second cas concerne les opérations de mise à jour des stocks de données (voir figure *IV.5d*).

Dans le cas d'une opération de test de valeur, toute la chaîne de transformations de données qui précède l'opération de test ne peut pas être couverte par une composante conservative (voir P_1, t_1, P_2, t_2 et P_3 figure *IV.5b*). En effet, cette opération de test (transition t_3) agit comme une transition puits vis-à-vis de la donnée ($Data$) qui est transformée d'une manière invisible par le DFD (événement $Data > 0$) et donc par le réseau de Petri correspondant. Or, le flot de contrôle partant de la transformation de données correspondante peut être considéré comme porteur d'une information au même titre qu'un flot de données. Il est donc possible de considérer cette information comme une information de bord de diagramme et la représenter par une place puits comme il a été fait dans le paragraphe §IV.3.1.3. Ainsi, au sens de la circulation des données (et des informations au sens large) les flots de contrôle (ex: $Data > 0$) partants des transformations de données vers une transformation de contrôle pourront être représentés par des places puits (ex: place P_4 de la figure *IV.5c*). Ceci permettra de couvrir les opérations de test de valeur et les transformations précédentes par des composantes conservatives.

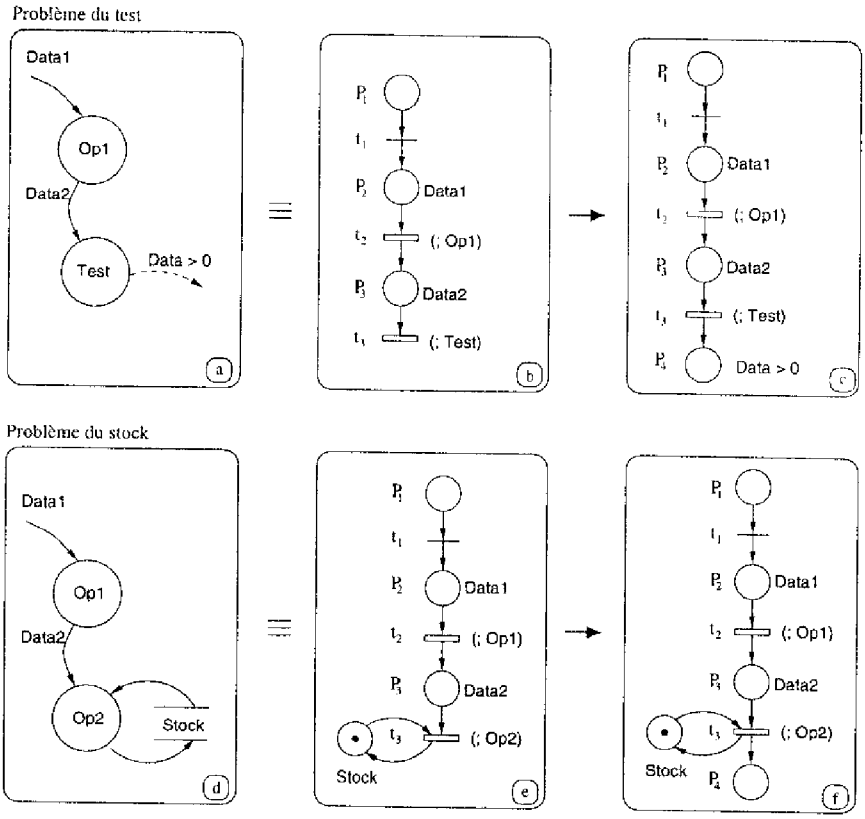


fig IV.5 - Cas particuliers - couverture par des composantes conservatives

De la même manière, dans le cas de la mise à jour d'un stock de données utilisé comme une mémoire, la chaîne de transformations de données précédant l'opération de mise à jour (P_1, t_1, P_2, t_2 et P_3 de la figure *IV.5e*) ne peut pas être couverte par une composante conservative. En effet, la transition représentant l'opération de mise à jour (t_3) agit comme une transition puits et le stock de données agit comme un puits de données (entités de bord de diagramme). Au sens de la circulation des données, il est également possible, comme cela a été fait pour les puits de données, de représenter la donnée avec une place puits (ex: place P_4 de la figure *IV.5f*), ce qui permettra de couvrir l'opération de mise à jour et les opérations précédentes par une composante conservative. (Le même raisonnement tient pour la représentation de places sources lors des opérations de lecture de mémoire.)

Ayant trouvé une représentation convenable pour ces cas qui émanent d'une représentation admissible vis-à-vis de SA-RT mais moins claire au sens de la circulation des informations, il est désormais possible de couvrir des réseaux de Petri représentant les DFDs par des composantes conservatives. Toutefois, s'il persiste des cas où le réseau de Petri ne peut pas être couvert par un ensemble de composantes conservatives, il faut d'abord s'assurer que le diagramme DFD est correctement établi, ensuite examiner individuellement la (les) partie non couverte pour l'allouer à un objet déjà défini ou en faire un objet spécifique.

IV.3.1.7 Reconstitution du comportement des objets à partir des DFDs

Tel qu'il a été défini dans le premier chapitre, un objet comporte aussi une structure de contrôle, appelée comportement (ou PNOBCS dans la méthodologie HOOD/PNO). Celle-ci gère les opérations et les interactions de l'objet avec son environnement. La recherche des composantes conservatives a permis de caractériser les objets par leurs attributs (données) et leurs opérations (transformations de données). Cependant, cette recherche s'est appuyé sur le réseau de Petri qui traduit le DFD et qui n'est pas un réseau de Petri de comportement. Ainsi, il reste à définir le comportement des objets obtenus.

Les DFDs dénotent une faible composante de contrôle malgré des caractéristiques statiques très dominantes. En effet, en l'absence de flot de contrôle, un flot de données entre deux transformations de données indique que les données subissent successivement ces deux transformations. De la même façon, un branchement dans un flot de données (*split*) peut suggérer l'existence d'activités indépendantes ou parallèles (du moins en ce qui concerne les spécifications, ce n'est pas le cas pour l'implémentation). Pour le contrôle des activités dans les DFDs voir [DeMarco 78], [Ward 86]. Le réseau de Petri représentant les DFDs explicite clairement cette structuration du contrôle entre les transformations de données. Par ailleurs, comme le montre le chapitre précédant, la résolution des ambiguïtés relevées dans les DFDs permet de préciser encore plus ce contrôle.

Ainsi, le sous-réseau de Petri isolé grâce à la recherche des composantes conservatives fournira un première support pour la composition du comportement de l'objet. A ce support du comportement, il faut rajouter les éventuels flots de contrôle entre transformations de données :

- Si ce flot de contrôle relie deux transformations de données relatives à un même objet, il sera matérialisé par une place entre les deux transitions relatives à ces transformations de données (séquencement explicite des deux opérations).
- S'il se situe entre deux transformations de données relatives à deux objets différents, il va induire une communication entre les deux objets (cf. paragraphe suivant).

Proposition

Le comportement d'un objet provient du sous-réseau de Petri représentant le DFD auquel il faut rajouter la prise en compte des éventuels flots de contrôle entre les transformations de données. Ce comportement doit être mis en forme pour correspondre au formalisme HOOD/PNO : un réseau de Petri généralement réinitialisable qui gère l'activation des opérations et la communication de l'objet avec son environnement. Par ailleurs, les mini-spécifications de certaines transformations de données primitives comportent une description textuelle des activités de contrôle de cette transformation. Il faut également prendre en compte cette description textuelle quand cela est nécessaire. ■

La composition du comportement associé à un objet est une opération très délicate car elle intègre sur un réseau de Petri des informations provenant de sources variées. C'est une activité de modélisation plus ou moins facilitée par le réseau de Petri représentant le DFD. Il est parfois nécessaire d'entreprendre une modélisation complète des activités de l'objet, mais cette modélisation reste une activité restreinte puisque la décomposition permet de restreindre le nombre d'opérations traitées. Il faut également boucler le réseau de Petri pour qu'il corresponde à une activité cyclique, si nécessaire, et le compléter par les activités de communications et d'échanges de données entre objets. Ces dernières sont exposées au paragraphe suivant.

Remarque : Il est à noter que pour un objet passif (cf. [Paludetto 91]) cette étape de définition du comportement de l'objet n'a pas lieu d'être puisque l'objet est réduit à un simple fournisseur d'opérations séquentielles.

IV.3.1.8 Communications entre objets déduites des DFDs

Les communications et les échanges de données entre objets peuvent être déduits de la décomposition du réseau de Petri représentant le DFD. En effet, les places reliant les sous-réseaux de Petri sélectionnés pour former des objets représentent, en même temps, des échanges de données et des requêtes d'opérations. Les transitions communes entre deux sous-réseaux représentent un appel d'opération de l'un de ces deux objets vers l'autre. Par ailleurs, un flot de contrôle seul entre deux transformations de données affectées à deux objets différents matérialise une requête d'opération sans échange de données.

Cependant, le type de la communication n'est pas systématiquement déduit du réseau de Petri. En fait, le formalisme HOOD/PNO classe les communications entre objets en trois types : ASER (ASynchronous Execution Request), LSER (Loosely Synchronous Execution Request), HSER (Highly Synchrounous Execution Request). A ce niveau, les spécifications SA-RT sont moins riches en détails. Le type de communication requis n'est pas toujours évident à voir. En général, l'échange de données avec requête d'opération peut être considéré comme une requête ASER. Par ailleurs, la lecture d'un stock de données qui appartient à un autre objet doit être traduite en requête HSER. Cependant il est difficile de définir systématiquement un type particulier de requête. C'est l'étape de conception qui permet de préciser d'avantage la nature des communications, nous nous appuyons ici simplement sur l'analyse des besoins.

La figure *IV.6* montre des exemples de spécification du type de communication. Le premier cas, à gauche, est une communication ASER vers l'objet *poste d'étiquetage*. Le second cas, à droite, montre le type de communication pouvant être spécifiée si un objet *Horodateur* est isolé (relativement à la composante conservative $C1$ de l'exemple précédent).

En réalité, la spécification du type de la communication est une activité propre aux représentations par objets. Elle est due à la notion d'appel d'opération «distante» (appartenant à un objet différent) induite par ces représentations par objets. Sauf cas particulier, ce mécanisme n'apparaît pas, en phase d'analyse, dans les spécifications

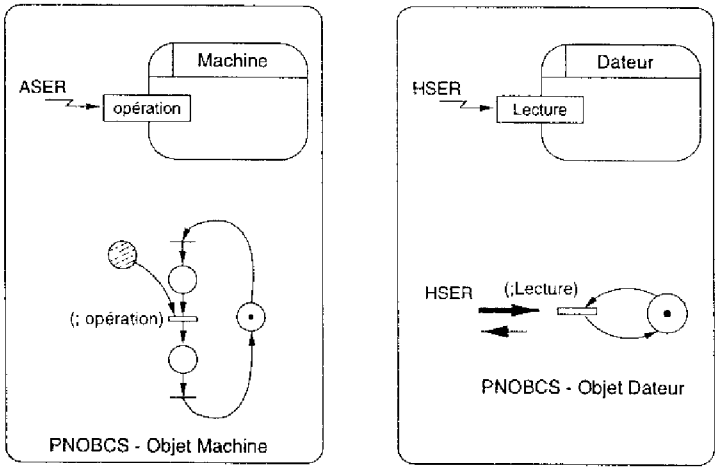


fig IV.6 – Exemples de communications entre objets

fonctionnelles, il revient alors à l'analyste-concepteur d'étudier chaque communication à part afin de lui associer un type particulier. Le sens est souvent guidé par les besoins de l'application.

Pour l'exemple étudié précédemment, l'association d'un comportement à chaque objet obtenu, et la spécification des communications entre ces objets conduisent à une représentation globale par objet selon le formalisme HOOD/PNO (voir figures *IV.7* et *IV.8*).

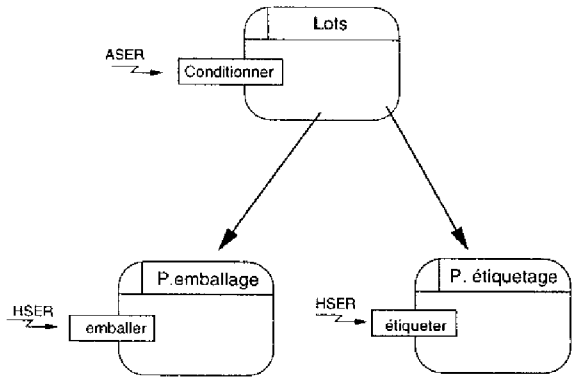


fig IV.7 – Exemple - Représentation objet

IV.3.2 Identification des objets à partir du réseau de Petri de contrôle

Dans la partie précédente, la décomposition en objets était basée sur l'exploitation du réseau de Petri représentant le DFD. Il s'agissait d'identifier les objets à partir des opérations et des données puis, de recomposer le comportement associé à chaque

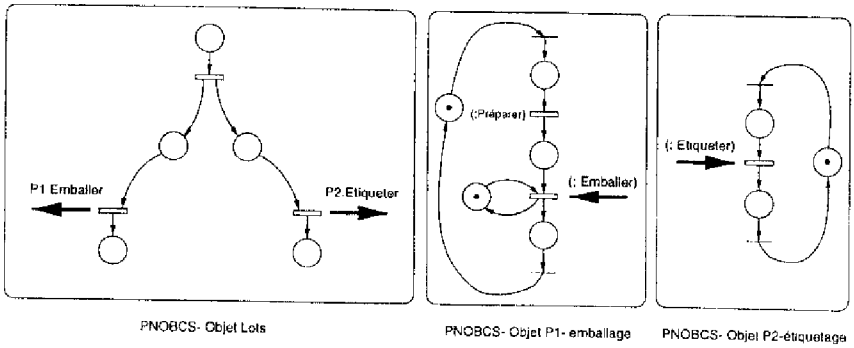


fig IV.8 – Exemple - PNOBCS des différents objets

objet. Dans les spécifications SA-RT des systèmes temps réel où l'aspect dynamique est important, il est plus intéressant d'identifier les objets d'abord par leur comportement. Dans ce cas, il faut baser la recherche des objets sur l'exploitation du réseau de Petri représentant la partie dynamique (le contrôle).

Cette démarche a déjà été exploitée dans la méthode HOOD/PNO pour la décomposition des objets pères en objets fils. En fait, dans HOOD/PNO, le PNOBCS de l'objet père est modélisé à l'aide d'un réseau de Petri à objets. La recherche des composantes conservatives dans ce réseau de Petri fournit de bons candidats à une décomposition en objets fils. Comme elle a déjà été étudiée, la méthode de décomposition ne sera pas très détaillée, un éventuel complément d'information peut être trouvé dans [Paludetto 91].

IV.3.2.1 Objets candidats à la décomposition

Pour l'exploitation du réseau de Petri représentant le contrôle d'une spécification SA-RT, il n'est pas nécessaire d'user d'artifices pour mettre en forme le réseau de Petri comme c'est le cas pour celui représentant le DFD. En effet, dans ce cas, le réseau de Petri est un modèle habituel de description d'un système à événements discrets. La recherche des bonnes propriétés (borné, vivant, réinitialisable...) de ce modèle permet de le valider.

Dans ce type de réseaux de Petri, une composante conservative est généralement relative à la conservation d'une ressource physique du système spécifié (machine, outil, aire de stockage...). Dans ce cas, comme les composantes conservatives ou P-invariants[1] sont relatifs à des entités physiques, ils donnent de bons candidats à la spécification en objets du système étudié.

1. Dans ce cas, il est possible de parler d'invariants de places ou P-invariants, puisque le marquage initial des réseaux de Petri est connu. Rappelons que dans le cas des réseaux de Petri représentant les DFDs le marquage initial n'a pas été fixé.

IV.3.2.2 Opérations et données

Dans la décomposition en objets, de la méthode HOOD/PNO, l'extraction des composantes conservatives suffit pour définir un objet : son comportement, ses opérations et ses attributs. En effet, l'utilisation des réseaux de Petri à objets permet de tout mettre sur le même modèle : les données, les opérations et leur contrôle.

Dans le cas des spécifications SA-RT, les aspects dynamiques et statiques sont séparés. Ainsi, si la recherche des objets se base sur l'étude du réseau de Petri correspondant à l'aspect dynamique, les objets obtenus ne seront définis que par leurs comportements. Alors, il faut compléter la spécification des objets en leur attribuant les opérations et les données qui les concernent. Ces informations supplémentaires doivent être déduites du DFD accompagnant le réseau de Petri de contrôle.

Cette opération n'est pas compliquée. En fait, le réseau de Petri de contrôle gère l'activation des transformations de données du DFD qui l'accompagne. Le fait d'extraire un sous-réseau de Petri relatif à un P-invariant permet d'isoler certaines transformations de données : celles qui sont gérées par les transitions ou les places de ce sous-réseau. Ces transformations de données sont les opérations associées à l'objet considéré, et les données qu'elles échangent sont les attributs du même objet.

Les opérations qui se retrouvent rattachées à des transitions partagées entre deux portions de réseaux de Petri relatifs à des objets différents définissent les points de communication entre objets. Dans ce cas, il faut définir l'objet appelant et l'objet offrant l'opération. Cet aspect ne peut pas être déduit directement des diagrammes, il faut le clarifier à partir du contexte du problème.

Par ailleurs, certaines opérations, a priori du même type, se trouvent rattachées à un même objet (c'est le cas des opérations *usiner pièce A* et *usiner pièce B* dans l'exemple suivant, figure *IV.10*). Dans ce cas il faut se fier aux mini-spécifications de ces opérations pour vérifier si elles sont parfaitement identiques. Ainsi, il est possible d'éviter la duplication des opérations dans les objets et les PNOBCS des objets concernés s'en trouvent simplifiés.

IV.3.2.3 Les invariants de transitions

Les réseaux de Petri modélisant la partie dynamique des spécifications comportent généralement des invariants de transitions relatifs à des cycles d'opérations. Les cycles peuvent correspondre à des opérations d'un même objet mais peuvent également mettre en jeu des opérations appartenant à des objets distincts. Ainsi, il n'est pas possible de s'appuyer systématiquement sur les invariants de transitions pour définir les objets intervenants dans la spécification.

Cependant, les invariants de transitions peuvent servir à regrouper les objets fils d'un objet père. En effet, dans la majorité des cas, le PNOBCS d'un objet est couvert par un ou plusieurs T-invariants. (En fait, dans ces cas, l'objet possède une «position d'équilibre» à laquelle il a tendance à revenir après avoir exécuté un cycle d'opérations.) Inversement, si dans un réseau de Petri deux objets identifiés ont un comportement cou-

vert par le même T-invariant, ils sont candidats à un regroupement pour la formation d'un objet père. Ce regroupement devient d'autant plus pertinent si le sous-réseau de Petri relatif au T-invariant est aussi couvert par un P-invariant (même non minimal). C'est une approche ascendante pour la composition des objets.

Ainsi, la détermination des objets peut se faire en deux étapes. Tout d'abord une décomposition du réseau de Petri à l'aide des P-invariants pour la localisation des objets. Puis, éventuellement, un regroupement partiel de certains objets au sein d'objets pères à l'aide des T-invariants.

IV.3.2.4 Exemple d'étude

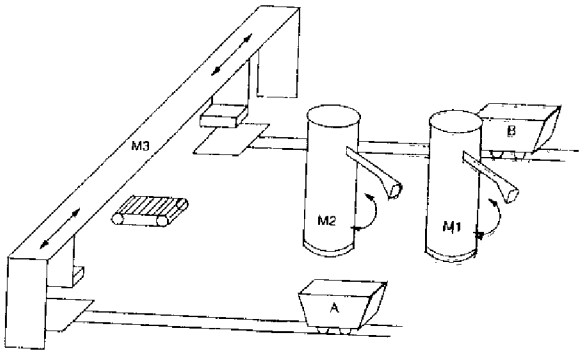


fig IV.9 - Exemple2 - Atelier de production

L'exemple suivant montre les spécifications SA-RT d'une partie d'un atelier qui s'occupe de l'usinage et de l'assemblage de pièces. Les pièces sont de deux types (A et B), leur traitement commence dans une autre partie de l'atelier (cet exemple ne modélise qu'une partie des opérations). Quand elles arrivent, elles sont montées sur des chariots filoguidés. Chaque pièce subit une opération d'usinage et une opération de polissage en restant sur les chariots, puis les pièces sont déposées par ces derniers sur les tables d'assemblage de la machine M_3. Les chariots sont libérés et les pièces sont assemblées pour donner le produit final qui est évacué par un tapis roulant. Trois machines assurent ces différentes tâches. La machine M_1 assure l'opération d'usinage, la machine M_2 l'opérations de polissage et la machine M_3 l'assemblage. Les machines M_1 et M_2 sont fixées entre les chemins de passage des chariots et peuvent tourner pour traiter les pièces sur les deux cotés (figure *IV.9*).

La figure *IV.10* montre un niveau des spécifications[2] SA-RT : la partie statique, à gauche, et le contrôle spécifié par un réseau de Petri, à droite. Dans cet exemple n'est traitée qu'une partie des spécifications, les détails n'apparaissent pas.

Le réseau de Petri de contrôle servira de support pour l'identification des objets. Comme l'indique la figure *IV.11*, la recherche de composantes conservatives dans ce

2. Il est à rappeler que ce modèle représente les besoins : il montre qu'il y a un conflit au niveau de l'accès à chacune des machines mais ne donne pas de politique d'allocation de ressources pour résoudre ce conflit.

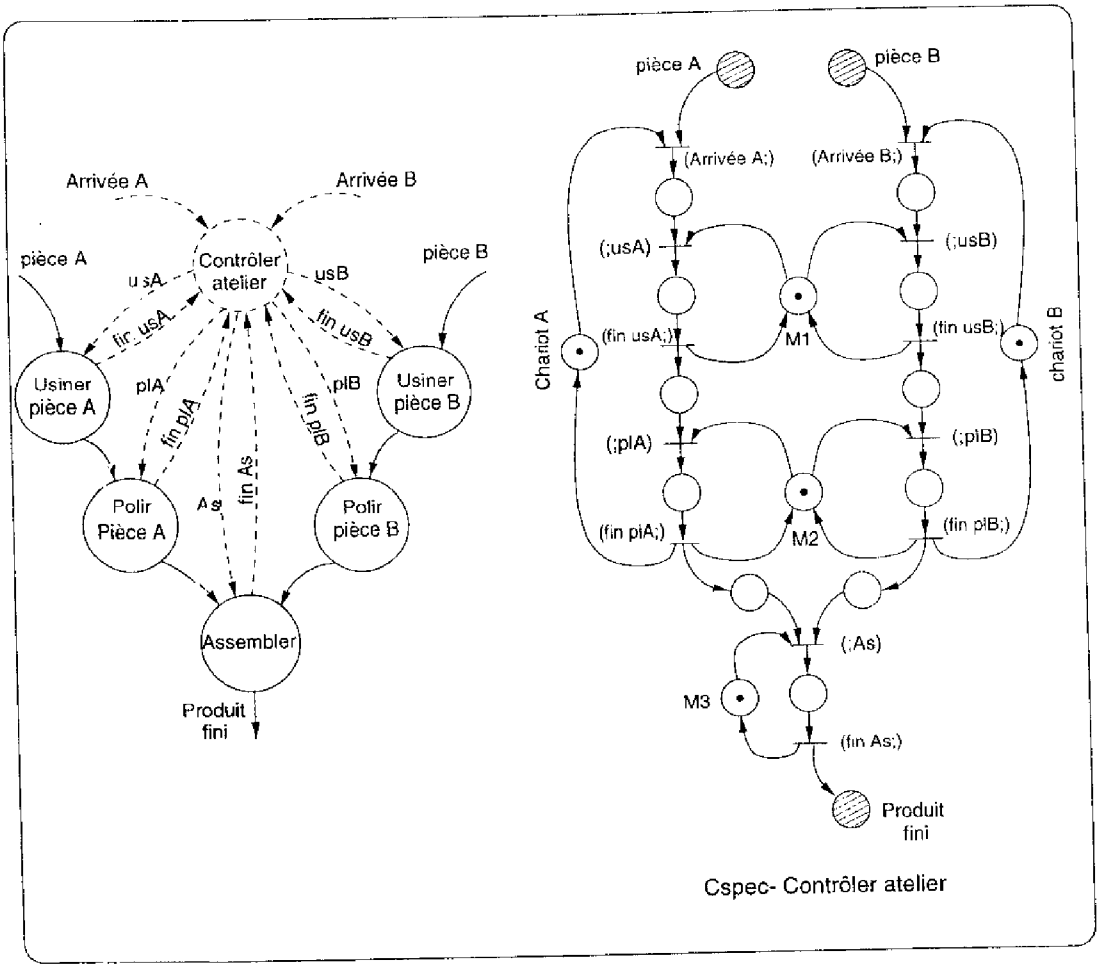


fig IV.10 - Exemple2- Atelier de Production - spécifications

réseau révèle l'existence des P-invariants monomarqués[3] suivants (fig *IV.11*) :

$$
\begin{aligned}
C_1 &\longrightarrow P_4, P_5, P_6, P_7, P_8 \\
C_2 &\longrightarrow P_9, P_{10}, P_{11}, P_{12}, P_{13} \\
C_3 &\longrightarrow P_6, P_{11}, P_{14} \\
C_4 &\longrightarrow P_8, P_{13}, P_{15} \\
C_5 &\longrightarrow P_{18}, P_{20}
\end{aligned}
$$

Ces P-invariants donnent de bons candidats pour une représentation par objets, d'autant plus qu'ils se rapportent à des entités physiques du système étudié. Les objets pressentis sont relatifs aux machines et aux chariots : **M1**, **M2**, **M3**, **ChariotA** et **ChariotB**.

A ce stade, il faut extraire le comportement de chacun de ces objets et spécifier les

3. Dans un P-invariant monomarqué la somme du marquage de toutes les places est toujours égale à un.

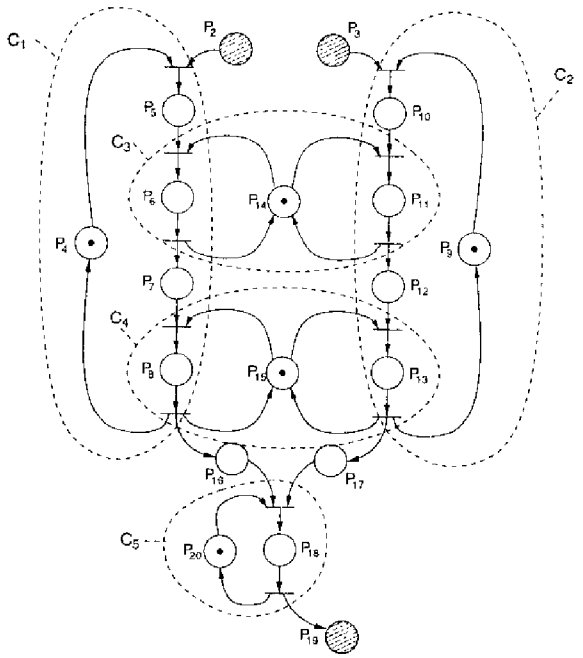


fig IV.11 - Exemple2- P-invariants

points de communications entre les objets. Cette tâche est conduite selon les propositions faites dans [Paludetto 91]. La figure *IV.12* donne le résultat de cette opération.

Ayant défini les objets par leurs comportements et leurs interactions, il faut les compléter en leur attribuant les données et les opérations qui les concernent. Ces opérations sont extraites du DFD représentant la partie statique. En fait, les opérations gérées par les différents réseaux de Petri de contrôle se réfèrent aux transformations de données du DFD qui l'accompagne.

Dans cet exemple les opérations *usiner pièce A* et *usiner pièce B* se trouvent partagées entre les PNOBCS des objets **machines** et des objets **pièces**. Il est logique, dans ce cas, que l'objet **machine** offre l'opération *usiner*.

Par ailleurs, l'objet **machine M1** se retrouve avec deux opérations a priori identiques : *usiner pièce A* et *usiner pièce B*. Il faut examiner les mini-spécifications de ces opérations pour voir s'il s'agit exactement du même type d'usinage. Pour cet exemple, supposons que les deux opérations soient identiques. Dans ce cas, l'objet **machine M1** offre une seule opération *usinage*. Ainsi, il existe un conflit pour l'accès à cette ressource **machine M1**. L'étape d'analyse des besoins doit préciser la politique de partage de cette ressource commune. Si cette politique a été précisée lors de l'étape d'analyse des besoins fonctionnels (utilisation d'une file d'attente par exemple), le passage vers une spécification orientée objet doit garder la même politique. En revanche, si cette politique n'a pas été précisée, le passage vers la spécification orientée objet doit garder entière l'ambiguïté. Cette dernière doit être levée ultérieurement.

La même démarche est suivie pour les autres objets. Le résultat final de la spécifi-

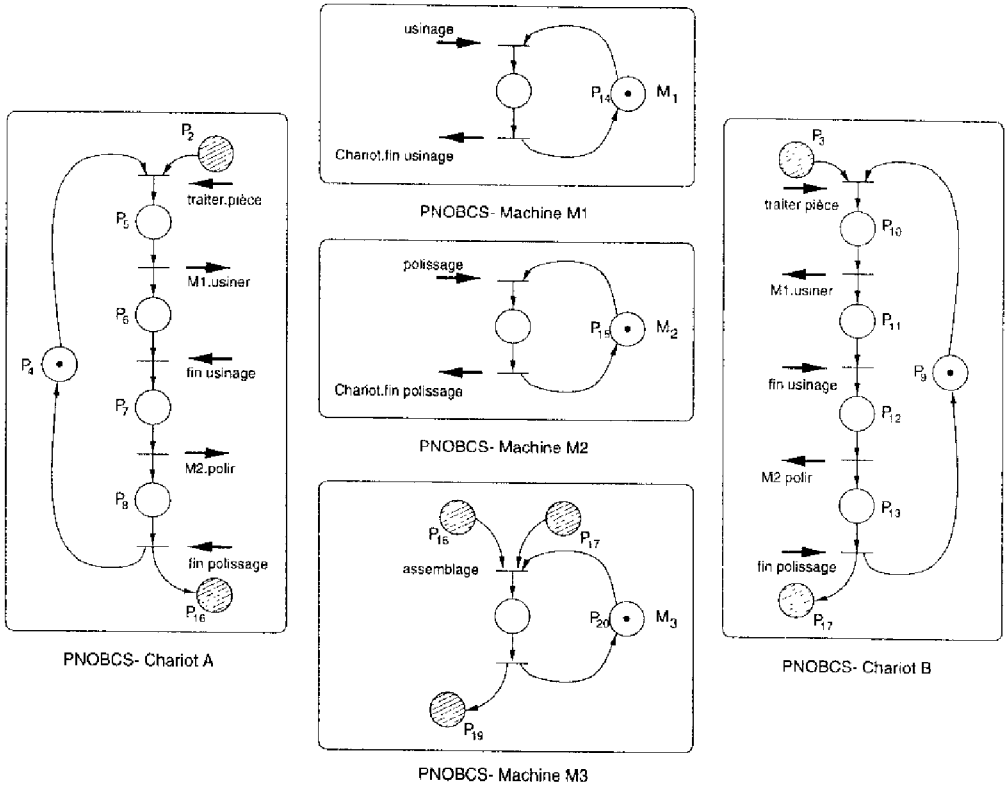


fig IV.12 - Exemple2- PNOBCS des différents objets

cation en objets de cet exemple est représenté dans la figure *IV.13*, les PNOBCS étant ceux de la figure *IV.12*. Remarquons que les objets **ChariotA** et **ChariotB** ont les mêmes spécifications. Alors, il est possible (si cela n'a pas déjà été fait) de définir une classe **Chariot**.

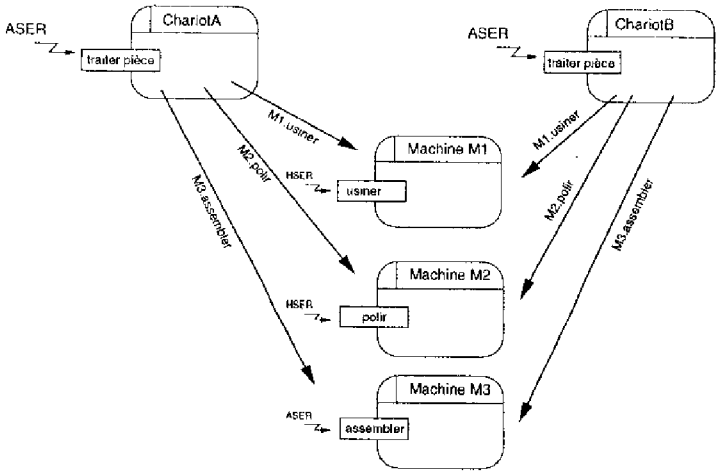


fig IV.13 - Exemple2- Représentation en objets

IV.3.3 Quelle approche adopter?

Dans la partie précédente, nous avons montré que la représentation en objets des spécifications fonctionnelles pouvait s'appuyer autant sur la représentation par réseaux de Petri de la partie statique (DFD) que sur l'exploitation du réseau de Petri de la partie dynamique (le contrôle). En fait, l'utilisation de l'une ou l'autre de ces approches dépend fortement de la nature du système étudié et de la démarche fonctionnelle adoptée pour établir les spécifications.

IV.3.3.1 La nature du système étudié

Approche basée sur l'aspect statique

Dans les systèmes d'information, et dans certains types de systèmes temps réel, la composante dynamique est marginale par rapport à la composante statique. Le contrôle est presque évident et l'essentiel des spécifications s'occupe des opérations effectuées et des données échangées. Dans ce cas, la recherche des objets à partir des composantes conservatives des réseaux de Petri représentants les DFDs semble être l'approche la plus adéquate. Cette approche peut être adoptée pour aller vers une représentation HOOD/PNO dans laquelle les objets identifiés seront majoritairement des objets passifs.

La même approche peut être également adoptée pour déduire une représentation à l'aide d'une méthode plus orientée donnée comme, par exemple, la méthode *Object Modeling Technique* (OMT). En effet, OMT ne donne pas de démarche précise pour allouer aux objets les opérations qui les concernent et qui sont représentées par des diagrammes de flots de données dans la vue fonctionnelle. Ainsi, la méthode que nous proposons pourrait servir de support à cet aspect laissé à la charge de l'analyste-concepteur dans OMT.

Approche basée sur l'aspect dynamique

Pour les systèmes temps réel où l'aspect dynamique est l'aspect dominant, nous remarquons que les transformations de données sont totalement asservies au réseau de Petri de contrôle qui gère leur fonctionnement et qui ne leur laisse aucune autonomie. C'est le cas de l'exemple présenté dans le paragraphe §IV.3.2.4. Dans de tels systèmes, il paraît plus intéressant de s'appuyer sur le réseau de Petri modélisant le contrôle pour la décomposition en objets. D'ailleurs, la méthodologie HOOD/PNO s'appuie aussi sur la structure de contrôle (PNOBCS reflétant l'aspect dynamique) de l'objet père pour rechercher ses objets fils.

Approche mixte

Les systèmes étudiés ne sont pas toujours exclusivement des systèmes à forte composante dynamique ou alors des systèmes à forte composante statique. La complexité des systèmes temps réel est telle que l'aspect dynamique et l'aspect statique sont d'égale importance: d'une part, le contrôle et la gestion des activités du procédé et, d'autre part, le traitement des informations internes (système de commande) et externes (en-

vironnement). Ainsi, une approche mixte basée sur l'identification des objets à partir de la description statique et de la description dynamique s'impose.

En fait, les deux aspects se complètent. A partir de chacun d'eux, il est possible d'identifier des objets que l'autre aspect ne permet pas de voir. Ainsi, l'aspect dynamique permet notamment d'identifier les objets relatifs à la gestion des activités du procédé (qui n'apparaissent pas toujours dans la partie statique) comme la commande des machines, des chariots, la gestion des capteurs, etc. Par ailleurs, l'aspect statique permet de déceler les objets (plutôt passifs) qui sont relatifs à certaines opérations de traitements internes au système informatique de commande (qui n'apparaissent pas systématiquement dans la partie dynamique) comme la gestion des fichiers, la mémorisation de certaines variables, le traitement de l'information, etc.

Proposition

Dans une spécification où les descriptions statique et dynamique sont d'égale importance, il faut mener l'identification des objets à partir des réseaux de Petri représentant les DFDs et des réseaux de Petri représentant le contrôle puis, examiner la pertinence des objets candidats obtenus :

- Le choix de certains objets est conforté quand ils sont retrouvés à partir des descriptions statique et dynamique grâce aux composantes conservatives. Dans ce cas, la description statique définit les opérations et les attributs de l'objet et la description dynamique définit son comportement. C'est un cas idéal pour la spécification à objets.
- Certains objets sont d'abord définis par leurs données et leurs opérations, en s'appuyant sur les composantes conservatives de la description statique. Ils sont éventuellement complétés par leurs comportements déduits (mais non extraits directement) de la description dynamique. Le comportement peut également être composé à partir d'informations venant de la partie dynamique et d'autres émanant de la partie statique (qui contient une faible composante de contrôle, cf. §IV.3.1.7).
- D'autres objets sont d'abord définis par leurs comportements, en s'appuyant sur les P-invariants de la description dynamique. Ils sont complétés par leurs opérations et leurs données déduits des spécifications statiques (cf. §IV.3.2.2).

■

Ainsi, de la nature du système étudié dépend le choix des modèles à utiliser (statique ou dynamique). Ce choix est laissé à l'appréciation de l'analyste-concepteur qui s'appuiera sur sa connaissance du système.

IV.3.3.2 Exemple

L'exemple suivant illustre l'application d'une démarche double (à partir des aspects statique et dynamique) pour l'identification et la définition des objets. Il s'agit d'un automate de distribution de carburant composé d'un tiroir pour recevoir des billets de 50 et 100 Francs, d'un afficheur du crédit disponible, de pistolets pour verser le carburant et d'une imprimante. La figure *IV.14* montre le premier niveau des spécifications

fonctionnelles par SA-RT de cet automate. Des spécifications plus complètes peuvent être consultées dans [Benzina et Paludetto 96b].

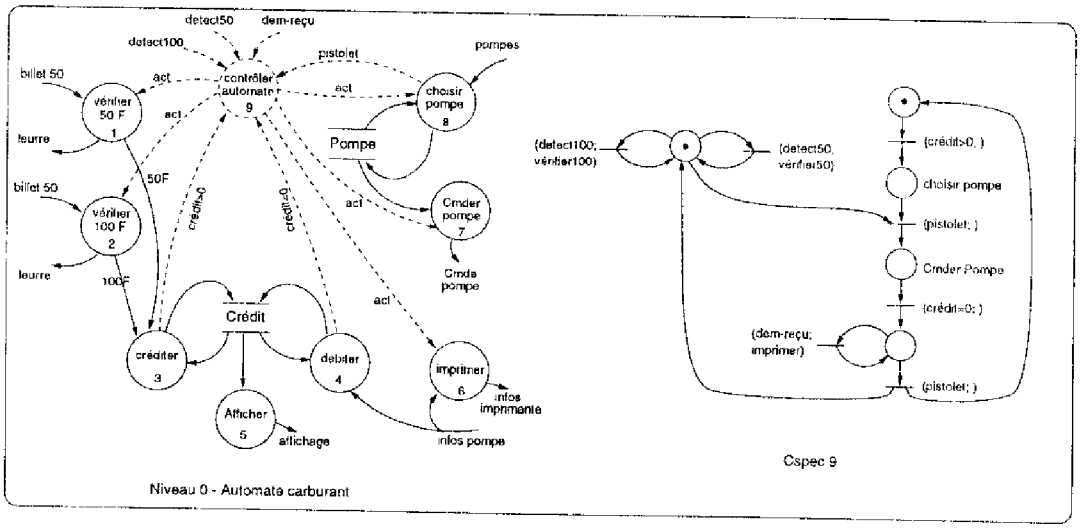


fig IV.14 - Spécifications d'un automate de distribution de carburant

La figure *IV.15* montre la représentation par réseau de Petri du DFD et les composantes conservatives qui peuvent servir pour l'identification des objets. Un premier examen nous permet de distinguer 4 objets pertinents : **Tiroir** (composante C_1), **Crédit** (composante C_2), **Imprimante** (composante C_4) et **Pompe** (composantes C_5, C_6 et C_7). Les composantes C_3 et C_8 n'ont pas été retenues puisque les seules opérations qu'elles couvrent sont inclues dans C_2.

La figure *IV.16* indique les P-invariants qui peuvent être extraits du réseau de Petri représentant le contrôle. Ces P-invariants fournissent deux objets candidats, il s'agit des objets **Pompe** (composante C_9) et **Tiroir** (composante C_{10}).

L'étude de cet exemple montre que certains objets candidats sont retrouvés à partir du réseau de Petri de contrôle et à partir de celui représentant le DFD : objet **Pompe** et objet **Tiroir**. Dans ce cas, le réseau de Petri de contrôle fournit le comportement de l'objet et le réseau de Petri représentant le DFD fournit les opérations et les données.

D'autres objets ne sont identifiés qu'à partir du réseau de Petri représentant le DFD : l'objet **Imprimante** (objet passif) et l'objet **Crédit** qui correspond à un traitement interne de l'information.

Les figures *IV.17* et *IV.18* donnent la spécification par objet et les PNOBCS retenus. Remarquons que les informations disponibles à partir des spécifications fonctionnelles ne sont pas complètes, elles ne montrent pas comment se fait le test du crédit. De ce fait, les PNOBCS obtenus ne sont pas explicites sur la nature de cette communication entre les objets **Pompe** et **Crédit**. Il faut approfondir l'étude des spécifications fonctionnelles (à partir des sous-niveaux SA-RT, ou à partir des informations textuelles des mini-spécifications) pour mieux préciser les PNOBCS des différents objets.

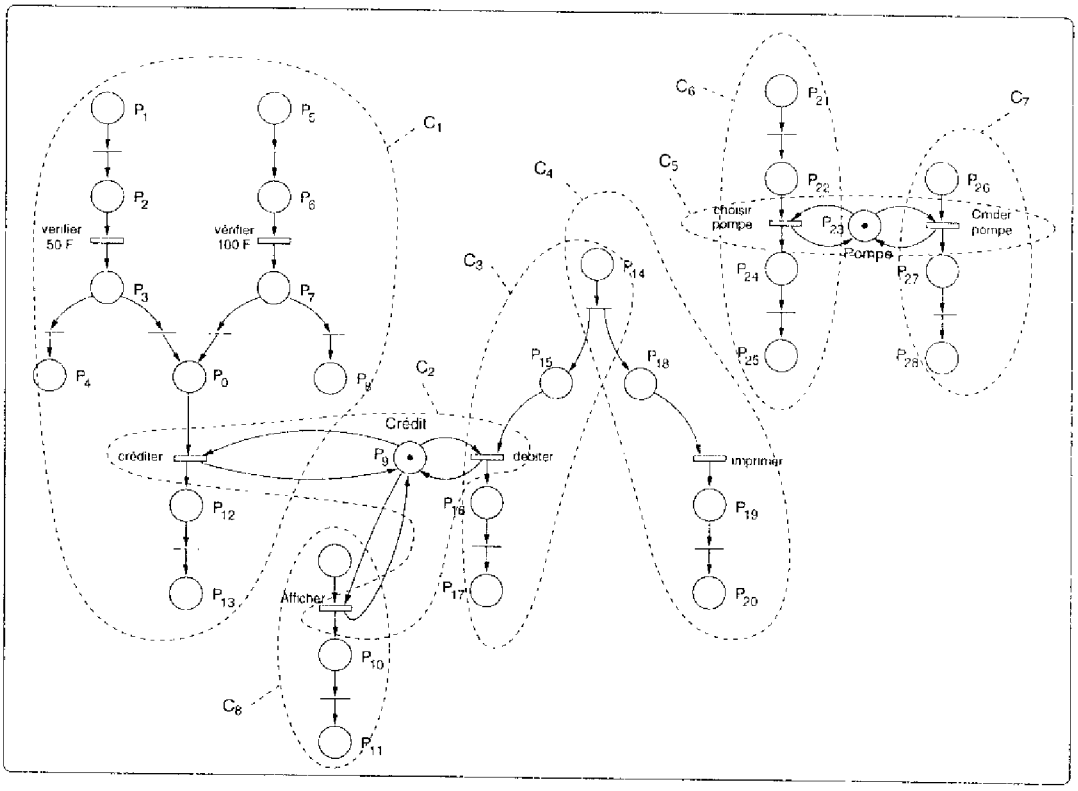


fig IV.15 – Automate - Réseau de Petri représentant le DFD

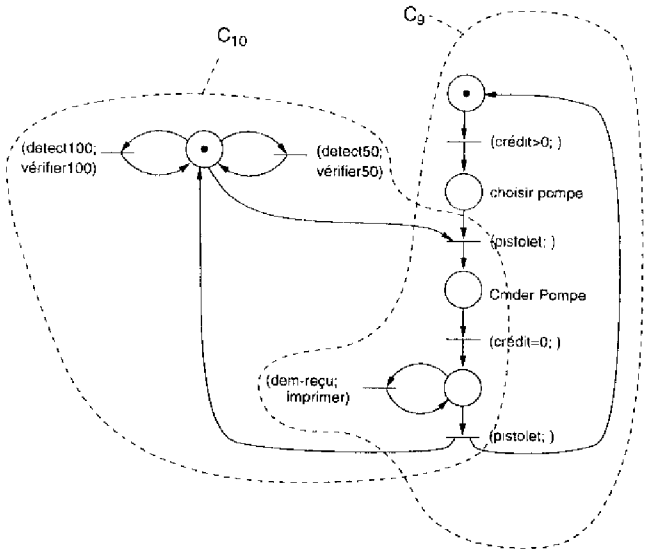


fig IV.16 - Automate - P-invariants du réseau de Petri de contrôle

IV.3.3.3 Le poids de la démarche fonctionnelle considérée

Il ne suffit pas de spécifier un système avec le symbolisme des diagrammes SA-RT pour affirmer que la démarche adoptée est une démarche fonctionnelle. Il est par-

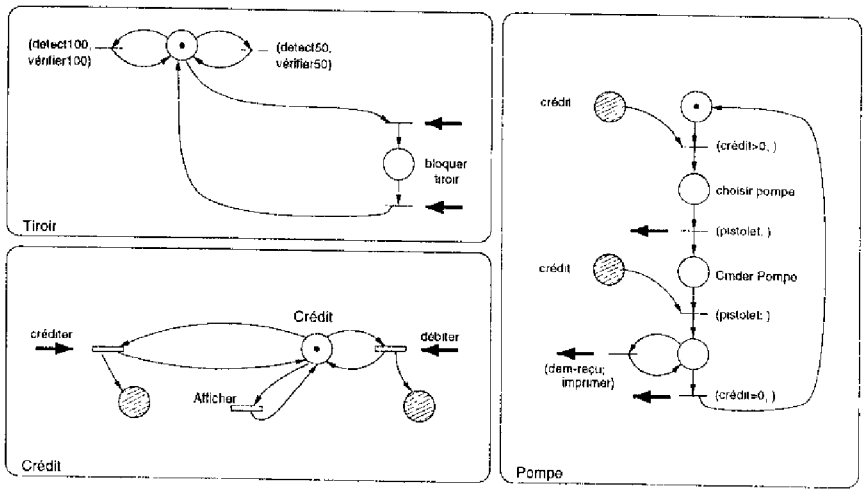


fig IV.17 – Automate - PNOBCS des objets retenus

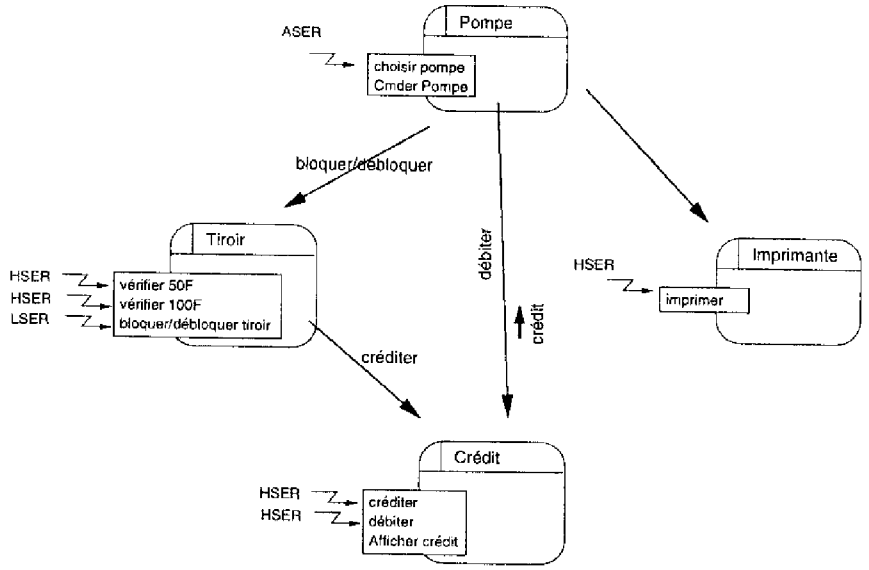


fig IV.18 – Automate - Spécification par objets

faitement possible d'adopter une démarche orientée objet tout en conservant l'apparence d'une démarche fonctionnelle (ou l'inverse). D'ailleurs, P. T. Ward affirme dans [Ward 89] qu'il est possible d'intégrer les concepts orientés objet dans les méthodes d'analyse et de conception fonctionnelles. Ward propose une démarche médiane qui offre certaines possibilités de réutilisabilité mais qui garde un contrôle centralisé.

Donc, avec un même symbolisme, les spécifications fonctionnelles peuvent diverger selon la démarche fonctionnelle adoptée. Par conséquent, le passage d'une spécification fonctionnelle à une spécification orientée objet dépend fortement du mode de réflexion de l'analyste qui le conduit à établir les spécifications fonctionnelles. En effet, si les spécifications ont été établies avec une "arrière pensée objet" (contrôle distribué et une

transformation de données par objet) le passage à une spécification objet se limitera à l'association d'une fonction et d'une structure de contrôle à chaque objet. En revanche, si les spécifications sont le résultat d'une véritable démarche fonctionnelle[4], le travail consistera à collecter les informations relatives à chaque objet (les données, les diverses transformations de données, une partie du contrôle...).

La figure *IV.19* montre une partie des spécifications d'un problème analogue à celui traité dans le paragraphe §IV.3.2.4. Cette façon d'établir les spécifications rend le passage vers un modèle objet presque immédiat. En effet, les spécifications ont été centrées sur les ressources (machines M1 et M2) dès le premier niveau. Ainsi, chacune des transformations de données du premier niveau est relative à un objet, chacune d'elles comprend un sous-niveau avec les opérations et le contrôle de chaque objet.

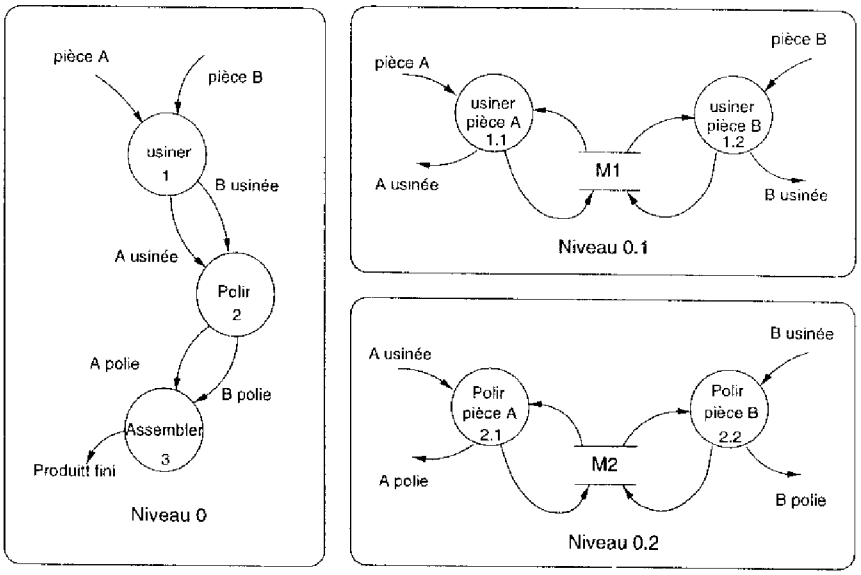


fig IV.19 – Exemple3- Spécification fonctionnelles avec des considérations objets

Il en ressort que le jugement de l'analyste-concepteur qui mène la translation vers une spécification à objet est essentiel pour sélectionner la meilleure démarche à suivre en fonction des spécifications SA-RT dont il dispose et en fonction du système qu'il étudie.

IV.4 Notions complémentaires

IV.4.1 La hiérarchie dans SA-RT

Dans les propositions faites dans ce chapitre, la hiérarchie n'a pas été explicitement évoquée. Dans une spécification fonctionnelle la hiérarchie possède un rôle de struc-

4. Une démarche est purement fonctionnelle quand le leitmotiv de l'analyste-concepteur, tout au long de l'étude, est la recherche des fonctions réalisées par le système.

turation primordial. Cependant, cette structuration est basée sur une hiérarchie de fonctions plutôt que sur une hiérarchie d'objets ou de données. Étant donné que les objets conduisent souvent à une vue verticale d'une structuration hiérarchique fonctionnelle, certains auteurs ont proposé une remise à plat de tous les niveaux hiérarchiques avant de commencer la décomposition en objets [Shumate 91].

Dans cet ordre d'idées, il est possible de décrire par un réseau de Petri le résultat d'une mise à plat d'une hiérarchie de diagrammes DFDs. Par ailleurs, il est également possible d'affiner toutes les transitions représentant les transformations de données non terminales, ce qui correspond aussi à la mise à plat des différents niveaux hiérarchiques. Cependant, cette mise à plat systématique est assez coûteuse en termes de difficulté de recherche des composantes conservatives et d'interprétation des résultats trouvés car elle conduit à des réseaux de taille prohibitive.

Cette solution de mise à plat systématique et totale nous apparaît comme une solution ultime, nous lui préférons une approche par niveau. En effet, il n'est pas toujours nécessaire de **tout** mettre à plat pour pouvoir procéder à une décomposition par objets. Généralement, dès les premiers niveaux hiérarchiques intervenants dans une spécification SA-RT, il est possible de commencer la décomposition en objets. Cette décomposition aboutit aux objets pères. Si une transformation de données est allouée à un objet père, les sous-niveaux hiérarchiques obtenus en décomposant cette transformation appartiendront au même objet, ils serviront à leur tour à constituer les objets fils.

Il faut alors étudier s'il y a lieu de décomposer une transformation de données. Il n'existe pas un algorithme systématique pour décider dans quelle mesure il faut pousser la mise à plat des différents niveaux. Tout au plus, il est possible de dégager certaines propositions informelles obtenues par la pratique de cette démarche.

Proposition

- Le diagramme de contexte ne peut révéler que l'existence d'objets relatifs aux sources et puits de données, il est trop réducteur et il est nécessaire de le décomposer pour affiner la recherche.
- A partir du niveau zéro, il est possible de commencer l'exploration des composantes conservatives :
  - Si la décomposition en composantes conservatives aboutit à une association d'un objet à chaque transformation de données (sur-décompostion), la hiérarchie obtenue est plutôt du type fonctionnel. Généralement, cela signifie que le niveau de détail n'est pas suffisant. Si cette décomposition ne correspond pas à la décomposition physique du système étudié, il faut pousser encore la mise à plat et reprendre la recherche des composantes conservatives.
  - Si la seule possibilité est de regrouper toutes les transformations du niveau dans un seul objet (sous-décomposition), alors il est probable que le niveau de détail ne soit pas encore suffisant.
  - Si les composantes conservatives trouvées permettent de dégager quelques objets en regroupant certaines données et certaines transformations, alors il

faut examiner la pertinence de ces objets par rapport au contexte du problème.

- Chaque objet retrouvé correspondra à un objet père, il faut alors reprendre la même démarche avec les niveaux inférieurs pour essayer de dégager, le cas échéant, ses objets fils.

■

IV.4.2 Classes et Héritage

Les concepts de classes et d'héritage sont des concepts de structuration propres aux méthodologies orientées objet. La classe permet de créer un modèle pour les objets ayant des caractéristiques identiques. Par ailleurs, la définition des classes permet de les utiliser dans d'autres problèmes traitant des mêmes entités physiques (réutilisabilité). L'héritage permet de créer de nouvelles classes à partir de celles déjà définies.

Dans le processus d'identification des objets, ces concepts n'ont pas été abordés. En fait, dans la phase d'analyse des besoins, le concept de classe n'est pas primordial. Pendant cette phase, l'analyste-concepteur modélise le système par des objets. Ces derniers peuvent servir éventuellement à créer des classes, mais ce n'est pas la préoccupation principale en phase d'analyse des besoins. Les classes sont beaucoup plus utiles en phase de conception et surtout en conception détaillée et en implémentation pour faciliter le travail de programmation.

Ainsi définis, ces concepts ne sont pas fondamentaux dans le passage d'une spécification fonctionnelle vers une spécification orientée objet. Le résultat de ce passage étant l'identification et la définition des objets, qui peuvent être des instances de classes. Si l'analyste le désire, il peut définir des classes à tout moment comme suggéré dans HOOD/PNO, pour participer à l'élaboration de l'«Object Oriented Domain Analysis» (OODA).

Plus particulièrement, les classes peuvent intervenir dans une démarche ascendante si, à partir de l'identification des objets d'une spécification fonctionnelle, l'analyste concepteur entrevoit la possibilité de réutiliser des classes déjà définis dans l'OODA.

IV.4.3 Le formalisme textuel

Le passage des spécifications fonctionnelles aux spécifications orientées objet a été conduit en s'appuyant sur des formalismes graphiques. Toutefois, les diagrammes ne suffisent pas pour établir les spécifications. Certes, ils forment un support et un guide mais à défaut de tout décrire, ils doivent être complétés par un formalisme textuel (mini-spécifications et dictionnaire de données pour les spécifications fonctionnelles et description textuelle des opérations et des attributs des objets pour les spécifications orientées objet : ODS- Object description skeleton et OCS- Object class specifications dans HOOD/PNO). Le formalisme textuel donne un complément d'information, plus détaillé, sur ce qui a été décrit globalement par le formalisme graphique.

Ce formalisme textuel ne doit pas être absent du passage entre les spécifications fonctionnelles et les spécifications orientées objet. La translation des formalismes textuels

doit se faire en se basant sur les associations établies grâce aux diagrammes. Ainsi les attributs des objets seront décrits à partir des informations répertoriées dans le dictionnaire des données. Par ailleurs, la définition des opérations, leurs PNOPCS (Petri Net Operation Control Structure) et leurs ODS devra être déduite des mini-spécifications des transformations de données.

IV.5 Limites de l'utilisation des RdP pour la décomposition en objets

IV.5.1 Limites et Difficultés

L'utilisation des réseaux de Petri pour le passage d'une spécification fonctionnelle à une spécification orientée objet présente certaines limites. La démarche ne peut pas être conduite sans l'assistance d'une personne qui connaît bien le contexte du problème pour établir les choix nécessaires à chaque étape.

Elle comporte plusieurs difficultés, notamment la sélection des composantes conservatives. En effet, un réseau de Petri peut être couvert par plusieurs composantes conservatives. Il faut toute l'expertise de l'analyste-concepteur pour effectuer un choix adapté au contexte du problème.

Par ailleurs, certaines spécifications fonctionnelles sont tellement complexes, ou complexifiées par la démarche utilisée pour les établir (cf. §IV.3.3), qu'il devient très difficile de retrouver des composantes conservatives pertinentes dans les réseaux de Petri les représentants. Dans ce cas, la recherche des composantes conservatives ne garantit pas l'identification de tous les objets intervenant dans les spécifications. Il faut donc enrichir cette démarche en l'associant à d'autres techniques d'identification des objets pour conforter le choix des objets et pour identifier les objets qui auraient échappés à la méthode utilisant les réseaux de Petri. Nous aborderons cela dans le paragraphe §IV.5.2.

En outre, il existe des difficultés liées au calcul matriciel pour la recherche des composantes conservatives. Dans ce mémoire, le problème du calcul des composantes conservatives n'a pas été abordé parce qu'il a fallu limiter les axes de recherche, mais il n'est pas toujours évident à résoudre, particulièrement quand la taille des réseaux de Petri devient considérable. Plusieurs travaux de recherche ont été conduits pour aboutir à des algorithmes efficaces, citons [Alaiwan et Toudic 85], [Silva *et al.* 85], [Paludetto 91].

Enfin, l'utilisation des réseaux de Petri pour le passage des spécifications fonctionnelles vers une représentation orientée objet n'est pas encore une démarche systématique. Cette démarche a prouvé son bien-fondé dans les cas expérimentaux étudiés. Cependant, il n'y a que la pratique industrielle intensive pour établir ses limites, lui donner une bonne assise et l'élever au rang de démarche systématique.

IV.5.2 Utilisation des réseaux de Petri avec d'autres techniques

Nous avons vu que la recherche des composantes conservatives donne de bons candidats pour le partitionnement en objets des spécifications fonctionnelles mais pose parfois quelques difficultés. Dans ces cas, des techniques complémentaires sont bien utiles.

Parmi les techniques d'identification des objets, présentées au chapitre II de ce mémoire, certaines peuvent être complémentaires à la recherche des composantes conservatives, plus particulièrement citons :

- La recherche des entités physiques du système étudié.
- Les sources et puits d'informations du diagramme de contexte.
- L'étude de la composition physique du système en utilisant les diagrammes entités-associations.

En effet, à chaque étape de la répartition des spécifications fonctionnelles en objets, l'analyste-concepteur peut confronter les choix indiqués par la recherche des composantes conservatives, avec les objets identifiés par les autres techniques présentés plus haut. Quand une opération se trouve partagée entre deux objets c'est la conservation d'une forte cohésion interne dans les objets et le respect de la décomposition physique du système étudié qui doivent guider le choix de l'allocation de cette opération à un objet ou un autre. Ainsi, l'utilisation de différentes techniques pour l'identification des objets peut s'avérer une garantie supplémentaire pour une bonne définition des objets.

IV.6 Récapitulation de la démarche

L'étude menée tout au long de ce chapitre, nous permet d'établir un guide des étapes à suivre pour la translation d'une analyse fonctionnelle du système étudié, exprimée par des spécifications SA-RT, à une spécification orientée objet, par le formalisme HOOD/PNO :

1. Étudier le diagramme de contexte des spécifications SA-RT pour identifier les objets à partir des sources et puits d'informations.
2. Étudier la composition physique du procédé pour identifier un maximum d'objets physiques qui peuvent servir à la spécification. Les comparer avec les objets obtenus à l'étape précédente.
3. Examiner la nature des spécifications disponibles et la nature du système traité pour décider s'il est intéressant de s'appuyer d'avantage sur l'aspect statique ou sur l'aspect dynamique (cf. §IV.3.3).

4. Considérer un premier niveau des spécifications SA-RT (DFD et contrôle associé). Représenter ce niveau par des réseaux de Petri : un premier pour le DFD et un autre pour le contrôle (s'il n'est pas déjà modélisé par un réseau de Petri).

5. Après une mise en forme du réseau de Petri représentant l'aspect statique (cf. §IV.3.1.3 et §IV.3.1.6), effectuer une recherche des composantes conservatives à partir des deux types de réseaux de Petri obtenus.

6. Examiner la pertinence des objets candidats obtenus par rapport aux objets identifiés aux deux premières étapes et par rapport aux objets conceptuels en rapport avec le système étudié.

7. Dans le cas où la répartition en objet n'est pas satisfaisante, effectuer une première décomposition des transformations à un niveau moins abstrait et reprendre l'étude à l'étape 5.

8. Dans le cas où des objets pertinents ont été localisés grâce à des composantes conservatives du réseau de Petri représentant le DFD ou celui représentant le contrôle alors :

   - Si un objet est retrouvé à partir du réseau de Petri représentant le DFD et du réseau de Petri de contrôle, la partie extraite du DFD indique les opérations et les données de l'objet et la partie extraite du réseau de Petri de contrôle indique son comportement (cf. exemple §IV.3.3.1).
   - Si un objet est retrouvé seulement à partir du réseau de Petri représentant le DFD, il est probable qu'il soit un objet passif. La partie extraite du DFD indique les opérations et les données mises en jeu. Toutefois, il faut examiner les sous-niveaux et les mini-spécifications des opérations pour recomposer le comportement de l'objet s'il y a lieu (cf. §IV.3.1.7).
   - Si un objet est retrouvé seulement à partir du réseau de Petri de contrôle, c'est un objet actif. Il faut le compléter par les opérations qui le concernent à partir du DFD (cf §IV.3.2.2). Si aucune des opérations du DFD n'est contrôlée par le réseau de Petri extrait, alors il s'agit d'un objet qui s'occupe exclusivement de coordonner certains événements et actions du système. Il est possible aussi que les opérations qui lui sont associées n'aient pas été représentées au niveau du DFD (trop simples), dans ce cas il faut compléter la spécification de l'objet en mentionnant ces opérations.

9. Quand les différentes composantes des réseaux de Petri ont été réparties sur les objets, examiner les opérations communes entre les différents objets et clarifier la nature de l'opération pour chaque objet (offerte ou demandée) en se fiant à la répartition physique des entités du système étudié et en pensant à maximiser la cohésion interne des objets (cf. §IV.3.1.5).

10. Identifier et clarifier la nature des communications entre objets (cf. §IV.3.1.8, §IV.3.2.4 et [Paludetto 91]).

11. Finaliser les PNOBCS de chaque objet, le diagramme objet et les descriptions textuelles des attributs et des opérations.

12. Pour les objets comprenant des opérations abstraites, décomposer ces opérations et reprendre à l'étape 5 pour rechercher les objets fils s'il y a lieu.

13. Quand toutes les répartitions sont faites examiner les spécifications par objets obtenues pour les valider par rapport aux spécifications fonctionnelles de départ.

Ayant effectué le passage vers des spécifications fonctionnelles, le travail doit être continué par des activités d'analyse orientée objet pour étudier les aspects propres à cette approche non traités dans les spécifications fonctionnelles.

IV.7 Perspectives d'automatisation de la démarche

Dans l'état actuel des connaissances, il est impossible de fournir une démarche complètement automatisée qui permette de traduire une spécification fonctionnelle en une spécification orientée objet. Tout au plus, il est possible d'élaborer des outils d'aide à l'analyste-concepteur pour l'assister dans sa tâche.

La démarche proposée, basée sur les réseau de Petri, présente l'avantage de comporter plusieurs activités automatisables, comme certains passages entre diagrammes SA-RT et réseaux de Petri ou alors, la recherche des composantes conservatives dans les réseaux de Petri.

Ainsi, il est possible d'établir un outil d'aide à l'analyste-concepteur pour passer des spécifications fonctionnelles aux spécifications orientées objet. Cet outil s'occuperait automatiquement des processus qui ne présentent aucune ambiguïté et se remettrait à l'avis de l'analyste-concepteur chaque fois qu'il est nécessaire de prendre des décisions qui dépendent du contexte du problème ou de la démarche choisie. Par exemple, lors de la représentation des diagrammes SA-RT par réseaux de Petri, chaque fois qu'une transformation de données avec deux flots de sortie se présente, le logiciel présenterait deux alternatives à l'analyste concepteur qui, connaissant le contexte du problème, choisirait la représentation adéquate (sortie en ET ou bien en OU exclusif).

Le calcul des composantes conservatives pourrait se faire automatiquement, grâce à un algorithme adéquat, et le logiciel présenterait à l'analyste concepteur les composantes conservatives possibles pour que ce dernier choisisse quelles sont les décompositions pertinentes pour le problème étudié.

Ainsi, l'interaction entre l'analyste-concepteur et le logiciel se limiterait essentiellement à des prises de décisions chaque fois que le logiciel doit faire un choix.

En outre, cette translation entre analyse fonctionnelle et analyse orientée objet permettrait d'effectuer un prototypage rapide des systèmes, automatisable grâce à un joueur de réseaux de Petri. Ainsi il est possible de faire une vérification à peu de frais de la représentation en objet par rapport aux spécifications fonctionnelles SA-RT.

Enfin, la translation entre spécifications fonctionnelles et spécifications orientées objet pourrait s'intégrer dans un environnement général, un atelier de génie logiciel, qui assisterait l'analyste-concepteur dans toutes les phases du cycle de vie du logiciel, et qui comporterait notamment les fonctions suivantes :

- Saisie et vérification des spécifications (SA-RT, RdP, HOOD/PNO).
- Assistance au passage des spécifications fonctionnelles aux spécifications orientées objet.
- Prototypage à l'aide d'un joueur de réseaux de Petri.
- Aide à la conception (HOOD et HOOD/PNO).
- Génération automatique de squelette de code.

Cet atelier de Génie Logiciel pourrait assister l'utilisateur dans un processus fonctionnel de bout en bout, dans un processus orientée objet de bout en bout, ou encore, dans un processus de spécifications fonctionnelles qui se continuent par une conception orientée objet.

La réalisation d'ateliers de génie logiciel utilisant des modèles graphiques suscite un intérêt croissant notamment pour les systèmes temps réel. Signalons, par exemple, l'environnement *LACATRE*, [Schwarz 92] adapté aux exécutifs temps-réel, qui génère des squelettes de code à partir d'un formalisme graphique. Les réseaux de Petri sont également utilisés comme outil graphique qui permet l'analyse et la validation comme dans le projet CASPAIM [Bourey et Gentina 91].

IV.8 Conclusion

La représentation des diagrammes SA-RT par réseaux de Petri et la recherche des composantes conservatives dans ces réseaux nous ont permis de définir une méthode permettant d'assister l'analyste-concepteur lors de la transcription des spécifications fonctionnelles en spécifications orientées objet. Cette méthode peut servir pour identifier et définir les objets. Elle présente l'avantage de prendre en compte les aspects statiques et dynamiques du système étudié, ce qui est capital pour les systèmes temps réel.

La méthode définie n'est pas entièrement automatisable mais comporte de larges parties où les activités peuvent être automatisées et l'intervention de l'utilisateur serait limitée aux choix nécessaires en tenant compte du contexte du problème. Cette méthode peut également être couplée avec les autres techniques d'identification des objets pour confronter les choix effectués.

Par ailleurs, les modèles et outils utilisés dans cette démarche, suggèrent de l'intégrer dans un environnement de génie logiciel plus complet qui permettrait d'assister

l'analyste concepteur dans les phases d'analyse, de conception et d'implémentation et plus particulièrement lors du passage entre spécifications fonctionnelles et spécifications orientées objet.

Enfin, l'utilisation des réseaux de Petri pour la description des spécifications SA-RT nous a permis d'entrevoir des perspectives de validation et d'évaluation des spécifications par les propriétés et les outils définis autour des réseaux de Petri. Nous explorons ces perspectives dans le chapitre suivant.

Chapitre V

Perspectives de validation et d'évaluation des spécifications SA-RT à l'aide des réseaux de Petri

L'utilisation des réseaux de Petri pour la représentation des spécifications SA-RT telle qu'elle a été exposée dans le troisième chapitre suggère plusieurs perspectives de validation. En effet, connaissant le potentiel des réseaux de Petri dans ce domaine et les travaux effectués sur la validation de ces derniers, il est intéressant d'explorer leurs propriétés pour la validation et l'évaluation des spécifications SA-RT. Ainsi, la démarche de translation est également l'occasion d'effectuer une validation ou une évaluation des spécifications.

La validation et l'évaluation des spécifications SA-RT constituent une garantie appréciable pour les concepteurs. Le but recherché par ces activités est double :

- Vérifier qu'il n'y a pas d'inconsistances dans les spécifications : ce qui revient à vérifier que les diagrammes ont été établis selon les règles d'utilisation de la méthode SA-RT.

- Vérifier qu'il n'y a pas d'incohérences : ce qui revient à vérifier que les spécifications du système étudié sont cohérentes entre elles et ne se contredisent pas.

Les réseaux de Petri peuvent être d'un apport appréciable pour atteindre ces objectifs. En effet, de part les analyses de propriétés possibles et de part les nombreux outils

construits autour d'eux, les réseaux de Petri peuvent amener des solutions de validations et d'évaluations aux spécifications SA-RT qu'ils représentent. Dans ce chapitre, nous examinerons certaines propositions permettant d'illustrer l'apport des réseaux de Petri pour la validation et l'évaluation des spécifications SA-RT.

La représentation de l'aspect dynamique des spécifications SA-RT par les réseaux de Petri s'apparente à la représentation de tout système à événements discrets et permet à l'analyste-concepteur d'accéder à toutes les possibilités offertes par ces derniers : validation par établissement du graphe des marquages accessibles, par la recherche des bonnes propriétés notamment la vivacité et l'aspect borné, par réduction, par simulation des modèles obtenus, etc.

Ainsi, dans ce chapitre, nous nous attarderons plus particulièrement sur l'analyse de l'aspect statique des spécifications SA-RT que sur l'aspect dynamique, pour lequel l'apport des réseaux de Petri semble être plus évident.

V.1 Les travaux sur la validations des spécifications SA-RT

Comme nous l'avons précisé, dans le chapitre III, la validation des spécifications SA-RT a été l'une des motivations de représentation des diagrammes SA-RT par réseaux de Petri. Elle a été effectuée soit par simulation des réseaux de Petri représentant les spécifications SA-RT, c'est le cas de [Elmstrom *et al.* 93b] par exemple, soit en exploitant les propriétés des réseaux de Petri obtenus. Dans cette seconde catégorie citons [Tse et Pong 89] et [Lee et Tan 92].

Dans [Tse et Pong 89] le formalisme défini (cf. figure *III.1* page 59) a été utilisé pour vérifier les propriétés structurelles des FDFD notamment la consistance globale, structurelle et comportementale. Ces termes sont explicités ci-dessous.

- Consistance globale : vérifier qu'une transformation de données ne réapparaît pas dans un niveau hiérarchique plus bas. L'utilisation des réseaux de Petri n'est pas déterminante pour cette vérification.
- Consistance structurelle : vérifier que les flots de données entrants et sortants d'une transformation de données apparaissent lors de la décomposition de cette transformation dans un niveau fils.
- Consistance comportementale : vérifier la conservation des règles de production de données en passant d'un niveau hiérarchique à un autre. Si, pour une transformation de données, une entrée E produit un sortie S, alors le DFD obtenu en décomposant cette transformation doit fournir la même sortie S pour la même entrée E.

Ces vérifications ont été effectuées en se basant sur la description algébrique associée aux FDFD et calquée sur les réseaux de Petri.

Dans [Lee et Tau 92] la représentation des DFDs par réseaux de Petri permet de vérifier certaines *contraintes* dans un même niveau hiérarchique et entre les niveaux hiérarchiques successifs.

A un niveau hiérarchique donné, les contraintes vérifiées sont : chaque source possède au moins un flot sortant, chaque puits possède au moins un flot rentrant, chaque stock possède au moins un flot rentrant et un flot sortant, chaque transformation de données possède au moins un flot rentrant et un flot sortant et chaque flot de données possède une origine et une destination valables.

Entre les niveaux, les contraintes vérifiées sont : chaque flot entrant dans une transformation de données mère est aussi un flot entrant pour l'une de ses transformations filles (idem pour les flots sortants) et la somme des flots entrant à une transformation de données mère est égal à la somme des flots passés par cette transformation à ses filles (idem pour les flots sortants).

Les vérifications sont réalisées en effectuant des tests sur les arcs et les sous ensembles de places et de transitions. Par exemple $\forall$ une place P_s représentant une source, $\exists$ une transition t et une place P_f représentant un flot de données telles que P_s est une place d'entrée de t et P_f est une place de sortie de t.

Dans ces propositions de validation, les auteurs essaient de s'inspirer de la structure des réseaux de Petri pour valider la structure des diagrammes de flots de données. Les différents types de validations traités s'intéressent au respect des différentes règles d'établissement des DFDs. Ce sont donc des vérifications de la consistance des spécifications. La validation est faite en effectuant des tests sur la succession des places et des transitions dans le réseau de Petri représentant le DFD. Ces validations sont utiles dans un environnement de génie logiciel où les spécifications sont saisies sous la forme de DFDs et traduites en réseaux de Petri pour assister l'utilisateur à établir les besoins. Cependant, les mêmes tests pourraient être effectués directement sur les DFDs avec pratiquement le même résultat. Ce type de validation ne tire pas profit de la panoplie de propriétés et d'outils construits autour des réseaux de Petri.

Parmi les différents outils et les différents modèles construits autour des réseaux de Petri, il nous semble plus intéressant de rechercher des éléments qui peuvent amener un aspect plus déterminant à la validation et à l'évaluation des spécifications. La validation de la consistance des DFDs est certes intéressante mais si les réseaux de Petri ne sont *"qu'une méthode de modélisation de plus"*, sans apport particulier, la quantité de travail ne se justifie pas. En revanche, si les réseaux de Petri ont été construits dans un autre but et que la validation de la consistance est un avantage qui en découle alors dans ce cas le travail mérite d'être effectué.

Dans la suite de ce chapitre, nous explorons les possibilités offertes par les réseaux de Petri en étudiant, d'abord, les techniques d'analyse des réseaux de Petri (vivacité, réseau borné, etc), puis en étudiant certains modèles construits autour des réseaux de Petri qui sont susceptibles d'amener de nouvelles perspectives de validation et d'évaluation pour les systèmes temps réel (réseau de Petri temporels, temporisés stochastiques).

V.2 Validation structurelle des DFDs

La représentation des DFDs par réseaux de Petri et l'analyse structurelle des réseaux obtenus permettent de vérifier la consistance de ces DFDs. Il s'agit ici de vérifier que les DFDs sont «correctement établis».

V.2.1 L'établissement du réseau de Petri

Les règles de représentation des DFDs par réseaux de Petri permettent de déceler certaines erreurs dans les DFDs. D'abord, la succession entre places et transitions permet de garantir, au niveau du DFD, une succession entre données et transformations de données. Par exemple, il devient facile de déceler l'existence d'un flot de données entre deux stocks de données puisque ce cas se matérialise, au niveau du réseau de Petri, par un arc entre deux places.

Ensuite, les places sources et puits d'un réseau de Petri relatif à un niveau hiérarchique donné doivent être conservées lors de l'agrégation de ce niveau. Cette conservation permet de garantir la conservation des flots de données entre les niveaux hiérarchiques successifs (consistance structurelle selon [Tse et Pong 89]).

Enfin, en distinguant les places et les transitions représentant les différents éléments des DFDs il est possible de faire des vérifications telles que celles proposées dans [Lee et Tan 92].

V.2.2 La vivacité

L'étude de la vivacité des réseaux de Petri représentant les DFDs, permet d'avoir des informations intéressantes sur la consistance des spécifications. En effet, suivant les règles de passage entre DFDs et réseaux de Petri une transition non vivante dénote nécessairement l'existence d'une transformation de données du DFD non sollicitée ou qui manque de données pour effectuer son traitement, ce qui révèle une inconsistance dans les spécifications.

V.2.3 L'aspect borné

Pour la représentation des DFDs, aboutir à un réseau de Petri non borné n'est généralement pas synonyme de mauvaises spécifications. Ceci s'explique par le fait que dans les DFDs la production de données par les sources n'est généralement pas limitée. Ce phénomène conduit à une accumulation de jetons dans le réseau de Petri obtenu sans que cela représente une inconsistance dans les diagrammes. Pour que l'étude de l'aspect borné des réseaux de Petri représentant les DFDs soit utile, il faut que les réseaux de Petri prennent en compte la représentation des mécanismes de rafraîchissement des données qui s'opèrent lorsqu'il y a des productions de données successives (cf. figure *III.8* page 68).

Cependant, si la représentation du flot de données est faite simplement par une place, le réseau de Petri obtenu sera sûrement non borné. Il est alors nécessaire que l'analyste-concepteur vérifie une à une les causes qui amènent à un réseau non borné. Il sera acceptable que certaines places soient non bornées (places associées à des flots de données en relation avec des sources de données). En revanche, d'autres places non bornées indiquerons l'existence d'un problème de spécification (ex des places représentant des stocks de données). L'examen de l'aspect borné est mieux illustré sur l'étude des exemples suivants.

Remarque: Une technique que nous utilisons pour la vérification des réseaux de Petri représentant des DFDs consiste à limiter les sources de données: il s'agit de mettre, en amont de la transition source de données, une place contenant un nombre de jetons suffisant pour vérifier le réseau (généralement un seul jeton suffit, voir exemple figure *V.4*).

V.2.4 Exemples

V.2.4.1 Un premier exemple

L'exemple suivant illustre l'étude de la vivacité et de l'aspect borné des réseaux.

Premier cas

Supposons que l'analyste oublie le flot numéro 1 dans le DFD de la figure *V.1* (hypothèse d'étude). Alors, dans le réseau de Petri représentatif de la figure *V.2*, les transitions *traiter* et *visualiser* sont non vivantes (ce qui dénote un problème de spécification). Par ailleurs, la place *demande* est non bornée mais ceci ne correspond pas à un défaut de spécification (car il s'agit d'une place représentant un flot partant d'une source de données et ce type de place est généralement non borné).

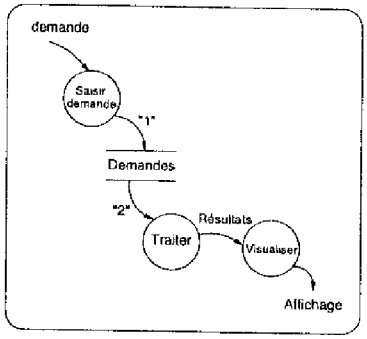


fig V.1 – Spécifications par DFD

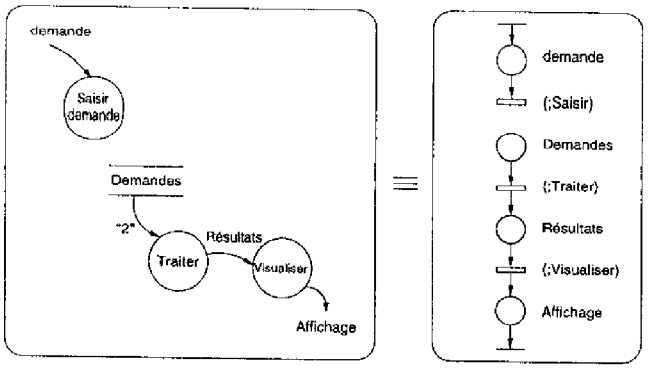


fig V.2 - Premier cas : DFD et sa représentation par réseau de Petri

Deuxième cas

Supposons que le flot numéro 2 est omis du DFD représentant les spécifications. Le réseau de Petri associé est alors celui de la figure *V.3*. Dans ce réseau, toutes les transitions sont vivantes, ce qui est a priori bon signe. Cependant, toutes les places sont non bornées. Il est difficile d'affirmer rapidement qu'il y a un problème de spécifications. Mais un examen plus détaillé s'impose. En écartant les places qui correspondent à des flots en relation avec des sources ou des puits de données, il apparaît une place puits, non bornée, représentant le stock des *demandes* ce qui dénote bien dans notre cas un problème de spécifications (des écritures dans le stock non suivies de lectures).

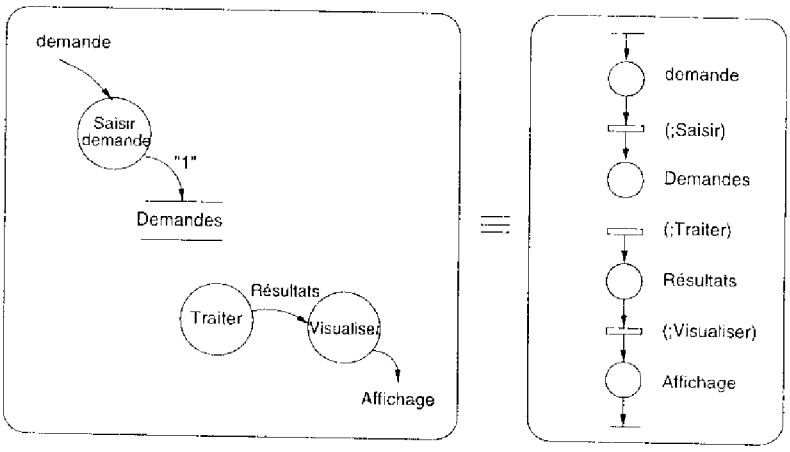


fig V.3 Deuxième cas : DFD et sa représentation par réseau de Petri

V.2.4.2 Un second exemple

Nous avons montré, lors de la représentation des DFDs par réseaux de Petri, que cette opération permet de clarifier les ambiguïtés qui existent au niveau de la consommation et la production des données par les transformations de données (cf. §III.6.2 page 69). L'association de règles de production et de consommation (ET, OU, ...) peut être source d'erreurs d'interprétation. L'analyse du réseau de Petri obtenu permet, dans certains cas, de déceler les erreurs qui peuvent être introduites et qui amènent à des blocages.

Dans l'exemple de la figure *V.4*, le DFD est consistant mais pas assez précis. Une première tentative de représentation par réseau de Petri et de clarification de la nature des opérations aboutit à un réseau non vivant. Le blocage qui s'en suit doit pousser l'analyste-concepteur à vérifier la cohérence du DFD et l'adéquation du réseau de Petri le représentant. Dans le cas présent, le blocage est dû à une ambiguïté au niveau de la transformation *afficher*. Celle-ci doit pouvoir traiter l'une ou l'autre de ses entrées (ou exclusif à l'entrée) et pas nécessairement les deux entrées en même temps. Cette précision, permet de corriger le réseau de Petri obtenu et ajoute de la précision au DFD. Elle doit être portée au niveau des mini-spécifications associées aux diverses transformations de données.

Dans cet exemple, il est intéressant de noter que nous avons limité la transition source de données (une seule donnée) pour des buts d'analyse.

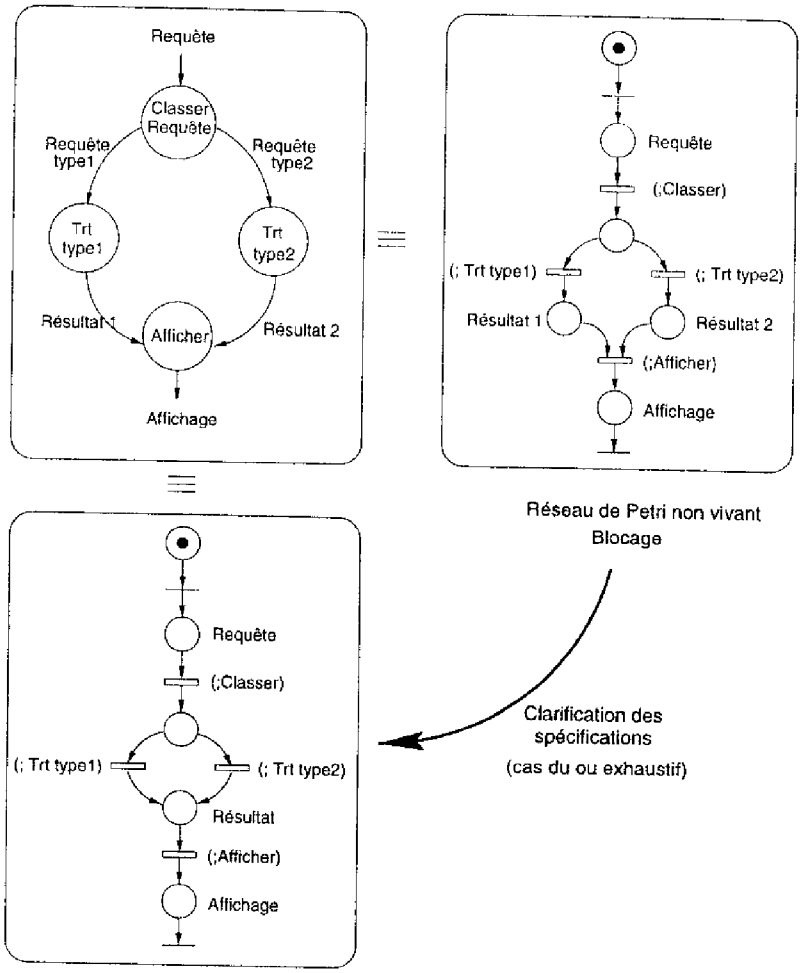


fig V.4 – Exemple de clarification des spécifications

V.2.5 Apport de la recherche des composantes conservatives

Dans le chapitre précédant, la recherche des composantes conservatives nous a amené à constater l'existence, dans certains cas, d'une composante conservative particulière (parfois la somme de plusieurs composantes) qui couvre tout le réseau de Petri représentant le DFD. Cette composante conservative traduit la conservation globale des données dans le système, c'est l'équivalent d'une équation de conservation des données.

Ce résultat est très intéressant pour la vérification de la consistance des spécifications. En effet, l'existence d'une composante conservative qui couvre tout le réseau prouve que ce réseau est borné [Murata 89] ce qui est synonyme d'une «bonne spécification».

Cependant, nous avons montré l'existence de certains cas où les DFDs étaient correctement établis sans que les réseaux de Petri les représentant puissent être couverts par une composante conservative globale. Ainsi, quand elles existent, les composantes conservatives, garantissent que le réseau de Petri qu'elles couvrent est borné. En revanche, l'inexistence de cette composante ne doit pas être perçue comme une preuve de mauvaise spécification.

Notons que la vérification de la consistance des DFDs à l'aide des composantes conservatives demande très peu d'effort puisque celles-ci sont déjà calculées dans le but du passage en objet.

V.2.6 Conclusion

La représentation des DFDs par réseaux de Petri et l'analyse des réseaux de Petri obtenus permettent de prévenir certaines erreurs dans l'établissement des DFDs. Toutefois, la vérification des bonnes propriétés des réseaux de Petri obtenus ne peut pas garantir que les DFDs qu'ils représentent sont exempts de défaut. Par ailleurs, la vérification structurelle des réseaux de Petri permet d'éviter certains problèmes mais ne peut pas déceler ceux résultant de l'interaction de ces réseaux ou de leurs interprétations.

Remarquons que les possibilités de validation structurelles amenées par l'utilisation des réseaux de Petri ne sont pas déterminantes. Cependant, comme les réseaux de Petri représentant ces DFDs ont été établis dans un autre but (la recherche des objets), il est alors intéressant de les utiliser pour des objectifs de validation.

V.3 Étude d'un niveau de spécification à l'aide du graphe des marquages accessibles

L'interprétation (conditions de tir, contraintes temporelles, etc) associée à un réseau de Petri conditionne l'évolution de ce dernier. En effet, l'interprétation peut interdire des évolutions qui étaient permises par le réseau de Petri sous-jacent. Ainsi, l'analyse d'un réseau de Petri peut conclure quant à la non validité d'un modèle mais ne peut pas garantir sa validité lorsque son évolution dépend de paramètres externes.

Une bonne technique d'analyse d'un niveau SA-RT donné consiste à établir le graphe des états accessibles des réseaux de Petri représentant ce niveau. C'est un moyen de tenir compte des interprétations associées aux réseaux de Petri et des interactions entre les parties statiques et dynamiques (spécifiées sous forme d'un couple *(condition;action)* au niveau des transitions). Ce graphe permet de s'assurer de la consistance des spécifications et de vérifier l'éventualité de certains blocages qui peuvent provenir d'une incohérence entre les parties statiques et dynamiques d'un même niveau. Il constitue une sorte de simulation exhaustive des spécifications, mais cette exhaustivité est assez chère payée puisque le graphe ne peut être établi que de façon manuelle.

Une objection généralement soulevée contre l'établissement d'un tel graphe est le

risque d'une explosion combinatoire pour les systèmes de grande taille. Pour l'étude des spécifications SA-RT, ce risque est limité puisque les niveaux sont étudiés un à un séparément et la taille des diagrammes, à un niveau donné, est limitée si l'analyste a respecté la règle de Miller [Miller 56] (les diagrammes ne doivent comprendre que sept, plus ou moins deux, éléments).

V.3.1 Exemple

Il s'agit d'une partie des spécifications d'un système de facturation. La figure *V.5* montre le DFD et le réseau de Petri spécifiant le contrôle du niveau considéré. Les transformations de données du DFD sont activées à partir de ce réseau de Petri par l'interprétation associée aux transitions. La figure *V.6* montre la description par réseaux de Petri des parties statique et dynamique. L'évolution du réseau de Petri représentant le DFD est conditionnée par l'évolution du réseau de Petri de contrôle. L'évolution de ce dernier est aussi conditionnée par les informations fournies par le premier. Cette double interaction risque de provoquer des blocages.

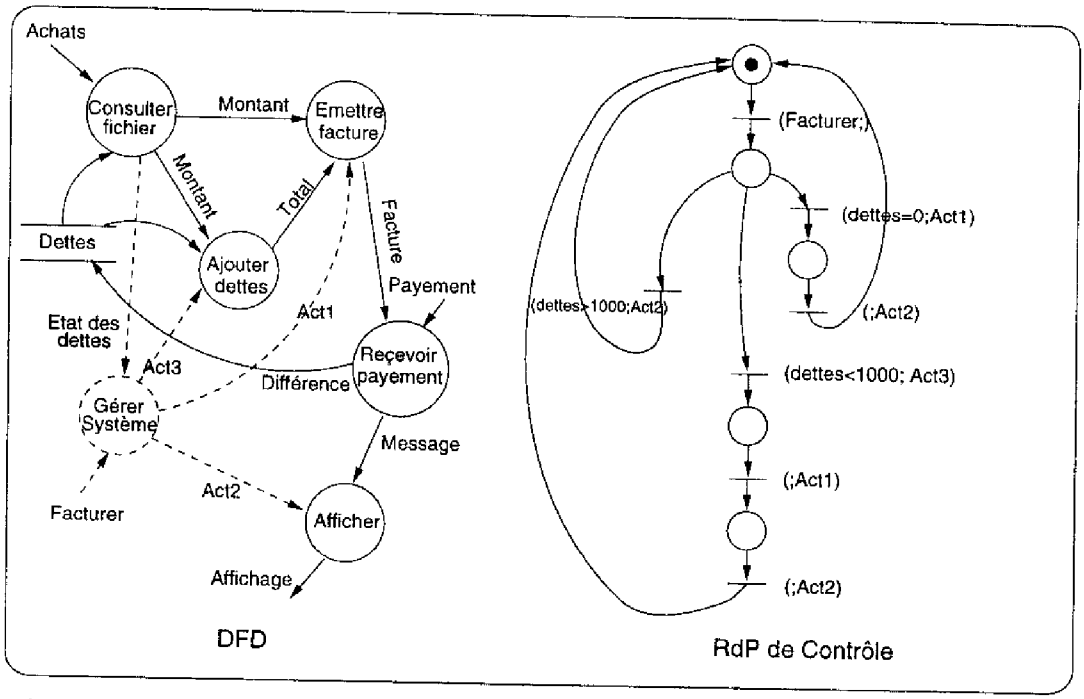


fig V.5 – Spécifications d'un système de facturation

Dans cet exemple, chaque réseau de Petri étudié à part est vivant et borné (pour un nombre fini de données en entrée). Cependant, en établissant le graphe des marquages accessibles qui tient compte des interactions entre les deux réseaux il apparaît un état de blocage: le tir de la transition t_2 fait passer le système de l'état $[P_3P_8P_{11}P_{21}]$ vers

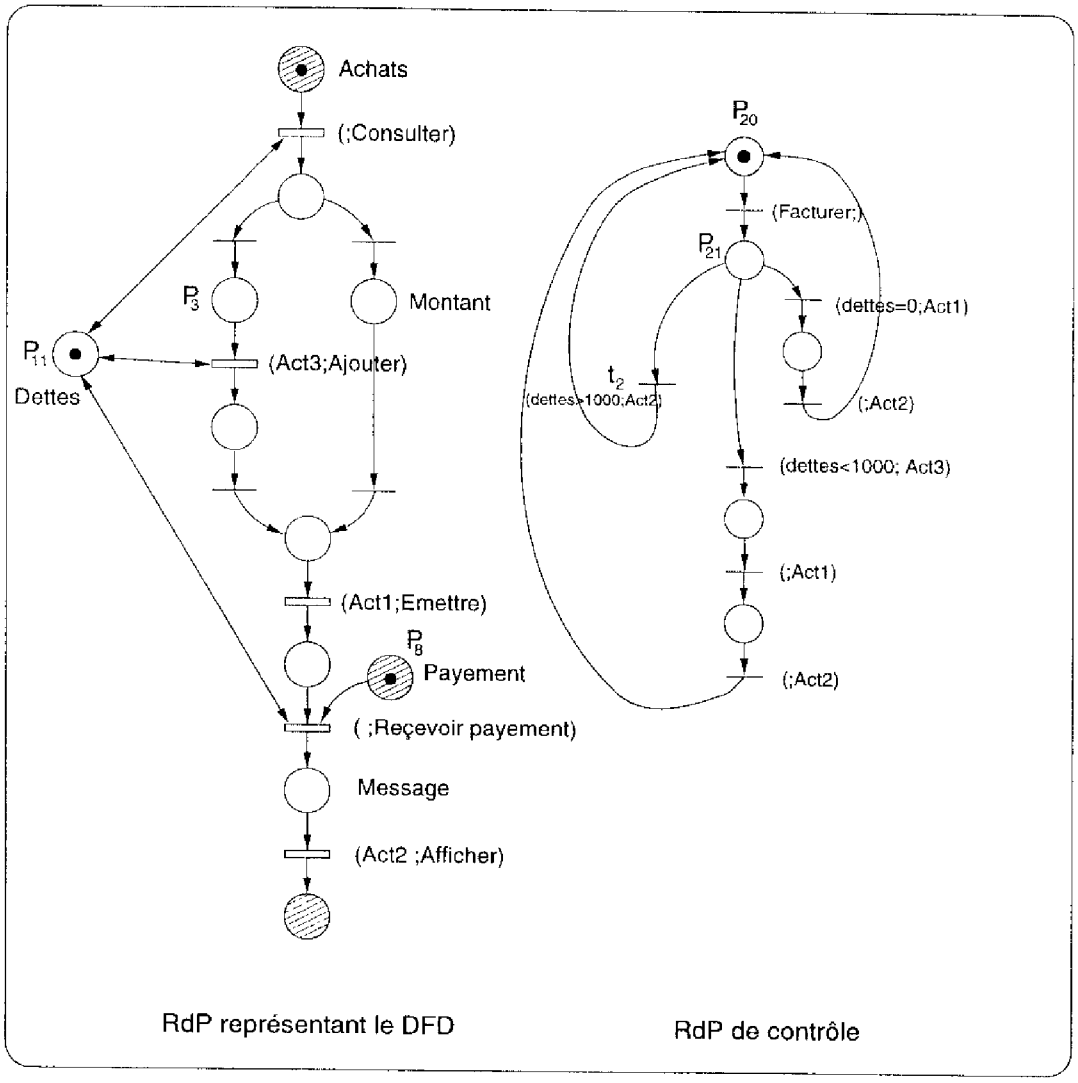


fig V.6 – Description par réseaux de Petri des spécifications

l'état $[P_3 P_8 P_{11} P_{20}]$ qui est un état de blocage. Ce problème vient du fait que dans le DFD le cas (dettes > 1000) n'a pas été prévu. Ce problème n'aurait pas pu être décelé par une analyse séparée des deux réseaux de Petri.

Dans la suite du chapitre, nous examinons les possibilités de validation et d'évaluations offertes par les modèles basés sur les réseaux de Petri qui sont, a priori, intéressants pour les systèmes temps réel. Il s'agit des modèles réseaux de Petri temporels et réseaux de Petri temporisés stochastiques.

V.4 Évaluations par les réseaux de Petri temporels

V.4.1 Motivations

De part leurs variantes temporelles, les réseaux de Petri peuvent amener plusieurs possibilités de représentation, de validation et d'évaluation des spécifications SA-RT. En effet, SA-RT utilisée comme méthode de spécification lors de l'étape d'analyse des besoins pour les systèmes temps réel, n'intègre dans ses diagrammes que les aspects fonctionnels et les spécifications temporelles implicites (séquences d'opérations, synchronisations, parallélisme). La représentation des spécifications temporelles explicites est généralement portée sur des tableaux à part. Par spécifications temporelles explicites, nous entendons des spécifications sur la durée explicite de l'exécution d'une tâche, l'intervalle de temps maximum tolérable qui sépare un événement et l'action correspondante (ou plus généralement deux événements) ou encore la fréquence d'arrivée des données (et si elles doivent toutes être prises en compte), etc.

Plusieurs travaux ont été entrepris pour compléter la méthode SA-RT par une spécification formelle des contraintes temporelles et par des outils de validation de ces contraintes. Citons notamment les travaux de [Delfieu et Sahraoui 94a], [Delfieu et Sahraoui 94b]. Dans le cadre de notre travail, nous nous intéressons particulièrement aux réseaux de Petri et à l'examen de leur apport pour l'expression et l'évaluation des contraintes temporelles.

En adoptant une représentation par réseaux de Petri des spécifications SA-RT, et en utilisant le modèle réseau de Petri temporel [Merlin et Farber 76], il est possible d'intégrer sur le même support les spécifications fonctionnelles et temporelles implicites et explicites ce qui n'était pas envisageable (ou du moins non envisagé) avec l'utilisation classique de la méthode SA-RT.

Remarquons que le modèle obtenu n'est pas un modèle qui va remplacer les spécifications SA-RT ni un modèle à présenter à l'utilisateur en guise de spécifications. C'est un modèle qui servira essentiellement à effectuer des validations et des évaluations des spécifications SA-RT. Par ailleurs, il faut aussi rappeler qu'en adoptant ce modèle temporel, l'hypothèse *"le temps d'exécution des transformations de données est supposé nul"* (formulée lors de l'établissement du modèle réseau de Petri pour la description des spécifications) est abandonnée. En fait, cette hypothèse était intéressante pour étudier la circulation des données.

V.4.2 Le modèle

Parmi les modèles de réseaux de Petri qui intègrent le temps, il existe des modèles temporisés qui intègrent le temps au niveau des délais associés aux places [Sifakis 77] ou aux transitions [Ramchandani 74] et des modèles temporels qui intègrent le temps au niveau des intervalles de tir associés aux transitions [Merlin et Farber 76]. Ce dernier type est plus général et semble être le plus intéressant pour notre représentation : les événements et les activités sont associés aux transitions et les durées peuvent varier

entre une borne minimale et une borne maximale.

Dans les réseaux de Petri temporels [Merlin et Farber 76], à chaque transition est associé un intervalle de tir $[a_i \quad b_i]$ où a_i est l'instant de tir au plus tôt de la transition et b_i est son instant de tir au plus tard (figure V.7). Ainsi, pour être tirée, une transition doit rester sensibilisée pendant au moins a_i unités de temps, elle doit être tirée, au plus tard, au bout de b_i unités de temps de sensibilisation.

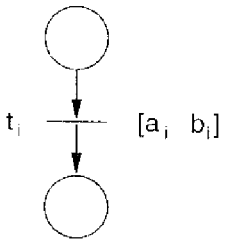


fig V.7 - Réseau de Petri temporel

Les évaluations faites à partir des réseaux de Petri temporels sont essentiellement basées sur l'établissement du graphe de classes d'états [Berthomieu et Menasche 82], [Merlin et Farber 76]. A partir de ce graphe, il est possible de calculer des durées minimales ou maximales qui séparent le tir de deux transitions particulières.

V.4.3 Application aux spécifications SA-RT

A part les délais et les temporisations imposés par les besoins de l'application, dans le cas de la représentation des DFDs, les réseaux de Petri temporels peuvent représenter deux aspects :

- La prise en compte du temps mis par une transformation de données pour accomplir sa tâche. Un intervalle de temps $[t_1 \quad t_2]$ au niveau d'une transition indique que la transformation de données représentée par cette transition met au moins t_1 unités de temps pour accomplir sa tâche et se termine au plus tard après t_2 unités de temps. Cette transition peut être vue comme l'agrégation d'une séquence [*début de tâche, exécution, fin de tâche*] dans laquelle l'intervalle de temps est associé à la transition de fin de tâche (voir figure V.8).

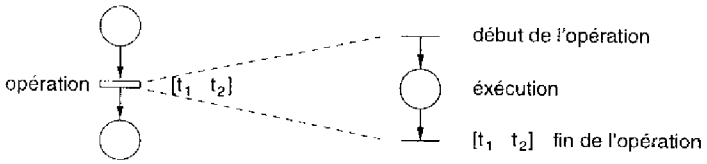


fig V.8 Agrégation d'une transition

- La prise en compte de la fréquence d'arrivée des données à partir des sources. Un intervalle de temps $[t_1 \quad t_2]$ au niveau d'une transition source indique qu'une nouvelle donnée est disponible au moins toutes les t_1 unités de temps et, au plus tard, toutes les t_2 unités de temps.

Pour l'aspect dynamique des spécifications, comme pour tout système à événements discrets, les réseaux de Petri temporels peuvent représenter les chiens de garde et les intervalles de temps pendant lesquels les événements sont attendus.

Proposition

A part les aspect temporels classiques, la description des spécifications SA-RT par les réseaux de Petri *temporels* permet de représenter deux aspects propres aux DFDs : les durées d'exécution des transformations de données et les fréquences d'arrivée des données. Ainsi, Le modèle obtenu intègre, sur le même réseau, *les spécifications fonctionnelles et temporelles implicites et explicites.*

■

L'analyse du réseau de Petri obtenu peut se faire avec le graphe de classes d'états qui prend en compte tous les instants de tir possibles [Berthomieu et Menasche 82]. Il est également possible de choisir des instants de tir particuliers pour établir les graphes de marquages pour des comportements particuliers. Les instants de tir intéressants sont le tir au plus tôt (bornes minimales, par exemple arrivée des données avec une fréquence maximale) ou alors le tir au plus tard (bornes maximales, par exemple exécution des transformations le plus lentement possible). Dans ce cas un graphe de marquage est établi pour un comportement précis dans le but d'examiner les pires cas pour la spécification.

Généralement, l'établissement d'un graphe des marquages amène, dans le cas des systèmes complexes, à une explosion combinatoire. Pour le cas des spécifications SA-RT, cet inconvénient peut être évité en analysant les spécifications SA-RT niveau par niveau. Toutefois, s'il est indispensable de remettre les spécifications à plat, le problème d'explosion combinatoire se repose.

V.4.4 Exploitation du modèle obtenu

Le modèle obtenu peut être exploité de deux manières différentes selon l'utilisation faite des spécifications SA-RT.

En tant que méthode de spécification des besoins, SA-RT doit simplement permettre de *spécifier ce que doit faire le système*. Dans ce cas, le modèle réseau de Petri temporel obtenu permet d'intégrer sur un même support **les spécifications fonctionnelles et temporelles implicites et explicites.** L'analyse de ce modèle à l'aide du graphe de classes d'états permet de vérifier la cohérence des spécifications (et plus particulièrement celle des spécifications temporelles, cf. exemple suivant).

Sur un autre plan, les diagrammes SA-RT sont également utilisés pour passer à la conception, l'aspect temporel est rajouté, sur le modèle réseau de Petri décrivant les spécifications, en fonction des performances espérées du système. En effet, en fonction du matériel existant, de l'architecture choisie et du modèle fonctionnel établi, il est possible d'associer des durées d'exécution aux différentes opérations. L'analyse de ce modèle, à l'aide du graphe de classes d'états, permet de faire des évaluations temporelles notamment sur des durées d'exécution de cycles, de tâches ou d'ensemble d'opérations. Il est alors possible d'utiliser ces évaluations pour comparer les performances temporelles du système conçu avec les exigences des spécifications ce qui permet éventuellement de justifier des choix d'architecture ou d'implémentation ou même de remettre en cause les spécifications fonctionnelles.

Proposition

Le passage SA-RT - réseaux de Petri et l'utilisation du modèle RdP temporel permettent de *valider la cohérence des spécifications* ou d'évaluer *les performances temporelles* du système à concevoir.

■

Dans les paragraphes suivants, nous illustrons, sur deux exemples, les possibilités d'utilisation de SA-RT avec les réseaux de Petri temporels pour l'évaluation des performances.

V.4.5 Exemple de vérification de la cohérence des spécifications

Dans cet exemple (présenté dans [Benzina *et al.* 97]), il s'agit de l'étude d'un système de recherche de trajectoire pour équiper des véhicules dits *intelligents*. Dès qu'il est activé, le système détermine la position courante du véhicule qu'il équipe, recherche le meilleur chemin qui mène au point destination et réactualise l'affichage de la position et du chemin sur l'écran. La localisation de la position du véhicule se fait grâce à un ensemble de balises réparties sur les routes concernées. Ces balises émettent un signal toutes les $3ms$. La recherche du meilleur chemin se fait en consultant une base de données distante commune à tous les véhicules d'une zone. Chaque base de données couvre une zone qui peut contenir jusqu'à 4 véhicules. La consultation d'une base de données dure $2ms$ et si plusieurs requêtes sont émises au même moment par plusieurs véhicules, elles sont mises en file d'attente.

Le donneur d'ordre exige que le système puisse répondre à une requête en moins de $10ms$.

La représentation de ces spécifications par un DFD (niveau d'abstraction élevé) est montré dans la partie gauche de la figure *V.9*. Le réseau de Petri représentant ce DFD avec les spécifications temporelles du cahier de charges est porté sur la partie droite de la même figure.

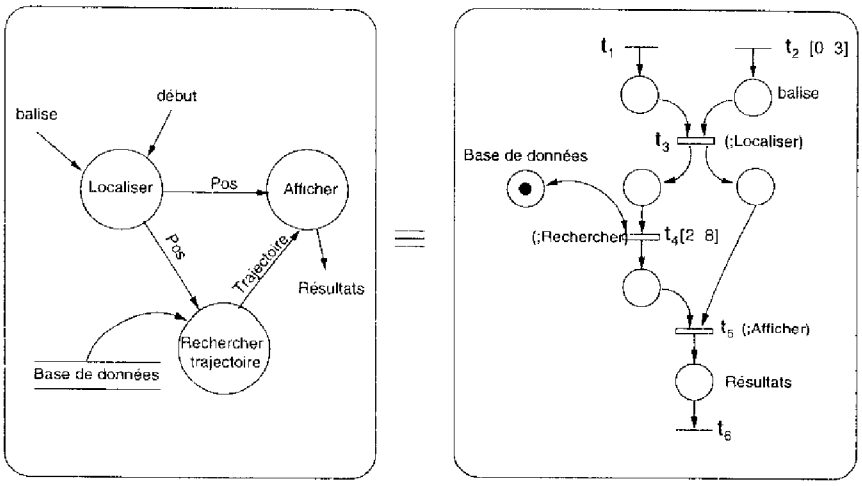


fig V.9 - Spécifications d'un système de localisation et sa représentation par RdP

Dans cet exemple, nous nous intéressons au temps mis par le système pour répondre à une requête de recherche. Il s'agit du temps qui sépare le tir des transitions t_1 et t_5. L'étude du réseau de Petri temporel obtenu montre que cette durée peut atteindre $11ms$ en prenant en compte uniquement les attentes dues aux différents composants mis en jeux dans le système. Une conclusion assez évidente est que les spécifications temporelles ne sont pas cohérentes : *il est impossible de réaliser les objectifs fixés avec les moyens fournis.*

L'analyste-concepteur peut formuler au donneur d'ordres du système des propositions dans le sens du réajustement des spécifications. Il est possible par exemple de relâcher légèrement la contrainte du temps de réponse à une requête, ou alors d'équiper les véhicules de bases de données individuelles et penser à un système de mise à jour de ces bases de données.

V.4.6 Exemple de choix d'architecture et d'implémentation basés sur des évaluations temporelles

Dans cet exemple, il s'agit de la réalisation d'un système pour l'acquisition, le traitement et la visualisation d'un signal périodique. Il faut concevoir un logiciel qui boucle, implanté sur un système informatique muni d'un écran et d'un boîtier d'acquisition. Ce système doit permettre d'acquérir des séries de N échantillons du signal, d'effectuer les traitements nécessaires sur cette série (élimination du bruit, mise en forme du signal), puis de visualiser le résultat.

Une première idée serait d'effectuer le travail selon le DFD de la figure *V.10*. Le réseau de Petri représentant ce DFD est assez facile à élaborer (même figure). A partir du DFD obtenu (ou même des spécifications SA-RT) il est difficile de faire des évaluations quant aux durées d'exécution et des techniques d'implémentation. En ajoutant des attributs temporels au réseau de Petri obtenu, il est possible de faire des évaluations. Dans ce cas, les attributs temporels consistent essentiellement en une estimation de la durée de réalisation de chacune des transformations de données, donc des durées associées aux transitions. Il est également possible de représenter la fréquence d'arrivée des données à partir des sources de données.

Pour cet exemple, les spécifications temporelles sont :

- Les échantillons sont relevés toutes les $10ms$.
- L'acquisition d'un échantillon dure $1ms$.
- Le traitement d'une série dure $100ms$.
- La visualisation du résultat dure $2ms$.

Le réseau de la figure *V.11* est obtenu en rajoutant les spécifications temporelles sur le réseau de Petri représentant la circulation des données. Dans ce cas, chaque transition (transformations de données) a une durée fixe et connue.

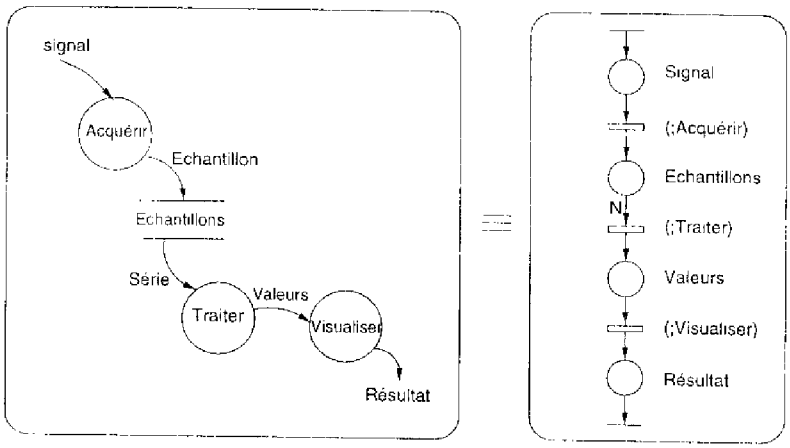


fig V.10 Système d'acquisition d'échantillons

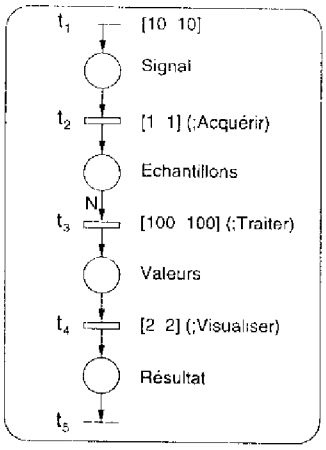


fig V.11 - Description par réseau de Petri temporel

Avant de concevoir en détail le logiciel, il est possible de faire une évaluation rapide du système. Il existe deux façons d'implémenter les spécifications du DFD présenté plus haut :

1. Le système effectue l'acquisition de tous les échantillons d'une série puis effectue le traitement et la visualisation. La durée d'une boucle peut être obtenue à partir du graphe des marquages temporisé en calculant l'intervalle de temps qui sépare le premier tir de la transition t1 et celui de la transition t5. La durée d'une boucle (pour une série de N échantillons) est donc de $(11N + 102)ms$. Par exemple, pour des séries de 10 échantillons, la durée d'une boucle est de $202ms$. Cependant, durant une boucle il existe un temps d'inactivité de $100ms$ (attente des échantillons). Il est donc plus intéressant d'exploiter ce temps libre, ce qui fait penser à la deuxième solution.

2. Tel qu'il est décrit, le réseau de Petri (comme le DFD) n'interdit pas de faire

tourner les tâches acquisition et traitement en parallèle (si toutefois il y a suffisamment de jetons dans le stock). Ce qui laisse penser qu'il est possible de continuer l'acquisition des échantillons de la série $i+1$ tout en effectuant le traitement et la visualisation de la série i. Dans ce cas, la durée d'une boucle (acquisition + traitement) est de $\max(11N, 102)ms$, soit $110ms$ pour une série de 10 échantillons.

Remarquons qu'en phase d'analyse des besoins, la spécification du système ne se préoccupe pas de la méthode à adopter pour le concevoir et l'implémenter (et généralement elle n'a pas à le faire sauf si c'est une des conditions du cahier des charges). Cependant, l'évaluation des durées des traitements qui a été faite plus haut et qui intervient après l'étape d'analyse des besoins va influencer la conception architecturale et l'implémentation du système. En effet, la parallélisation de *l'acquisition* et du *traitement* peut être implémentée sur deux processeurs différents (avec gestion de l'accès à la mémoire partagée -le stock-, gestion de la communication entre les deux processeurs, etc). Il est également possible de l'implémenter sur un même processeur et dans ce cas, il faut activer l'acquisition par des interruptions toutes les $10ms$ alors que le système effectue le traitement de la série précédente. Il devient alors nécessaire de créer un nouveau stock de données pour différencier l'ancienne série de la nouvelle (voir figure *V.12*).

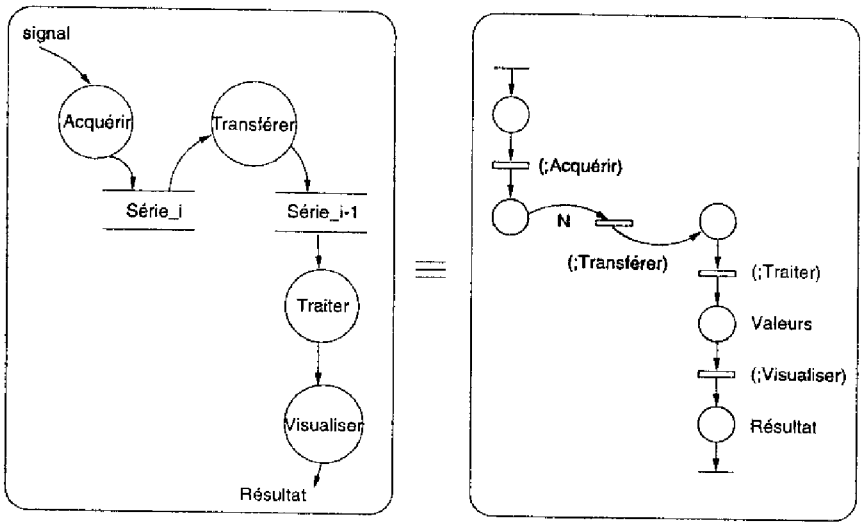


fig V.12 - Modification des spécifications

Ainsi, l'utilisation des réseaux de Petri temporels a permis de faire une évaluation rapide des spécifications et de justifier un choix de conception architecturale et d'implémentation. La détermination des intervalles de temps aurait été difficile à faire sans passer par la traduction du modèle SA-RT vers le modèle réseau de Petri temporel et l'utilisation du graphe des marquages temporisé.

V.5 Évaluations à l'aide des réseaux de Petri temporisés stochastiques

V.5.1 Motivations

Les réseaux de Petri temporels ont permis d'intégrer sur le même support les spécifications fonctionnelles et temporelles implicites et explicites. Grâce aux réseaux de Petri temporisés stochastiques, il est possible de rajouter à cette description une composante stochastique pour décrire une incertitude sur la date d'arrivée d'une donnée, la date d'exécution d'une tâche ou le temps mis pour son exécution.

Remarque: Les réseaux de Petri temporisés stochastiques se suffisent à eux mêmes pour modéliser les systèmes et effectuer des évaluations de performances. Si tel est le but de l'ingénieur, le passage SA-RT→RdPTS n'est qu'une complication inutile. Cependant, le modèle RdPTS est d'un apport certain, lorsque des spécifications SA-RT sont déjà disponibles et qu'une évaluation de ces spécifications plutôt qu'une évaluation du système doit être effectuée (toutefois les deux ne sont pas antagonistes).

V.5.2 Le modèle

La modélisation des systèmes par réseaux de Petri temporisés stochastiques (figure *V.13*) permet d'associer à chaque transition t_i :

- Un intervalle de tir $[a_i \quad b_i]$ où a_i est l'instant de tir au plus tôt et b_i est l'instant de tir au plus tard de la transition t_i.
- Une fonction $f(x)$ distribution de la densité de probabilité de tir de la transition t_i sur l'intervalle $[a_i \quad b_i]$ telle que :

$$\begin{cases} f(x) = 0 & si\ x \notin [a_i \quad b_i] \\ f(x) \geq 0 & si\ x \in [a_i \quad b_i] \\ \int_{a_i}^{b_i} f(x)dx = 1 \end{cases}$$

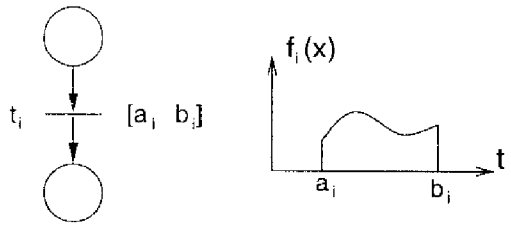


fig V.13 - RdPTS: Réseau de Petri Temporisé Stochastique

Plus particulièrement, tout en restant réaliste et en gardant le niveau de difficulté des calculs raisonnable, il est possible de travailler avec trois types de distributions sur l'intervalle $[a_i \quad b_i]$:

- une distribution uniforme :

$$f(x) = \frac{1}{b_i - a_i} \qquad pour \quad x \in [a_i \quad b_i]$$

- une distribution exponentielle (sans mémoire temporelle, voir [Atamna 94]) :

$$f(x) = \lambda e^{-\lambda x} \qquad pour \quad x \in [a_i \quad b_i]$$

- une distribution impulsionelle (dirac) :

$$f(x) = \delta(x - t) \quad pour \quad a_i = b_i = t$$

- ou une composition de ces distributions.

Ce modèle permet de faire des évaluations quantitatives et qualitatives des systèmes distribués. Il est principalement utilisé dans la modélisation et l'évaluation des protocoles de communication [Atamna 94] [Juanole et Gallon 96].

L'analyse du modèle se fait à l'aide du graphe d'états probabilisé (voir les références précédentes). Les nœuds de ce graphe représentent les états du système. Un état est caractérisé par le marquage du réseau de Petri, l'ensemble des transitions sensibilisées t_i, l'intervalle de tir associé à chacune de ces transitions $[a_i \quad b_i]$ et la distribution $f_i(x)$ de la probabilité de tir de chaque transition t_i sur l'intervalle $[a_i \quad b_i]$. Le passage d'un état à un autre est caractérisé par la transition tirée t_i, la probabilité p_i de tir de cette transition et l'instant moyen de tir θ_{moy}.

La procédure permettant de calculer ces valeurs est la suivante :

- Calculer le DIT (Dernier Instant de Tir), c'est l'instant max avant lequel l'une des transitions sensibilisées doit être tirée : c'est donc la plus petite des bornes maximales des intervalles de tir des transitions sensibilisées.

- Calculer p_i, la probabilité de tirer la transition t_i avant le DIT. Elle est donnée par la relation :

$$p_i = P(\theta_i < DIT) = \int_{a_i}^{DIT} f_i(x) \left(\prod_{j \neq i} \int_x^{b_j} f_j(y) dy \right) dx \qquad j/\ t_j\ est\ sensibilisée$$

  C'est la probabilité de tirer t_i avant le DIT ($\theta_i < DIT$) sachant que toutes les autres transitions sensibilisées t_j doivent être tirées après le tir de t_i.

– Calculer θ_{moy}, l'instant moyen de tir donné par la relation :

$$\theta_{moy} = \frac{1}{p_i} \int_{a_i}^{DIT} x f_i(x) \left(\prod_{j \neq i} \int_{x}^{b_j} f_j(y) dy \right) dx \qquad j/\ t_j\ est\ sensibilisée$$

Au passage dans un nouvel état du graphe d'états probabilisé, il faut revoir la liste des transitions sensibilisées et remettre à jour les intervalles de tir et les distributions des probabilités de tir associées à ces transitions (notion de mémoire temporelle).

D'autre types de graphes d'états peuvent être établis avec différents instants de tir possibles comme le tir au plus tôt ($\theta = \theta_{MIN} = a_i$), ou le tir au plus tard ($\theta = \theta_{MAX} = DIT$) [Juanole et Gallon 96]. La possibilité d'établir un graphe d'état probabilisé exhaustif (qui prend en compte tous les instants de tirs possibles) est en cours d'étude par les concepteurs de ce modèle.

Des outils logiciels ont été développés pour analyser les RdPTS. Signalons particulièrement le logiciel STPN [Atamna 93], qui permet de calculer le graphe d'états probabilisé (jusqu'à 5000 états) et d'effectuer des évaluations de performances à partir de ce graphe. Il permet de calculer les probabilités stationnaires, les temps de récurrences, de faire des projections sur des états, etc.

V.5.3 Application aux spécifications SA-RT

Comme pour les réseaux de Petri temporels, lors de la représentation des DFDs par réseaux de Petri, les intervalles de temps permettent de représenter deux aspects relatifs aux DFDs. A part les délais et les temporisations imposés par les besoins de l'application, il est possible de quantifier le temps mis par une transformation pour effectuer sa tâche, et de représenter la fréquence d'arrivée des données.

Par ailleurs, l'aspect stochastique permet de représenter la distribution de la probabilité d'occurrence de la date de fin d'exécution d'une transformation ou de la date d'arrivée d'une donnée pendant l'intervalle de temps spécifié.

Les aspects temporels et stochastiques peuvent être déduits des spécifications ou alors introduits par l'analyste-concepteur en fonction du matériel disponible, de l'architecture choisie et des différentes techniques d'implémentation.

V.5.4 Exploitation du modèle obtenu

Le modèle RdPTS obtenu à partir des spécifications SA-RT peut être exploité de deux façons différentes.

Une première application consiste à faire une évaluation de la cohérence des spécifications. Dans ce cas, il faut porter sur le modèle RdPTS les spécifications temporelles et stochastiques exprimant **les besoins**. Il est alors possible de donner à l'utilisateur des indications sur les probabilités pour que le système, tel qu'il a été spécifié, donne

des comportements non cohérents ou des performances peu désirables (cf. exemple suivant).

Une seconde possibilité consiste à utiliser la description par RdPTS comme un outil d'aide au passage à la conception et à l'implémentation en évaluant les choix architecturaux possibles et les techniques d'implémentation possibles. Comme pour les réseaux de Petri temporels, les RdPTS permettent aussi de déterminer dans quelles mesures la conception choisie rejoint les spécifications du cahier des charges et ils apportent en plus une appréciation qualitative de cette conformité.

Proposition

Relativement aux réseaux de Petri temporels l'utilisation des RdPTS pour la représentation des spécifications SA-RT permet d'ajouter une composante stochastique quantitative à l'évaluation des spécifications. ■

V.5.5 Exemple

Reprenons l'exemple du paragraphe §V.4.5. En utilisant les réseaux de Petri temporels nous avons vu que les spécifications temporelles étaient incohérentes (le temps de réponse pouvait dépasser $10ms$). En introduisant une composante stochastique, il est possible de donner à l'utilisateur une réponse plus intéressante : *qu'elle est la probabilité pour que le temps de réponse dépasse 10ms.* En fonction de cette probabilité et en fonction de la rigidité de cette contrainte, l'utilisateur peut choisir de revoir ses spécifications ou de courir un risque «calculé».

Dans cet exemple il est possible d'introduire deux composantes stochastiques :

- Le signal *balise* peut arriver dans un intervalle de $[0 \quad 3]ms$ avec une distribution de probabilité uniforme (de valeur 1/3).
- La consultation de la base de données peut durer de 2 à $8ms$ avec une distribution de probabilité uniforme (de valeur 1/6). Il est possible de considérer d'autres distributions, par exemple des diracs de valeurs 1/4 à 2, 4, 6 et $8ms$.
- Les autres transitions sont tirées dès qu'elles sont sensibilisées (il faut leur associer des diracs à l'instant 0).

L'évaluation du réseau de Petri obtenu se fait par l'établissement du graphe d'états probabilisé. Dans ce cas c'est un graphe séquentiel simple (figure *V.14*). La réponse à la question «*quelle est la probabilité pour que le temps de réponse dépasse* $10ms$» peut être obtenue à partir de ce graphe. En effet, le temps de réponse est le temps T mis entre le tir de la transition t_1 (début du traitement) et la transition t_5 (affichage du résultat). L'analyse du graphe montre que T peut varier entre $2ms$ et $11ms$ avec une valeur moyenne de $6.5ms$. La probabilité pour que T dépasse les $10ms$ est :

$$P[T > 10] = P[(\theta_4 + \theta_2) > 10] = \int_0^3 f_2(\theta_2)(1 - F_4(10 - \theta_2))d\theta_2 = \frac{1}{36}.$$

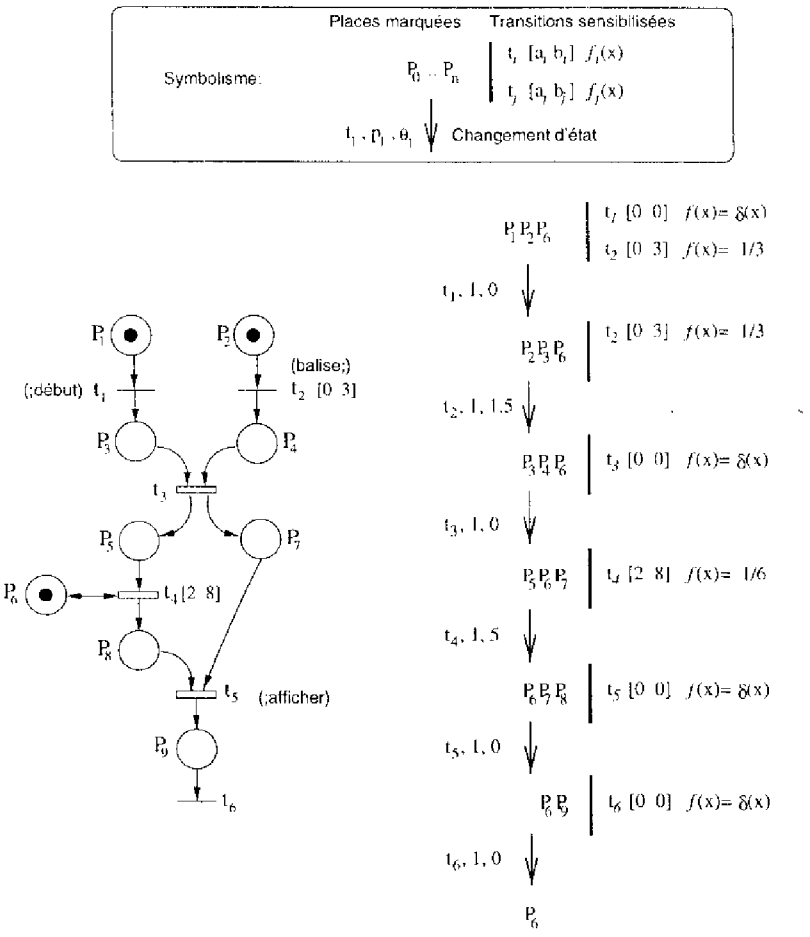


fig V.14 Graphe d'états temporisé stochastique

Cette probabilité aurait pu être calculée en armant une temporisation de $10ms$ après le tir de t_1 (transition t_0 de la figure *V.15*) et en relevant directement sur le graphe de classes d'états probabilisé exhaustif (graphe de droite de la figure *V.15*) la probabilité de passer à un état indésirable qui correspond à la fin de la temporisation (tir de t_0) avant la fin du traitement (tir de t_5). Cet état indésirable est un état ou la place P_0 n'est pas marquée. En effet, tant que la place P_0 est marquée le temps maximal ($10ms$) n'est pas écoulé. Quand la temporisation est finie, le tir de t_0 amène le réseau de Petri au marquage $[P_5 P_6 P_7]$ (c'est l'état indésirable) avec une probabilité de $1/36$.

Ainsi, l'évaluation des spécifications avec les réseaux de Petri temporisés stochastiques permet de donner une information quantitative supplémentaire sur la probabilité de l'occurrence d'une incohérence. Dans ce cas, le système risque une fois sur 36 de ne pas répondre dans les délais imposés. En fonction de cette information, c'est au donneur d'ordres à estimer si le risque encouru justifie des investissements dans des moyens plus performants.

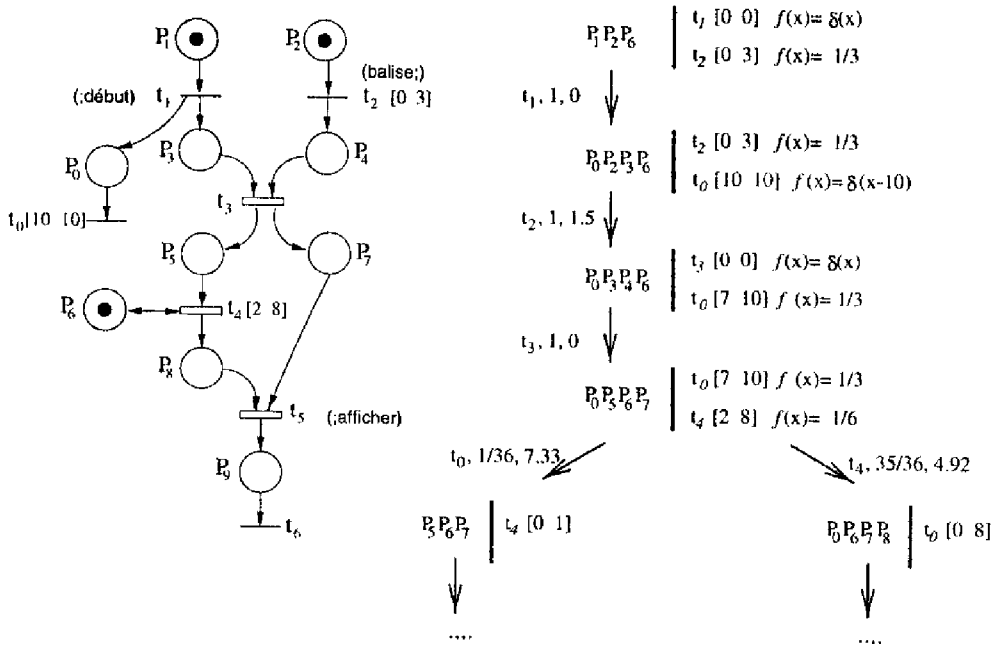


fig V.15 – Étude du RdPTS par graphe d'états probabilisé

V.6 Conclusion

La description des spécifications SA-RT par réseaux de Petri a déjà été entreprise pour des buts de validation des spécifications. Les travaux publiés s'appuient généralement sur le formalisme structurel des réseaux de Petri pour vérifier l'aspect structurel des spécifications (généralement des DFDs).

Dans ce document, nous avons mis en évidence l'apport des réseaux de Petri pour la validation structurelle des spécifications. Par ailleurs, nous avons montré que si le seul objectif est la validation structurelle, la représentation des spécifications par des réseaux de Petri ne constitue pas une solution rentable. En effet, les incohérences relevées à l'aide des réseaux de Petri peuvent être décelées plus simplement sur les DFDs.

En revanche, nous montrons que les réseaux de Petri apportent des possibilités appréciables grâce aux modèles complémentaires. Ainsi, nous avons montré que la description des spécifications par réseaux de Petri temporels permet d'intégrer, sur un support unique, les spécifications fonctionnelles et temporelles implicites et explicites, ce qui rend possible l'analyse des spécifications en prenant en compte leurs différents aspects.

En outre, les réseaux de Petri temporels permettent de faire des évaluations temporelles explicites (comme les calculs de durées d'exécution de tâches, les limites minimales et maximales d'intervalles de temps séparant deux événements) ce qui est très intéressant dans le cas des systèmes temps réel. Ces évaluations sont d'autant plus utiles qu'elles facilitent le passage entre les étapes d'analyse des besoins et de

conception. Elles permettent de vérifier l'adéquation du résultat obtenu vis-à-vis des spécifications et de justifier les choix architecturaux et les techniques d'implémentation pour la conception des systèmes temps réel. Par ailleurs, l'utilisation des réseaux de Petri temporisés stochastiques permet d'ajouter aux étapes d'analyse des besoins et de conception un aspect stochastique quantitatif, utile pour évaluer les probabilités de trouver les systèmes étudiés dans des états particuliers ou de suivre des évolutions particulières.

Conclusions et Perspectives

La recherche sur les méthodologies d'analyse et de conception des systèmes temps réel est un domaine où les concepts ont rapidement évolué ces dernières années. Cette évolution pose de nombreux problèmes de transition pour les utilisateurs. Plus particulièrement, les récentes méthodologies orientées objet ne peuvent pas s'accommoder de concepts fonctionnels plus traditionnels. Nous avons mis en évidence les diverses incompatibilités entre ces deux types d'approches et l'utilité de trouver le moyen de passer de l'une à l'autre.

Une étude bibliographique nous a permis de constater la polémique que suscite ce sujet et d'étudier les divers moyens proposés pour effectuer la translation recherchée. L'analyse des diverses techniques utilisées nous a permis de bien cerner les lacunes existantes et les contraintes à respecter pour aboutir à une démarche efficace. Principalement, nous avons constaté que la plupart des techniques étudiées reposent exclusivement sur l'aspect statique des systèmes. Ce type d'approche n'est pas toujours satisfaisant pour les systèmes temps réels où l'aspect dynamique prend un poids non négligeable. Nous avons pu proposer d'autres approches pour la recherche d'une solution. Ces dernières sont néanmoins peu satisfaisantes.

Pour la recherche d'une solution efficace, nous nous sommes orientés vers un outil très utilisé pour la modélisation des systèmes temps réel : les réseaux de Petri. L'existence d'une certaine analogie entre les diagrammes de flots de données et les réseaux de Petri, d'une part, et la possibilité d'exprimer l'aspect dynamique des spécifications par le même outil, d'autre part, nous ont permis d'entrevoir une solution au problème. Il s'agit d'exploiter les propriétés structurelles des réseaux de Petri pour effectuer le passage entre les spécifications fonctionnelles et les spécifications orientées objet.

Nous avons entrepris la représentation des divers aspects des spécifications fonctionnelles du type SA-RT à l'aide des réseaux de Petri. Cette représentation intègre les aspects statiques et dynamiques des diagrammes SA-RT. Pour l'aspect statique, elle est basée sur la circulation des données dans les diagrammes de flots de données. Elle a également été effectuée dans le souci de garder la lisibilité des diagrammes. La repré-

sentation proposée n'est pas toujours automatisable mais comprend de larges parties qui peuvent l'être. Elle permet en outre de formaliser certains aspects ambigus des spécifications SA-RT.

L'exploitation des réseaux de Petri obtenus pour le passage des spécifications fonctionnelles aux spécifications orientées objet a été entreprise selon deux démarches distinctes. La première convient aux systèmes à faible composante dynamique. Elle s'appuie sur l'aspect statique pour la définition des objets. Ces derniers sont par la suite complétés par les considérations dynamiques. La seconde convient plutôt aux systèmes à forte composante dynamique. Elle s'appuie sur l'aspect dynamique en premier lieu, puis complète les objets obtenus par les opérations et les données. Nous avons également proposé une démarche qui permet d'intégrer ces deux aspects dans une même étude pour avoir des résultats plus proches des systèmes considérés.

Les techniques et outils utilisés dans la démarche proposée permettent de retrouver la trace des besoins fonctionnels chaque fois qu'il est nécessaire de remonter le cycle de vie afin d'effectuer une validation ou de rediscuter un problème particulier avec l'utilisateur.

Par ailleurs, l'utilisation des composantes conservatives permet d'envisager la possibilité d'automatiser, en partie, la démarche. Ainsi, elle pourrait être intégrée dans un atelier de génie logiciel, permettant d'assister l'analyste-concepteur depuis la phase d'analyse des besoins jusqu'à la génération automatique de squelettes de code.

L'utilisation des réseaux de Petri pour la description des spécifications SA-RT nous a orientée sur la voie de la validation et de l'évaluation des spécifications SA-RT. Nous avons pu étudier l'apport des réseaux de Petri pour ces objectifs. Principalement, l'adoption du modèle temporel des réseaux de Petri pour la représentation des spécification SA-RT permet d'envisager l'évaluation des spécifications fonctionnelles et temporelles, implicites et explicites simultanément. En effet, les réseaux de Petri temporels, déjà utilisés pour l'évaluation des performances des systèmes temps réel, permettent d'intégrer sur le même support les divers aspects des spécifications. En outre, l'utilisation du modèle stochastique temporisé permet d'ajouter une composante stochastique à l'étude.

Les solutions proposées pour résoudre le problème du passage d'une analyse fonctionnelles à une analyse orientée objet ouvrent la voie à de nouvelles perspectives de recherche notamment sur les points suivants :

- L'exploitation "industrielle" de l'approche proposée pour asseoir ses capacités et ses limites.
- L'automatisation partielle de la démarche proposée.
- Son intégration dans un environnement de génie logiciel plus complet basée sur les travaux déjà effectués. Principalement sur les méthodes SA-RT, HOOD/PNO et les réseaux de Petri.
- L'approfondissement des perspectives d'évaluation des spécifications grâce aux travaux effectués sur les réseaux de Petri temporisés stochastiques.

Annexe A

Étude de cas

Cette annexe est dédiée à l'étude d'un exemple proposé par le groupe AFCET "Objets et temps réel" pour la comparaison des méthodologies d'analyse et de conception des systèmes temps réel. Le volume prohibitif de l'étude complète ne permet pas de l'intégrer en totalité dans cette annexe. Nous nous limiterons à exposer brièvement les parties les plus intéressantes pour notre problématique: description sommaire du problème, représentation de quelques niveaux de spécifications SA/RT et les passages les plus importants pour la translation en analyse orientée objet.

A.1 Le problème

Le système à concevoir doit gérer un système ferroviaire. Il est composé d'un réseau ferroviaire divisé en rails (rails simples ou aiguillages), un ensemble de trains ayant chacun un trajet à parcourir, un boîtier de commande et un panneau d'affichage fournissant des informations sur l'état des trains et du réseau. A chaque train est associé un trajet et il s'agit de lui faire exécuter son parcours sans collision. Les trains sont alimentés électriquement par la voie et sont commandés à l'aide d'un signal de commande superposé au signal d'alimentation.

L'acquisition de l'état du réseau ferroviaire se fait moyennant un ensemble de capteurs délimitant les rails. Les capteurs détectent le passage d'un train et renvoient au système de commande une information sur les numéros du capteur et du train détecté.

A chaque passage devant un capteur, le train "s'attend" à recevoir une commande lui indiquant s'il peut continuer et lui donnant sa nouvelle commande en vitesse. S'il ne reçoit pas cette commande au bout de $110ms$, il s'arrête.

Les aiguillages permettent de guider les trains sur des sections différentes. A l'abord d'une section commune, la priorité d'entrée est donnée au train le plus rapide. La vitesse réelle des trains n'est pas connue, le système estime la vitesse des trains à partir des dates de leurs passages sur les capteurs. Le cahier des charges, assez volumineux, donne plusieurs autres contraintes fonctionnelles et temporelles non reprises ici.

Nous exposons dans cette annexe deux approches, basées sur deux points de vue différents, ainsi que les translation en objet correspondantes. Dans les deux cas nous nous limitons à quelques niveaux de spécifications suffisants pour illustrer la démarche.

A.2 Approche #1

A.2.1 Spécifications par SA-RT et réseaux de Petri

Le diagramme de contexte est illustré par la figure *A.1*. Il montre les diverses entités en relation avec le système de commande et les échanges d'informations effectuées.

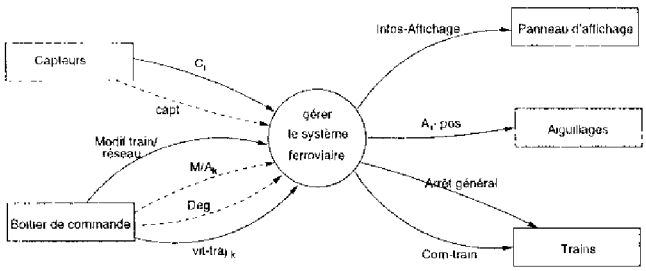


fig A.1 – diagramme de contexte

La figure *A.2* présente le développement du niveau 0. Ce niveau illustre les fonctions principales assurées par le système. Il montre que la solution adoptée est centrée autour de deux structures de données principales : «Les trains» et «Le réseau» [1].

Le stock "*Les trains*" donne pour chaque train les informations suivantes : numéro, position, vitesse de consigne, vitesse estimée, trajet, état (marche, attente), durée d'attente, capteur bloquant, dates de passage (sur les deux derniers capteurs). Le stock "*Les réseaux*" donne des informations sur la structure du réseau ferroviaire et son état (pour chaque rail occupé/libre, le train qui l'occupe et la position de chaque aiguillage).

Parmi les différentes fonctions importantes réalisées par le système nous allons nous intéresser à la transformation "gérer la progression des trains" qui rassemble les activités principales liées au contrôle des trains. Elle est développée selon la figure *A.3*.

Cette transformation est activée à chaque déclenchement de capteur. A chaque activation, le train qui a déclenché le capteur est identifié et les données le concernant sont mises à jour. La structure de données correspondante est déposée dans le stock «train

1. Dans plusieurs DFDs et dans les réseaux de Petri les représentant, les stocks de données sont dupliqués pour une facilité de représentation et une meilleure clarté des diagrammes.

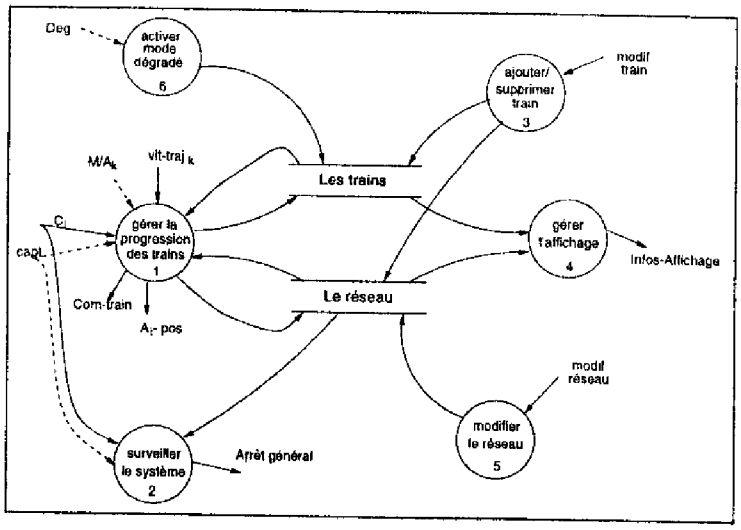


fig A.2 – Niveau 0 - Gérer le système ferroviaire

courant» afin qu'il soit traité par "*commander le train*" qui fait le nécessaire pour faire avancer le train ou le mettre en attente.

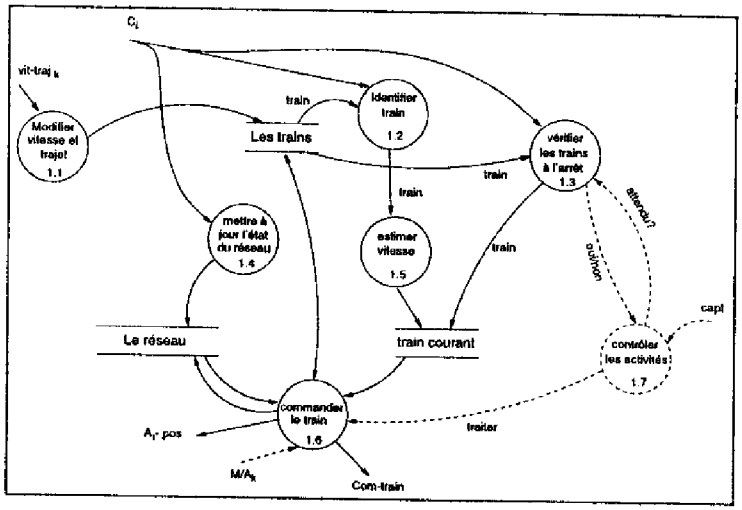


fig A.3 – Niveau 1 - DFD 1 - Gérer la progression des trains

Par ailleurs, le capteur activé est aussi susceptible de débloquer l'arrêt d'un train qui était en attente de libération d'un rail ou d'un aiguillage. Dans ce cas, le train est sélectionné et la structure de données correspondante est déposée dans «train courant» afin qu'il soit également traité. La coordination entre les trains à traiter est assurée par la transformation de contrôle "1.7- contrôler les activités" dont la Cspec est spécifiée par le réseau de Petri de la figure *A.4*.

Remarque: Le DFD de la figure *A.3* suppose une sérialisation du traitement des trains. Ceci est applicable tant qu'il s'agit de l'étape d'analyse des besoins. En effet, le cahier des charges propose une sérialisation de l'information provenant des capteurs.

Au passage à la conception il faudra prendre en compte la possibilité de traiter parallèlement les trains.

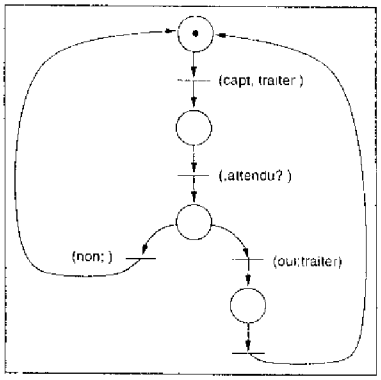


fig A.4 – Niveau 1 - Cspec 1.7 - Contrôler les activités

Parmi les différentes transformations de données du niveau 1 nous développons la transformation "1.6 Commander le train" (figure *A.5*). Elle s'occupe de la réservation du prochain rail (en fait il faut réserver deux rails à l'avance pour assurer la sécurité du réseau). Si ce rail est un aiguillage il faut alors effectuer un traitement particulier à l'aiguillage (vérifier la priorité des éventuels trains à l'entrée de l'aiguillage et commander l'aiguillage).

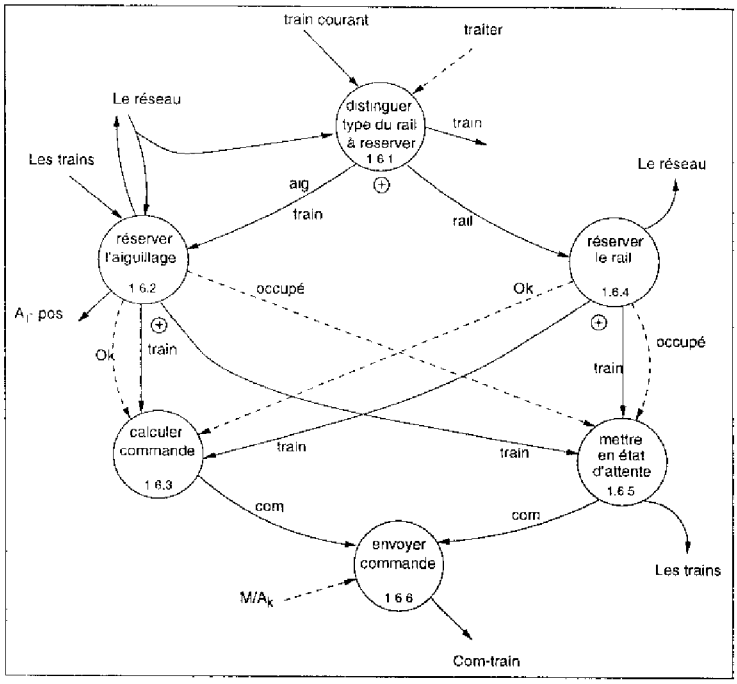


fig A.5 – Niveau 2 - DFD 1.6 - Commander le train

La réservation du rail ou de l'aiguillage peut échouer (le rail peut être occupé par un autre train), dans ce cas le train est mis en attente (envoi de commande d'arrêt et

mémorisation du capteur qui débloquera la situation). Dans le cas où la demande de réservation est satisfaite, la nouvelle commande de train est calculée (en fonction de la vitesse de consigne et de la vitesse estimée) puis envoyée au train. L'opérateur peut demander à tout moment l'arrêt d'un train particulier.

La transformation de données "1.6.2 Réserver l'aiguillage" semble la plus complexe. Elle est décomposée dans le diagramme de la figure *A.6*. Dans le cas où deux trains désirent emprunter la même section, avant d'accorder la réservation au premier qui le demande, il faut vérifier la priorité d'entrée, si l'aiguillage n'est pas déjà réservé.

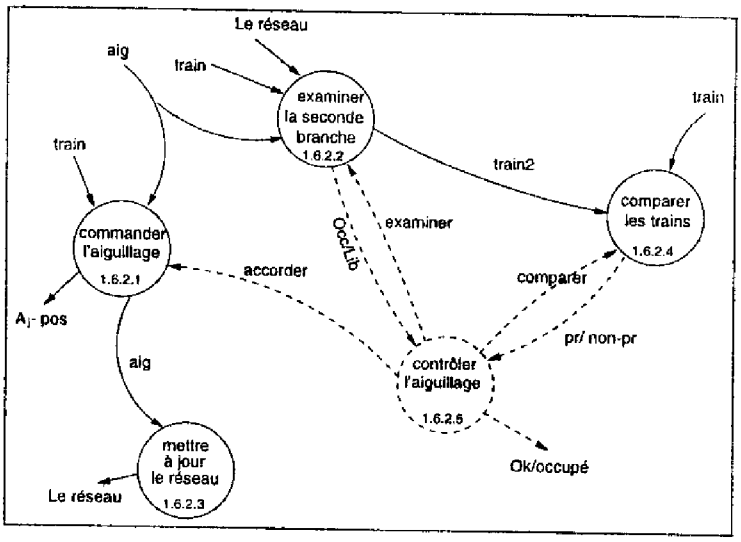


fig A.6 – Niveau 3 - DFD 1.6.2 - Réserver l'aiguillage

Les activités de ce niveau sont gérées par la transformation de contrôle "Contrôler l'aiguillage" dont la Cspec est spécifiée par le réseau de Petri de la figure *A.7*.

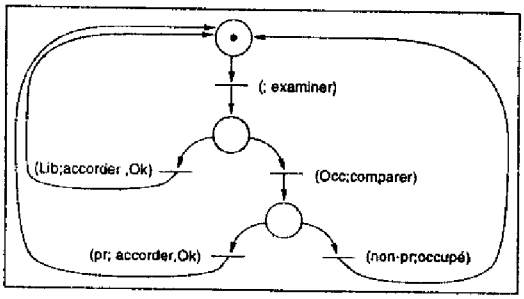


fig A.7 – Niveau 3 - Cspec 1.6.2.5 - Contrôler l'aiguillage

Les spécifications données dans cette partie ne sont pas complètes; elles ne prennent en compte qu'une partie des besoins figurant dans le cahier des charges. Néanmoins, elles donnent un aperçu de la structuration globale de l'approche adoptée. Ces spécifications doivent être complétées par le développement des transformations de données restantes, par la prise en compte des divers besoins fonctionnels et temporels, par l'établissement des mini-spécifications et du dictionnaire des données.

A.2.2 Translation vers une analyse HOOD/PNO

L'examen préliminaire des spécifications proposées par l'*approche #1* montre que l'analyse s'appuie principalement sur les DFDs. L'aspect contrôle a un rôle secondaire, il assure la coordination des fonctions. Nous choisissons donc de conduire l'identification et la définition des objets à partir des DFDs.

Le diagramme de contexte (figure *A.1*) fournit des suggestions sur les objets physiques du système: **capteurs**, **boîtier de commande**, **panneau d'affichage**, **aiguillages** et **trains**. Toutes ces suggestions sont susceptibles de se concrétiser par des objets à l'issue de l'analyse. Cependant, comme nous avons limité notre étude SA-RT à la fonction "gérer la progression des trains", les objets **boîtier de commande** et **panneau d'affichage** ne seront pas définis.

Par ailleurs, comme le montre le DFD de niveau 0 (figure *A.2*), les spécifications s'appuient sur deux structures de données principales: "Les trains" et "Le réseau". Ce sont deux suggestions supplémentaires pour l'identification des objets.

La figure *A.8* montre le réseau de Petri qui représente le DFD 1 du niveau 1 (figure *A.3*).

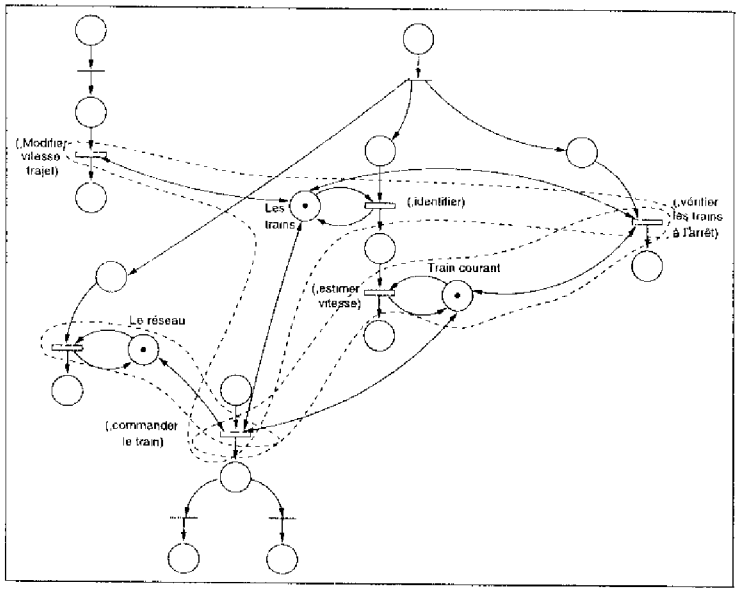


fig A.8 – RdP représentant le DFD 1 du niveau 1

La recherche de composantes conservatives à partir de ce réseau de Petri montre que trois d'entre elles couvrent toutes les opérations associées aux transitions. Elles sont liées aux stocks de données: «Les trains», «Le réseau» et «train courant». D'autres composantes conservatives peuvent être retrouvées mais elles sont redondantes. La présence de composantes liées aux stocks confirme le fait que l'*approche #1* est principalement basée sur les structures de données.

Le réseau de Petri spécifiant la Cspec du niveau 1 (figure *A.4*) ne fournit pas d'in-

formation particulière pour l'identification des objets. Il servira lors de la définition des objets.

Le réseau de Petri représentant le DFD 1.6 du niveau 2 (*commander le train* figure *A.5*) est présenté dans la figure *A.9*. L'étude de ce réseau de Petri ne révèle pas de nouveaux objets; les mêmes objets relatifs aux stocks de données y sont retrouvés. Il existe également une composante conservative qui couvre toutes les opérations du réseau et qui peut concerner l'objet **train courant**.

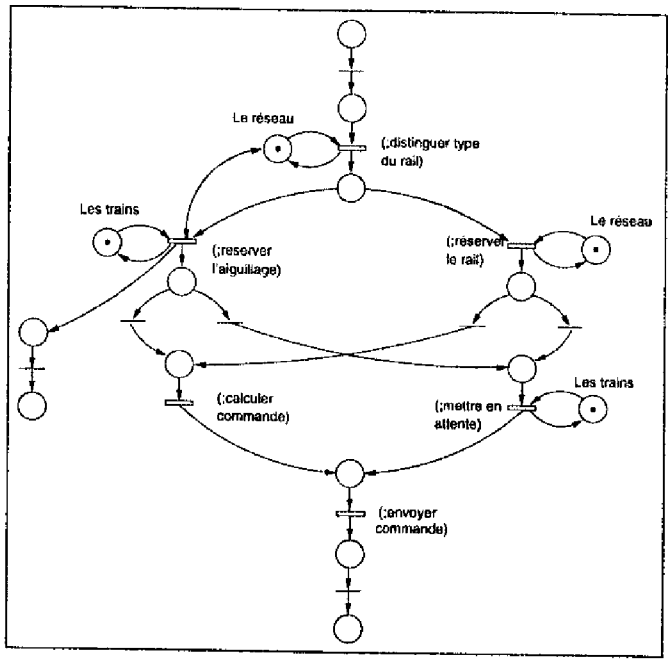


fig A.9 – RdP représentant le DFD 1.6 du niveau 2

Le réseau de Petri de la figure *A.10* représente le DFD 1.6.2 du niveau 3 (*réserver l'aiguillage* fig *A.6*). Il comporte des composantes conservatives reliées à l'objet réseau et une composante qui couvre des fonctions indépendantes des objets déjà identifiés. Cette dernière est représentative d'un objet *aiguillage*. Le choix de cet objet est conforté par le fait qu'il a été également identifié à partir du diagramme de contexte. Par ailleurs, la structure de contrôle associée à la transformation de contrôle "*contrôler l'aiguillage*" (fig *A.7*) est couverte par une seule composante conservative et peut alors être totalement allouée à l'objet **aiguillage**.

Ainsi, après l'examen des spécifications nous avons a priori 4 objets pertinents pour le problème: **trains**, **train courant**, **réseau** et **aiguillage**.

L'examen des fonctions associées aux objets **trains** et **train courant**, montre qu'il s'agit en fait de deux vues d'un même objet. En effet, les fonctions associées à la structure de données "les trains" permettent d'accéder aux données concernant les trains en circulation. D'autre part, les fonctions associées au "train courant" concernent les opérations de commande du train. Ainsi, nous avons choisi de regrouper ces deux vues dans un seul objet: un objet **train** instancié pour chaque train en circulation sur

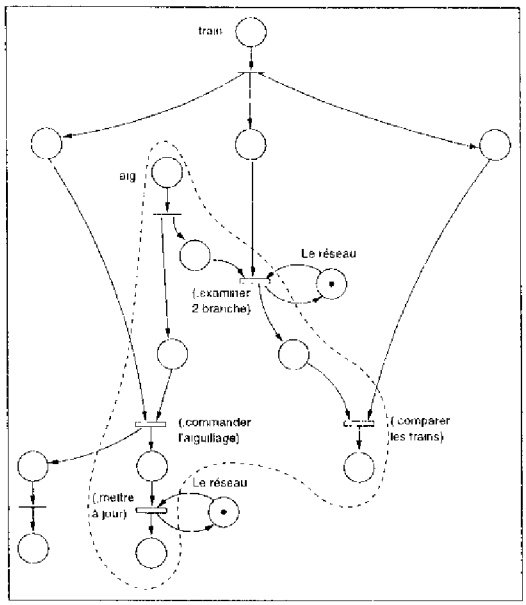


fig A.10 – RdP représentant le DFD 1.6.2 du niveau 3

le réseau.

L'objet **réseau** est un objet passif qui gère les données relatives à la structure du réseau ferroviaire et à l'état de ses différentes parties. Il offre les opérations nécessaires à l'accès à ces données.

L'objet **aiguillage** s'occupe de la gestion des priorités et de la commande d'un aiguillage. Un objet **aiguillage** est instancié pour chaque aiguillage du réseau. Il est possible de définir deux classes pour les objets aiguillages et trains.

Après identification des objets pertinents, il s'agit de définir leurs méthodes à partir des opérations regroupées à l'aide des composantes conservatives. Il faut également préciser le comportement interne des objets actifs et les communications entre objets.

La figure *A.11* illustre la répartition en objets de la partie traitée du problème. Elle devra être complétée en prenant en compte les besoins qui ne l'ont pas été et en traitant la partie relative au boîtier de commande et au panneau d'affichage.

La figure *A.12* présente les PNOBCS des objets **Train** et **Aiguillage**. Le PNOBCS de l'objet **Train** n'est pas complet car toutes les fonctions des DFDs étudiés n'ont pas été développées.

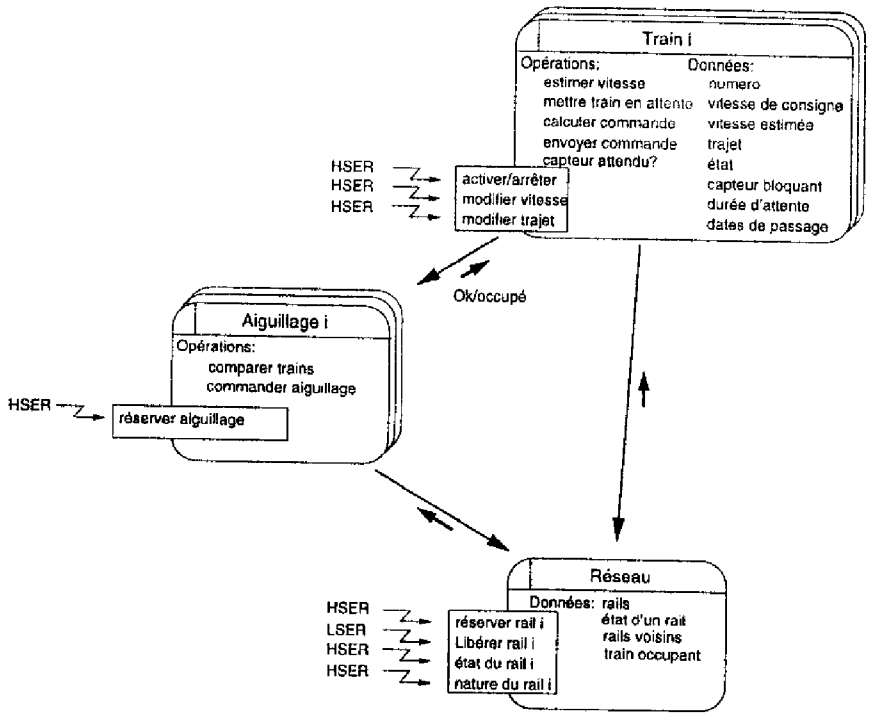


fig A.11 – Répartition en objets

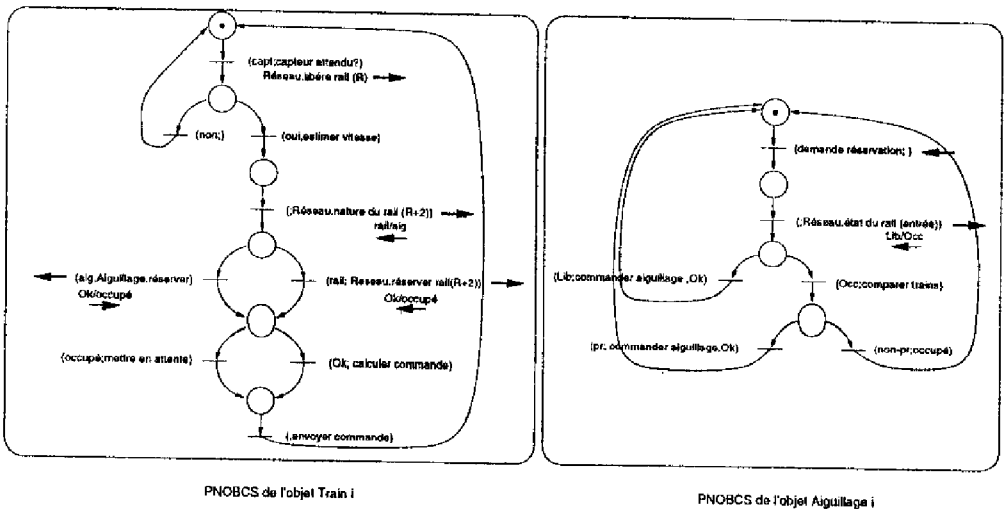


fig A.12 – Réseaux de Petri des PNOBCS

A.3 Approche #2

A.3.1 Spécifications par SA-RT et réseaux de Petri

La seconde approche abordée est issue d'une démarche orientée vers le contrôle des activités plutôt que vers la structuration des différentes fonctions réalisées. Dès le premier niveau, elle montre la centralisation de l'analyse sur le contrôle des activités nécessaires à la progression des trains. Les spécifications présentées donnent simplement l'ébauche d'une solution. Elles permettront d'illustrer une seconde démarche pour conduire la translation vers une analyse orientée objet.

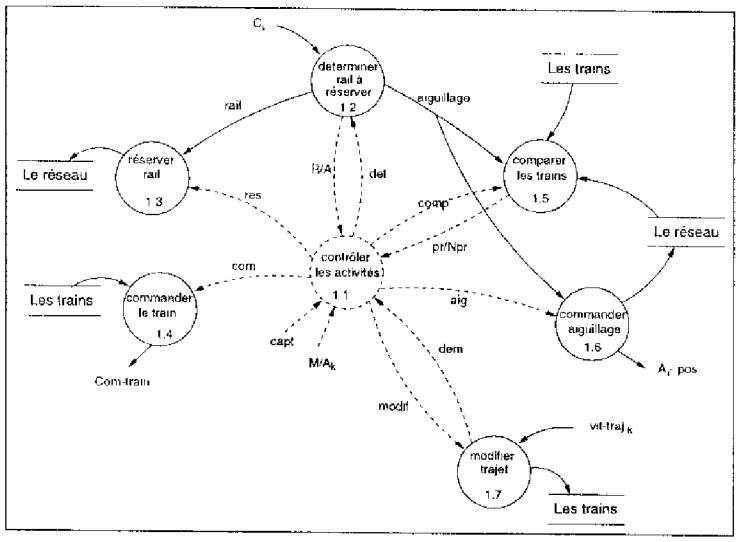


fig A.13 - Niveau 1 - DFD 1 - Gérer la progression des trains

Le DFD présenté dans la figure *A.13* est obtenu en développant (selon une nouvelle approche) la transformation de données "*1 Gérer la progression des trains*" de la figure *A.2*. Ce niveau illustre les diverses activités centrées autour de la transformation de contrôle. Les fonctions principales réparties sur plusieurs niveaux dans l'*approche #1*, apparaissent ici d'une façon abstraite au même niveau.

Le réseau de Petri de contrôle, présenté dans la figure *A.14* s'appuie sur le même raisonnement que celui adopté dans l'*approche #1*: une activité cyclique du système, effectuée à chaque activation de capteur. Le réseau de Petri présenté est un réseau de haut niveau (à objets, ou prédicats-transitions). Les ressources relatives aux différents rails sont rassemblées dans la place "Rails". De même, les ressources aiguillages sont également rassemblées dans la place "Aiguillages".

A chaque passage sur un capteur, un rail est libéré. Si le prochain (R+2) est un rail standard, le train le réserve (s'il est libre) et continue son trajet. S'il s'agit d'un aiguillage, le train demande à l'aiguillage la priorité. Elle lui est accordée ou non selon la situation des autres trains par rapport à l'aiguillage. Si le train n'a pas eu la priorité, il reste en attente et redemande la priorité jusqu'à satisfaction. Tel qu'il est présenté,

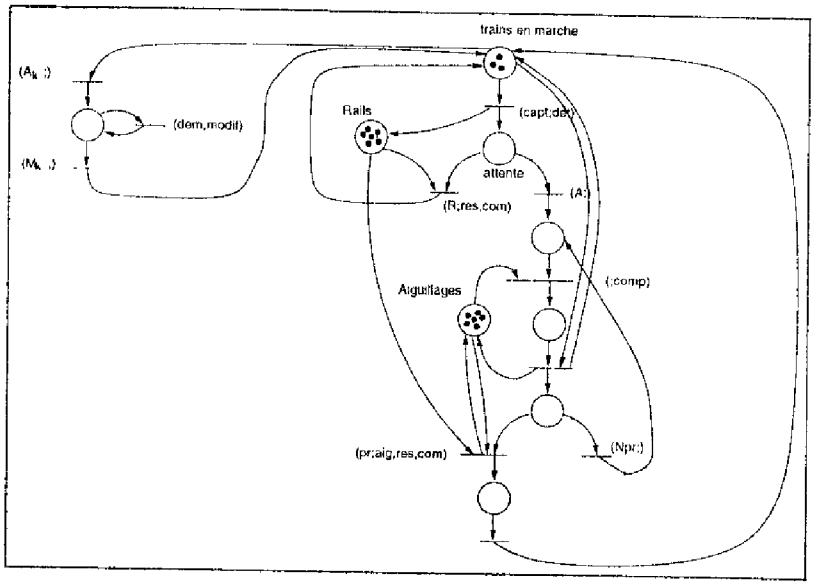


fig A.14 – Niveau 1 - Cspec 1.6 - Contrôler

le réseau de Petri seul ne garantit pas que le train ne restera pas indéfiniment bloqué. Il faut prévoir au niveau de la gestion des priorités, dans la transformation de données "*comparer les trains*", la possibilité d'accorder la priorité à un train qui a attendu "trop longtemps".

Parallèlement, les trains en marche peuvent être arrêtés par l'utilisateurs et ce n'est qu'à l'arrêt que leur trajet peut être modifié.

A.3.2 Translation vers une analyse HOOD/PNO

Comme nous l'avons remarqué plus haut, les spécifications de l'*approche #2* sont dirigés plutôt vers l'aspect dynamique. C'est une démarche de translation s'appuyant sur cet aspect que nous voulons illustrer ici.

Le réseau de Petri représentant le contrôle peut être couvert par trois composantes conservatives. Celles-ci sont relativement bien visible sur la figure *A.14*. Deux d'entre elles sont relatives aux ressources utilisées (les rails et les aiguillages); la troisième est relative aux opérations de commande du train. Par ailleurs ces entités ayant été pressenties comme objets à partir de l'étude du contexte du problème, les composantes conservatives trouvées sont retenues pour la définition des objets du problème.

Les composantes trouvées sont multi-marquées (voir figure *A.14*). Il est donc possible de définir plusieurs objets de même type pour chaque ressource. Il est également intéressant de définir des classes à partir de ces composantes.

La figure *A.15* présente la description des objets ainsi que les données et les opérations qui leur sont associées. Ce diagramme est moins complet que le précédant car seule une partie des spécifications a été prise en compte, c'est principalement la démarche

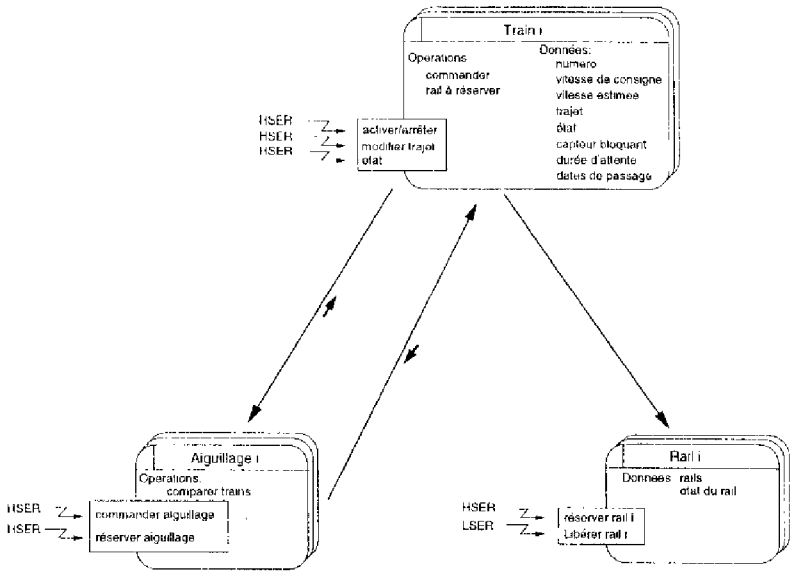


fig A.15 – Description des objets

adoptée que nous voulons illustrer ici.

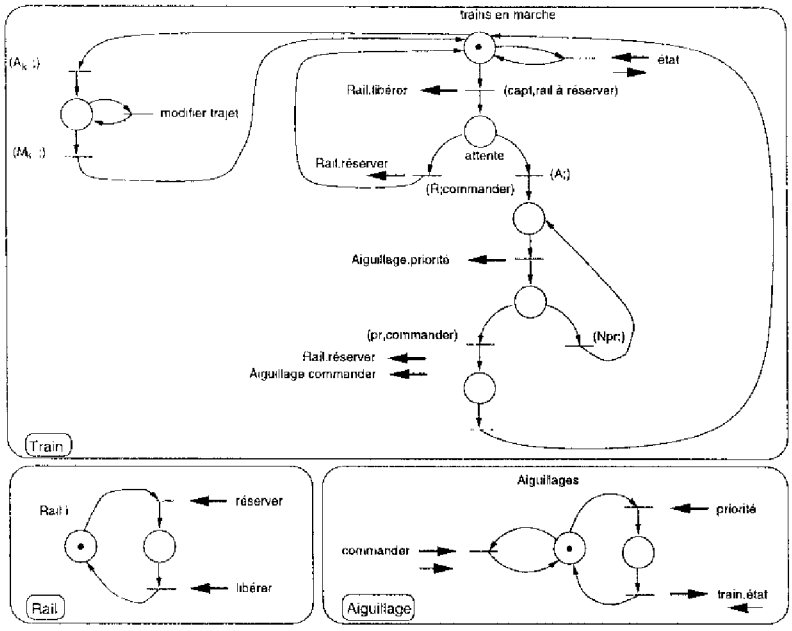


fig A.16 – PNOBCS des objets définis

La figure *A.16* précise les PNOBCS de chaque objet. Il est à remarquer que la structuration générale de la spécification en objet ressemble à celle définie dans l'*approche #1* mais les fonctionnalités allouées au objets diffèrent légèrement. En fait, il ne s'agit pas d'aboutir à la même spécification en objets mais plutôt d'aboutir à une spécification

en objet cohérente.

Par ailleurs, il est nécessaire de continuer la démarche par une phase d'analyse des besoins orientées objet pour affiner la définitions des objets et améliorer la solution choisie.

Ainsi, cet exemple appelé "*problème du train*" nous a permis d'illustrer la méthode proposée pour la translation d'une analyse fonctionnelle vers une analyse orientée objet. Deux techniques ont été utilisées selon le type des spécifications fonctionnelles de départ.

Malheureusement, comme tout exemple applicatif, ce problème n'illustre pas toutes les solutions que peuvent apporter les réseaux de Petri au problème d'identification et de définition des objets. Il est difficile de trouver une application qui couvre tous les aspects de la problématique. Néanmoins, l'étude menée permet de donner un aperçu global de la démarche proposée.

Références bibliographiques

[Alabiso 88] B. Alabiso. Transformation of data flow analysis models to object oriented design. *OOPSALA '88 Conference Proceedings*, SIGPLAN Notices, pages 335–353, Septembre 1988.

[Alaiwan et Toudic 85] H. Alaiwan et J. M. Toudic. Recherche des semi-flots, des verrous et des trappes dans les réseaux de Petri. *Technique et Science informatiques - Spécial Réseaux de Petri*, Vol. 4, n^o 1, pages 103–112, Janvier-Février 1985.

[Atamna 93] Y. Atamna. RdPTS, A Tool for Stochastic Timed Petri Nets. *PNPM'93 Fifth International Workshop on Petri Nets and Performance Models*, Toulouse, 20-22 Octobre 1993.

[Atamna 94] Y. Atamna. *Réseaux de Petri Temporisés Stochastiques Classiques et Bien Formés: Définition, Analyse et Application aux Systèmes Distribués Temps Réel.* Thèse de Doctorat, Université Paul Sabatier, Toulouse, Octobre 1994.

[Bachatene 94] H. Bachatene. *Une approche modulaire intégrant les réseaux colorés pour la spécification de systèmes distribués.* Thèse de Doctorat, Université Pierre et Marie Curie, Septembre 1994.

[Bailin 89] S. C. Bailin. An Object-Oriented Requirements Specification Method. *Communications of the ACM*, Vol. 32, n^o 5, pages 608–623, Mai 1989.

[Ben Ahmed *et al.* 96] S. Ben Ahmed, M. Moalla, et M. Courvoisier. Approche multi-modèles utilisant SA-RT et les réseaux de Petri pour la spécification de la commande des systèmes de production flexibles. *Actes du Congrès Modélisation des systèmes réactifs, MSR'96*, AFCET, pages 45–54, Brest 28 et 29 Mars 1996.

[Benzina *et al.* 97] A. Benzina, M. Paludetto, et J. Delatour. About the Suitability of Petri Nets for Describing, Validating and Evaluating SA-RT Specifications. *APSEC/ICSC'97- Asia Pacific Software Eng. Conf. and International Computer Science Conf.*, Hong Kong 2-5 Décembre 1997.

[**Benzina et Paludetto 96a**] A. Benzina et M. Paludetto. Méthodologies de Conception pour le Génie logiciel Temps réel. *Forum de Recheche en Informatique FRI'96*, Tunis 16-18 Juillet 1996.

[**Benzina et Paludetto 96b**] A. Benzina et M. Paludetto. Systèmes dynamiques et utilisation conjointe de la méthode SA-RT et des réseaux de Petri. *Actes du Congrès Modélisation des systèmes réactifs, MSR'96*, AFCET, pages 173-179. Brest 28 et 29 Mars 1996.

[**Benzina et Paludetto 97**] A. Benzina et M. Paludetto. Une méthodologie fonctionnelle SA-RT et réseaux de Petri. *Concepts et Outils pour les Systèmes de Production.* Cépaduès Editions - Collection Automatique et Production - Coordination J.-C. Hennet, Toulouse, 1997.

[**Berard 89**] E. V. Berard. *Object Oriented Requirements Analysis.* EVB Software Engineering Inc, Frederick, Maryland, 1989.

[**Berthomieu et Menasche 82**] B. Berthomieu et M Menasche. A state enumeration approach for analyzing Time Petri Nets. *proceedings of the 3rd European Workshop on Applications and Theory of Petri Nets*, pages 27–65, Italie, 27-30 Septembre 1982.

[**Boehm *et al.* 84**] B. W. Boehm, T. L. Gray, et T. Seewaldt. Prototyping Versus Specifying: A Multiproject Experiment. *IEEE Transactions on Software Engineering*, Vol. SE-10, n^o 3, pages 290–302, Mai 1984.

[**Booch 86**] G. Booch. Object-Oriented Development. *IEEE Transactions on Software Engineering*, Vol. 12, n^o 2, pages 211–221, Février 1986.

[**Booch 94**] G. Booch. *Object-Oriented Analysis and Design - 2^{nd} Ed.* Benjamin-Cummings, Redwood city, California, 1994.

[**Bourey et Gentina 91**] J. P. Bourey et J. C. Gentina. Conception structurée et modulaire d'architectures de contrôle réparti en production flexible manufacturière. *Automatique - Productique - Informatique Industrielle*, Vol. 25, n^o 4, pages 349–376, 1991.

[**Brams 83**] G. W. Brams. *Réseaux de Petri: théorie et pratique*, vol. 1 et 2. Masson Editeur, 1983.

[**Chen 76**] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, Vol. 1, n^o 1, pages 9-36, Mars 1976.

[**Coad et Yourdon 89**] P. Coad et E. Yourdon. *OOA Object Oriented Analysis.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[**Courvoisier et Valette 80**] M. Courvoisier et R. Valette. *Systèmes de commande temps réel.* Edition SMC, 1980.

[**David et Alla 89**] R. David et H. Alla. *Du Grafcet aux Réseaux de Petri.* Editions Hermès, 1989.

[**de Champeaux *et al.* 90**] D. de Champeaux, L. Constantine, I. Jacobson, S. Mellor, P. Ward, et E. Yourdon. PANEL: Sructured Analysis and Object Oriented Analysis. *OOPSLA/ECOOP '90 Conference Proceedings*, vol. 24, n^o 10, *SIGPLAN Notices*, pages 135-139, Octobre 1990.

[**Delfieu et Sahraoui 94a**] D. Delfieu et A. E. K. Sahraoui. Automata Timing Specification. *11th International Conference on Analysis and Optimization of Systems*, pages 115–121, Sophia-Antipolis, 15-17 Juin 1994.

[**Delfieu et Sahraoui 94b**] D. Delfieu et A. E. K. Sahraoui. Expression and Validation of Timing Constraints and Integration in Software Specification Methods. *IEEE Workshop on Real-Time Applications*, pages 63–68, Washington, 21-22 Juillet 1994.

[**DeMarco 78**] T. DeMarco. *Structured Analysis and System Specification*. Yourdon Press, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[**Di Giovanni 91**] R. Di Giovanni. Hood nets. E. Rosenberg Editeur, *Advances in Petri Nets*, Lecture Notes in Computer Sciences - LNCS 524. Springer Verlag, 1991.

[**DoD 88**] DoD. *Defense System Software Development*, vol. DoD Std 2167A. U.S. Depart. of Commerce, NTIS n Ada 198 825, Février 1988.

[**Elloy *et al.* 88**] J. P. Elloy *et al.* Le temps réel - Rapport établi par le Groupe de Réflexion Temps Réel du CNRS. *Technique et Science Informatiques*, Vol. 7, n^o 5, pages 493–500, Mai 1988.

[**Elmstrom *et al.* 93a**] R. Elmstrom, P. B. Lassen, et M. Andersen. An Executable Subset of VDM-SL in an SA/RT Framework. *Real-Time Systems*, Vol. 5, n^o 2/3, pages 197–211, Mai 1993.

[**Elmstrom *et al.* 93b**] R. Elmstrom, R. Lintulampi, et M. Pezzé. Giving Semantics to SA/RT by Means of High-Level Timed Petri Nets. *Real-Time Systems*, Vol. 5, n^o 2/3, pages 249–271, Mai 1993.

[**ESA 89**] ESA. *HOOD User Manual, HOOD Reference Manual*. European Space Agency, version 3.0, Septembre 1989.

[**Felder *et al.* 93**] M. Felder, C. Ghezzi, et M. Pezzé. High-Level Timed Petri Nets as a Kernel for Executable Specifications. *Real-Time Systems*, Vol. 5, n^o 2/3, pages 235–248, Mai 1993.

[**Firesmith 91**] D. Firesmith. Structured analysis and object oriented development are not compatible. *ADA Letters*, Vol. XI, n^o 9, pages 56–66, Nov/Dec 1991.

[**France 92**] R. France. Semantically Extended Data Flow Diagrams: A Formal Specification Tool. *IEEE Transactions on software Engineering*, Vol. SE-18, n^o 4, pages 329–346, Avril 1992.

[**Fuggetta *et al.* 93**] A. Fuggetta, C. Ghezzi, D. Mandrioli, et A. Morzenti. Executable Specifications with Data-flow Diagrams. *Software-Practice and Experience*, Vol. 23, n^o 6, pages 629–653, Juin 1993.

[**GAO 79**] US Government Accounting Office GAO. Contracting for computer software development. Serious problems require management attention to avoid wasting additionnal millions. Rapport, FGMSD-80-4, Novembre 1979.

[**Genrich 86**] H. J. Genrich. Predicate/Transition Nets. *Advances in Petri Nets*. Lecture Notes in Computer Sciences - LNCS 254, pages 207-247. Springer Verlag, 1986.

[**George 96**] J. George. A Strategy for Mapping from Function-Oriented Software Models to Object-Oriented Software Models. *ACM SIGSOFT - Software Engineering Notes*, Vol. 21, n^o 2, pages 56 63, Mars 1996.

[**Ghezzi *et al.* 91**] C. Ghezzi, D. Mandrioli, S. Morasca, et M. Pezzé. A Unified High-Level Petri Net Formalism for Time-Critical Systems. *IEEE Transactions on Software Engineering*, Vol. 17, n^o 2, pages 160 172, Février 1991.

[**Giron 96**] P. Giron. Formalisation de la sémantique de diagrammes SA-RT en termes de systèmes de transitions étiquetés. *Actes du Congrès Modélisation des systèmes réactifs, MSR'96*, AFCET, pages 359–366, Brest 28 et 29 Mars 1996.

[**Gray 88**] L. Gray. Transitionning from Structured Analysis to Object Oriented Design. *Proceedings of the 5th Washington Ada Symposium*, pages 151 162, 27-30 Juin 1988.

[**Harel et Pnueli 85**] D. H. Harel et A. Pnueli. On the Development of Reactive Systems in Logic and Models of Concurrent Systems. *NATO ASI series in Computer Sciences*, vol. 13, pages 447–498. Springer Verlag, 1985.

[**Hatley et Pirbhai 87**] D. J. Hatley et I. A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House Publishing, New York, 1987.

[**Jacobson *et al.* 92**] I. Jacobson, M. Chrisrterson, P. Jonsson, et G. Overgaard. *Object Oriented Software Engineering: a use case driven approach*. Addison-Wesley, 1992.

[**Jensen 81**] K. Jensen. Colored Petri Nets and the invariant method. *Theory of Computer Science*, Vol. 14, 1981.

[**Juanole et Gallon 96**] G. Juanole et L. Gallon. Le modèle réseaux de Petri temporisés stochastiques et son adéquation aux systèmes distribués temps critique. *Actes du Congrès Modélisation des systèmes réactifs, MSR'96*, AFCET, Brest 28 et 29 Mars 1996.

[**Khalsa 89**] G. K. Khalsa. Using Object Modeling to Transform Structured Analysis into Object Oriented Analysis. *Proceedings of the 6th Washington Ada Symposium*, pages 201–212, 26-29 Juin 1989.

[**Krückeberg et Jaxy 87**] F. Krückeberg et M. Jaxy. Mathematical Methods for Calculating Invariants in Petri Nets. E. Rosenberg Editeur, *Advances in Petri Nets*, Lecture Notes in Computer Sciences - LNCS 266, pages 104–131. Springer Verlag, 1987.

[**Laplante 92**] P. A. Laplante. *Real-Time Systems Design and Analysis, an Engineer's handbook*. IEEE Computer Society Press, 1992.

[Lee et Tan 92] P. T. Lee et K. P. Tan. Modelling of visualised data-flow diagrams using Petri net model. *Software Engineering Journal*, pages 4–12, Janvier 1992.

[McMenamin et Palmer 84] S. McMenamin et J. Palmer. *Essential Systems Analysis.* Yourdon Press, 1984.

[Merlin et Farber 76] P. Merlin et D. J. Farber. Recoverability of communication protocols. *IEEE Transactions on Communications*, Vol. Com-24, n^o 9, pages 1036-1043, Septembre 1976.

[Miller 56] G. A. Miller. The Magical Number Seven, Plus or Minus two. *The Psychological Review*, Vol. 63, n^o 2, Mars 1956.

[Mrhailaf et Sahraoui 93] R. Mrhailaf et A. E. K. Sahraoui. DFD Extended Methods for Specifying Hybrid Systems. *IEEE Systems Man and Cybernetics*, pages 681-686, Le Touquet, 17-20 Octobre 1993.

[Murata 89] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, Vol. 77, n^o 4, pages 541–580, Avril 1989.

[Paludetto 91] M. Paludetto. *Sur la commande de procédés industriels: une méthodologie basée objets et réseaux de Petri.* Thèse de Doctorat, Université Paul Sabatier, Toulouse, Décembre 1991.

[Paludetto *et al.* 90] M. Paludetto, R. Valette, et M. Courvoisier. Génération de code Ada à partir d'une approche orientée objet Hood/Réseaux de Petri. *Actes des Journées Internationales sur Le Génie Logiciel et ses Applications*, pages 795–826, Toulouse, 3-7 Décembre 1990.

[Paludetto et Benzina 97] M. Paludetto et A. Benzina. Une méthodologie orientée objet Hood et réseaux de Petri. *Concepts et Outils pour les Systèmes de Production.* Cépaduès Editions - Collection Automatique et Production - Coordination J.-C. Hennet, Toulouse, 1997.

[Paludetto et Raymond 93] M. Paludetto et S. Raymond. A methodology based on objects and Petri nets for development of real time software. *IEEE Systems Man and Cybernetics*, Le Touquet, 17-20 Octobre 1993.

[Parnas 72] D. L. Parnas. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM*, Vol. 15, n^o 12, pages 1053–1058, Décembre 1972.

[Peterson 81] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[Pulli *et al.* 88] P. Pulli, j. Dahler, H. P. Gisiger, et A. Kundig. Execution of Ward's Transformation Schema on the Graphic Specification and Prototyping Tool Specs. *Poceedings of COMPEURO-88*, pages 16–25, 1988.

[Pulli et Elmstrom 93] P. Pulli et R. Elmstrom. IPTES: A Concurrrent Engineering Approach for Real-Time Software Development. *Real-Time Systems*, Vol. 5, n^o 2/3, pages 139–152, Mai 1993.

[**Ramchandani 74**] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. Rapport, MIT. Project MAC. TR-120, 1974.

[**Rational 97**] Rational. *UML- Summary v1.0.* Rational Software Corporation, http://www.rational.com, Santa clara, USA, January 1997.

[**Raymond et Paludetto 92**] S. Raymond et M. Paludetto. C++ et le parallélisme pour une méthodologie orientée objet. *Actes de la Cinquième Conférence Internationale sur Le Génie Logiciel et ses Applications*, Toulouse, 7-12 Décembre 1992.

[**Richter et Maffeo 93**] G. Richter et B. Maffeo. Toward a Rigorous Interpretation of ESML-Extended Systems Modeling Language. *IEEE Transactions on Software Engineering*, Vol. 19, n^o 2, pages 165–180, Février 1993.

[**Royce 70**] W. W. Royce. Managing the Development of Large Software Systems: Concepts and Techniques. *Proceedings of WESCON*, Août 1970.

[**Rumbaugh 96**] J. Rumbaugh. To form a more perfect union: Unifying the OMT and Booch methods. *Journal of object-oriented programming*, Vol. 8, n^o 8, pages 14–18, Janvier 1996.

[**Rumbaugh *et al.* 91**] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, et W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[**Schwarz 92**] J.-J. Schwarz. Langage d'Aide à la Conception d'Applications multitâches-Temps Réel. *Automatique - Productique - Informatique Industrielle*, Vol. 26, n^o 05-06, pages 355–384, 1992.

[**Seidewitz et Stark 87**] E. Seidewitz et M. Stark. Towards a General Object-Oriented Software-Development Methodology. *SIGADA - Ada Letters*, Vol. 7, n^o 4, pages 54–67, Jui-Août 1987.

[**Shi et Nixon 96**] L. Shi et P. Nixon. An Improved Translation of SA/RT Specification Model to High-Level Timed Petri Nets. *Proceedings of FME'96, Formel Models Europe*, Lecture Notes in Computer Science, LNCS 1051, pages 518–537, Oxford University, Mars 1996. Springer Verlag.

[**Shlaer et Mellor 92**] S. Shlaer et S. J. Mellor. *Object Lifecycles - Modeling the World in States.* Yourdon Press, Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[**Shlaer et Mellor 96**] S. Shlaer et S. J. Mellor. Migration from Structured to OO Methodologies. *Embedded Systems Programmig, http://www.projtech.com*, pages 56–66, Avril 1996.

[**Shumate 91**] K. Shumate. Structured analysis and object oriented design are compatible. *ADA Letters*, Vol. XI, n^o 4, pages 78–90, Mai-Juin 1991.

[**Shumate 93**] K. Shumate. BATCES Solution #1: An object-oriented design from functional requirements analysis. *ADA Letters*, Vol. XIII, n^o 6, pages 133–161, Nov-Dec 1993.

[Sibertin-Blanc 85] C. Sibertin-Blanc. High level Petri nets with data structures. *6th Europ. Workshop on Applications and Theory of Petri nets*, pages 335–353, Helsinki, Finland, Juin 1985.

[Sibertin-Blanc 96] C. Sibertin-Blanc. Un environnement pour la simulation de Systèmes Répartis modélisés par Objets Coopératifs. *Actes du Congrès Modélisation des systèmes réactifs, MSR'96*, AFCET, pages 317–324, Brest 28 et 29 Mars 1996.

[Sifakis 77] J. Sifakis. Use of Petri nets for performance evaluation. *Measuring, modeling and evaluating computer systems*, pages 75–93. North-Holland, 1977.

[Silva *et al.* 85] M. Silva, J. Martinez, P. Ladet, et H. Alla. Generalized inverses and the calculation of symbolic invariants for colored Petri nets. *Technique et Science informatiques - Spécial Réseaux de Petri*, Vol. 4, n^o 1, pages 113–126, Janvier-Février 1985.

[Stankovic *et al.* 96] J. A. Stankovic *et al.* Strategic Directions in Real-Time and Embedded Systems. *ACM Computing Surveys*, Vol. 28, n^o 4, pages 751–763, Décembre 1996.

[Stankovic et Ramamritham 88] J. A. Stankovic et K. Ramamritham. Introduction to Real-Time Systems. *Tutorial on Hard Real-Time Systems*. ISBN 0-8186-0819-6, Février 1988.

[TNI 96] TNI. *STOOD 4.0*. Techniques Nouvelles d'Informatique, 1996.

[Tse et Pong 89] T. H. Tse et L. Pong. Towards a Formal Foundation for DeMarco Data Flow Diagrams. *The Computer Journal*, Vol. 32, n^o 1, pages 1–12, Janvier 1989.

[Valette 76] R. Valette. *Sur la description, l'analyse et la validation des systèmes de commande parallèles*. Thèse de Doctorat, Université Paul Sabatier, Toulouse, Novembre 1976.

[Valette 79] R. Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, Vol. 18, n^o 1, pages 35–46, Février 1979.

[Valette 95] R. Valette. Les Réseaux de Petri. Support de cours. http://www.laas.fr/~robert/cours96.ps.gz, LAAS-CNRS, Toulouse, Janvier 1995.

[Vazquez 96] F. Vazquez. An Algebra Approach to the Deduction of Data Flow Diagrams and Object Oriented Diagrams from Algebraic Specifications. *ACM SIGSOFT - Software Engineering Notes*, Vol. 21, n^o 4, pages 71–80, Juillet 1996.

[Ward 86] P. T. Ward. The transformation Schema: An Extention of the Data Flow Diagram to Represent Control and Timing. *IEEE Transactions on software Engineering*, Vol. SE-12, n^o 2, pages 198–210, Février 1986.

[Ward 89] P. T. Ward. How to integrate object orientation with structured analysis and design. *IEEE Software*, Vol. VI, n^o 2, pages 74–82, Mars 1989.

[Ward et Mellor 85] P. T. Ward et S. J. Mellor. *Structured Development for Real-Time Systems*, vol. 1,2 et 3. Yourdon Press, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[**Wasserman *et al.* 90**] A. I. Wasserman, P. A. Pircher, et R. J. Muller. The Object-Oriented Structured Design Notation for Software Design Representation. *IEEE Computer*, Vol. 23, n° 3, pages 50-63, Mars 1990.

[**Wood et Wood 89**] D. P. Wood et W. G. Wood. Comparative Evaluations of Specification Methods for Real-Time Systems. Rapport, SEI, CMU, Décembre 1989.

[**Zuberek et Bluemke 96**] W. M. Zuberek et I. Bluemke. Hierarchies of Place/Transition Refinements in Petri Nets. *Proceedings of the 5th IEEE International Conference on Emerging Technologies and Factory Automation ETFA'96*, pages 355-360, Kauai, Hawaï, 18-21 Novembre 1996.

Analyse et Conception des Systèmes Temps Réel : Translation d'une Approche Fonctionnelle à une Approche Orientée Objet

Résumé :

Les travaux développés dans ce mémoire se situent dans le cadre de l'analyse et de la conception des systèmes temps réel. Nous proposons une approche d'aide à la translation entre analyse fonctionnelle et analyse orientée objet. L'utilité d'une telle démarche ainsi que les différents travaux effectués dans ce domaine sont d'abord étudiés. Ensuite, nous exposons les différentes étapes de la démarche. Son originalité découle de l'utilisation des réseaux de Petri comme moyen de translation. D'abord, les spécifications fonctionnelles (de type SA-RT) sont décrites par réseaux de Petri. Les composantes conservatives de ces derniers sont alors utilisées pour identifier et définir les objets. Le formalisme HOOD/PNO est adopté pour la spécification des objets. L'approche proposée peut s'appuyer sur l'aspect statique des spécifications ou sur l'aspect dynamique. Il est également possible d'adopter une approche mixte. Les différentes étapes de la démarche et les approches possibles sont illustrées par des exemples significatifs. Nous étudions également les limites de la démarche proposée, ses parties automatisables et la possibilité de l'utiliser avec d'autres techniques. Par ailleurs, nous nous intéressons à la possibilité d'exploiter les résultats obtenus dans des objectifs d'évaluation des spécifications fonctionnelles. L'apport de certaines extensions temporelles des réseaux de Petri est mis en évidence.

Mots clefs : Systèmes Temps Réel, Spécification, Approches Fonctionnelles, Approches Orientées Objet, Translation, Migration, SA-RT, HOOD/PNO, Réseaux de Petri.

Analysis and Design of Real-Time Systems : Translation of Structured Analysis to Object-Oriented Analysis

Abstract :

The work presented deals with the analysis and design of real-time systems. We present an original technique for the translation of structured specifications to object-oriented specifications. The need for such a technique and related literature are first presented. Then, are presented the different steps of the proposed technique. Its originality comes from the use of Petri nets as a translation medium. First, structured specifications (SA-RT formalism) are described by means of Petri nets. Then, resulting Petri nets are decomposed by using S-invariants to identify and define objects. Resulting objects are specified according to HOOD/PNO formalism. The proposed technique can be based on the static view of the specifications, the dynamic view or can be balanced between both of them. Several examples illustrate the different steps and possible approaches. Limits of the proposed technique, possibilities to use it jointly with others and to systematize it are also studied. Moreover, we investigate the possibility to use the obtained results to evaluate structured specifications. Some time extensions of Petri nets are investigated for this goal.

Key words : Real-Time Systems, Specification, Structured Methodologies , Object-Oriented Methodologies, Translation, Migration, SA-RT, HOOD/PNO, Petri Nets.