TRADUCTION et ANALYSE d'un MODELE en vue d'une IMPLEMENTATION OPTIMALE

Bernard COUSIN

Pascal ESTRAILLIER

Laboratoire MASI - CNRS UA 818

Université Pierre et Marie CURIE

4, place JUSSIEU

75252 PARIS cedex 05

tél : (1) 43.36.25.25 - poste 3192

RESUME

Nous présentons une méthode originale d'**implémentation** réalisée à partir d'un modèle en réseau de Petri exprimant le schéma de contrôle de l'application.

La méthodologie repose sur la décomposition de l'implémentation en phases permettant l'intégration progressive des contraintes liées à l'application, à son environnement et au langage de programmation.

Nous introduisons une technique originale pour tendre vers une implémentation **optimale**, minimisant le nombre de tâches supportant l'exécution du l'application tout en conservant le degré de parallélismes potentiel du modèle. L'idée majeure consiste à réutiliser les invariants produits par l'étape de validation du modèle pour déterminer le nombre minimal de tâches. Cette technique permet la caractérisation des types d'objets utilisés par l'implémentation.

ABSTRACT

We propose a new methodology for the **implementation of parallel applications** modelized by **Petri nets.** The formal model describes the control scheme of the application.

Our methodology organizes the implementation into 3 steps that allow the progressive integration of the constraints of the application, the environment, and of the programming language.

We introduce a new technique to tend towards an **optimal implementation** that minimizes the number of processes. This set of processes, that support the execution of the application, must keep the intrisic parallelism of the model. The major idea is to reuse the **linear invariants** given by the validation of the model. These invariants allow us to define the optimal process number. This technique characterizes the objects used by the implementation steps.

PLAN

MOTS-CLEFS

Implémentation, Système parallèle, Modèle formel, Réseau de Petri, Génération automatique, Invariant.

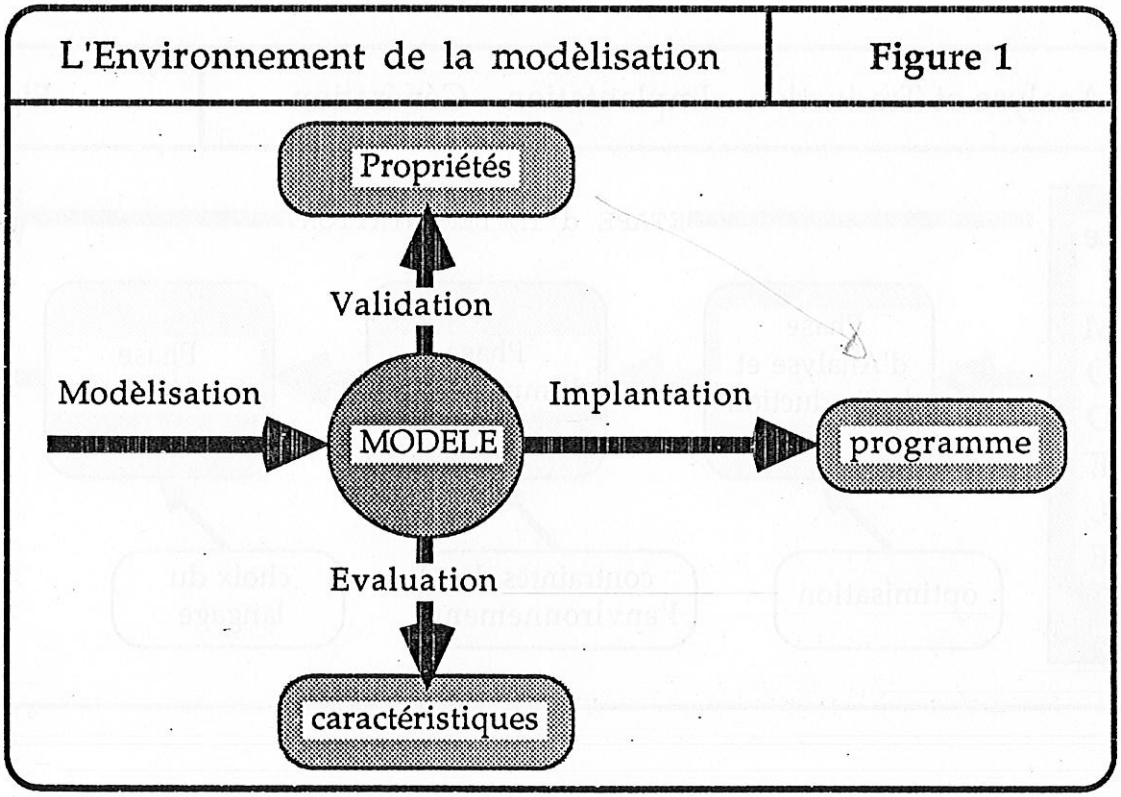
1. Introduction

Il n'est plus besoin aujourd'hui de prêcher les avantages qui découlent de l'utilisation d'un modèle formel lors de la conception d'applications informatiques complexes qui mettent en oeuvre des techniques réellement parallèles (protocoles de communication, ateliers flexibles, ...).

A partir de la spécification précise des caractéristiques et des contraintes liées à l'application, l'étape de **modélisation** construit un modèle formel. Soutenu par un corpus de résultats mathématiques rigoureux, ce type de modèle permet, avant que l'application ne soit réalisée, d'en étudier les caractéristiques et les dimensionnements (disfonctionnement, charge, délai, etc...). Il peut de plus être exploité de manières très diverses (Figure 1) :

- L'étape de **validation** de modèles formels permet d'obtenir l'ensemble des propriétés qualitatives vérifiées par le modèle, et partant de contrôler la conception de l'application modélisée [Cousin 87].
- L'étape d'**évaluation** constitue la partie quantitative de l'exploitation des modèles formels. Elle permet de s'assurer du respect des caractéristiques du cahier des charges (débit, délai, etc...)et de vérifier si le dimensionnement prévu pour l'application est correct.
- L'étape d'**implémentation** de l'application, c'est-à-dire de sa programmation, est le prolongement logique du travail effectué par la modélisation. Le modèle est alors utilisé comme base de spécification (éventuellement validée à priori) permettant d'obtenir le produit correspondant.

L'Environnement de la modèlisation — **Figure 1**



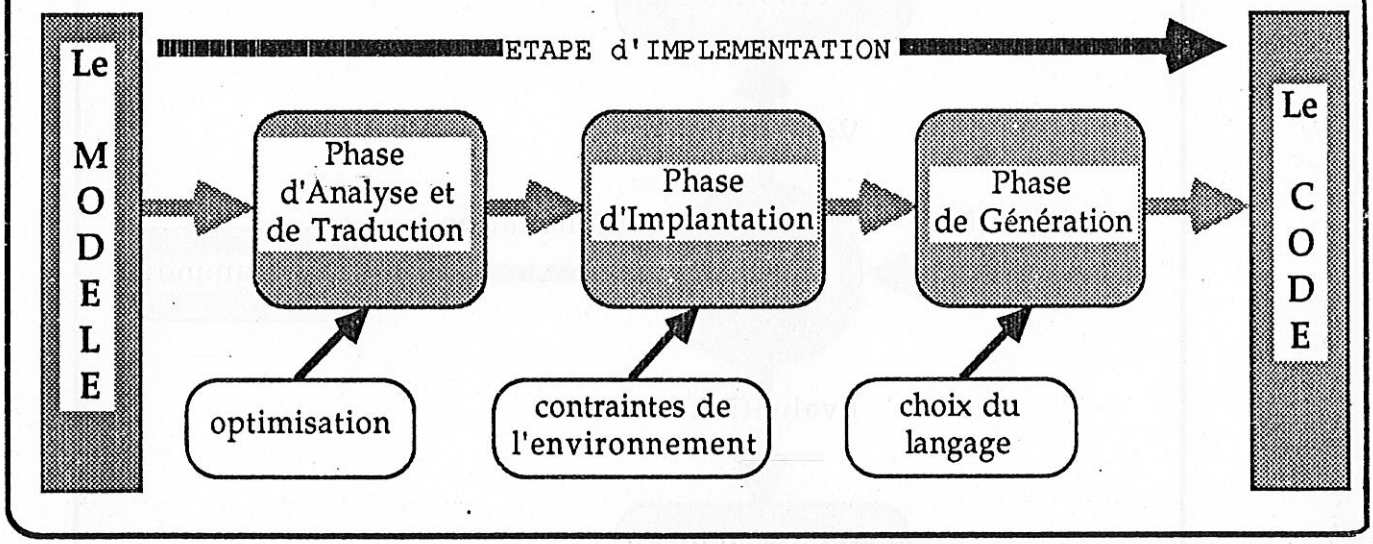
exécuter, et la gestion des éventuels conflits.
L'idée originale de notre solution consiste à utiliser les propriétés (flots) résultant de la validation du modèle.

- La deuxième phase met en oeuvre l'**Implantation** des objets caractérisés en intégrant les contraintes matérielles et logicielles d'un environnement donné. On doit disposer d'informations sur ces contraintes. En particulier, si on ne dispose que d'un nombre limité de processeurs; s'ils sont reliés entre eux par des voies spécifiques; si certaines données ou certaines opérations doivent être implantées sur des sites précis, etc... Cette phase associe les objets caractérisés aux unités matérielles ou logicielles réelles du système.

- La troisième phase **génère un code** source dépendant du langage de programmation choisi. De nouvelles contraintes sont ajoutées par le langage lui même. En effet, il est nécessaire d'exprimer fidèlement, à l'aide du langage de programmation, les règles sémantique de fonctionnement des réseaux de Petri. En particulier, les conditions de franchissement des transitions dans le Réseau de Petri ne sont pas directement implémentables dans un langage de programmation, même sophistiqué comme ADA et possédant des mécanismes proches comme le "rendez-vous".

Grâce à la structuration en trois phases distinctes, on note que la première phase est exclusivement liée à la sémantique du modèle de l'application et à l'expression des synchronisations internes; et qu'à aucun instant on ne prend en compte les choix d'implantation liés aux contraintes de l'environnement introduits dans la seconde phase. De plus, les deux premières étapes sont indépendantes du langage de programmation choisi puisque ce choix est repoussé à la dernière phase de génération. Ce choix peut être effectué parmi l'ensemble des langages concurrents de haut niveau (ADA, OCCAM, CHILL, etc...).

Analyse et Traduction - Implantation - Génération — **Figure 2**



L'objet de cet article est de présenter les fondements d'une méthode originale permettant l'**implémentation** d'une application à partir de son modèle [Cousin 88]. Nous décrivons la démarche de caractérisation des différents objets de l'implémentation à partir de la **traduction** et de **l'analyse** du modèle, notamment la détermination des tâches devant être exécutées.

Notre contribution réside dans l'introduction de trois idées novatrices :
. L'intégration progressive des contraintes liées à l'application, à son environnement et au langage de programmation.
. L'utilisation des propriétés produites par les étapes de validation et d'évaluation du modèle pour caractériser les objets.
. La définition exhaustive des types d'objets utilisés par l'implémentation.

Notre étude repose sur l'utilisation de la théorie des **Réseaux de Petri** (RdP) pour la spécification des applications par des modèles formels. En effet, cette théorie a démontré son adéquation à traiter l'ensemble des problèmes soulevés par les applications parallèles [Ayache 85]. De plus, on observe actuellement une augmentation spectaculaire, d'une part de la puissance d'expression des modèles offert par la concision des Réseaux de Petri colorés [Jensen 81] et, d'autre part, du nombre de résultats théoriques [Haddad 87].

L'essor de la théorie et l'accroissement de ses champs d'applications ont conduit à la mise en oeuvre d'ateliers logiciel comportant un ensemble d'outils développés pour assurer une modélisation aisée, la validation des modèles obtenus et leur évaluation [Bernard 88]. Il est alors possible de compléter de tels ateliers en y intégrant des outils d'implémentation et ainsi contribuer à la démonstration de l'apport des Réseaux de Petri dans la conception d'applications complexes et parallèles.

2. Méthodologie d'Implémentation

Contrairement à de nombreux autres travaux [Shatz 85], nous nous intéressons ici à produire du code à partir d'un modèle spécifiant le comportement de l'application à étudier, et non l'inverse ! A nos yeux, la modélisation est une étape préliminaire au codage, accompagnant les étapes de conception et de spécification et permettant de les contrôler. Une fois le codage réalisé, il est toujours coûteux et souvent vain de vouloir le remettre en cause (c'est trop tard !).

Nous discernons trois phases à enchaîner successivement pour générer un code à partir de la spécification d'un modèle formel (Figure 2) :

- La première phase de **Traduction et d'Analyse** permet d'exploiter le modèle en associant le schéma de contrôle (le RdP) au schéma d'interprétation (les opérations, le code). Le schéma de contrôle se charge spécifiquement des synchronisations et de l'ordonnancement des opérations; opérations que le schéma d'interprétation a pour charge d'exécuter. Cette phase permet de caractériser les objets, qui prennent en charge l'ensemble des opérations à

3. Traduction et Analyse

3.1. Introduction

La structuration en 3 phases nous permet, donc, de nous dégager des contraintes d'environnement et de programmation. C'est pourquoi la machine caractérisant les objets est une **machine virtuelle**, sur laquelle il n'a pas de contraintes matérielles ou logicielles. Par exemple, n'importe quel parallélisme potentiel est atteignable (le nombre de processeurs de cette machine est infini).

La phase d'analyse et de traduction étant la première de l'étape d'implémentation, elle repose, elle aussi, sur un modèle formel à propos duquel nous effectuons les hypothèses suivantes :

- Le modèle représente de manière adéquate l'application à réaliser. La conception de l'application a été vérifiée à l'aide de la validation et de l'évaluation du modèle.
- Le modèle comporte de manière explicite toutes les synchronisations, toutes les précédences, toutes les exclusions nécessaires au fonctionnement de l'application.

3.2 Problèmes

Afin de présenter clairement nos différentes techniques de traduction et d'analyse utilisées pour définir les objets à analyser, il est nécessaire d'introduire les différents problèmes posés par l'implémentation des modèles formels décrivant des application parallèles : la gestion des conflits et le degré de parallélisme.

La gestion des conflits

Pour maintenir la cohérence avec le Réseau de Petri qui stipule que le franchissement d'une transition est indivisible, il est nécessaire de résoudre les problèmes d'accès aux ressources partagées (les places et des transitions du modèle) entre plusieurs tâches potentiellement actives simultanément.

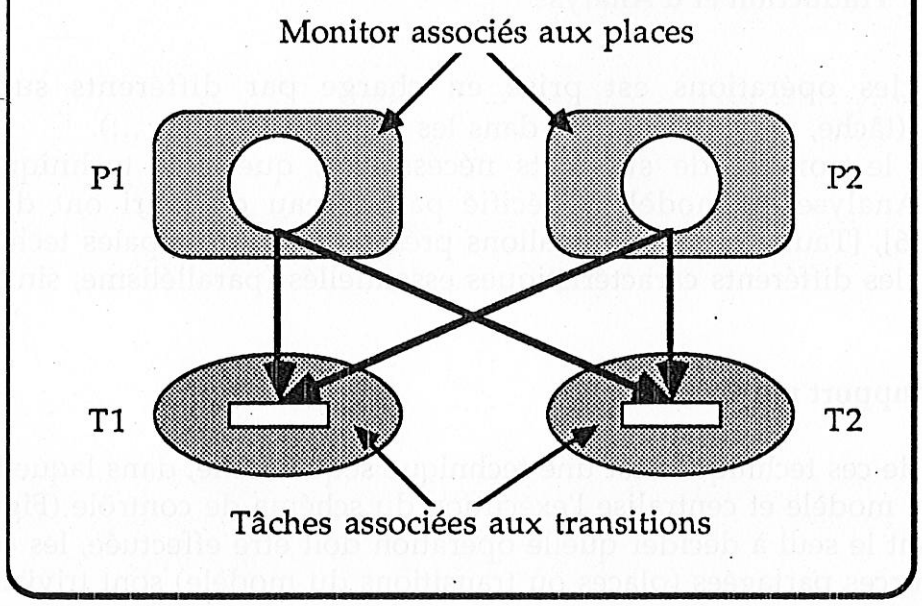
La solution de ces conflits consiste permettre de modifier le marquage de chaque place du modèle de manière indivisible.le code accédant ou modifiant une place est considéré comme une ressource protégée par un "monitor" au sens de Hoare). Toutefois, l'application de cette technique risque de faire apparaître des interblocages.

La gestion des interblocages

Les interblocages découlent naturellement des contraintes d'intégrité nécessaires aux résolutions des conflits d'accès aux ressources partagées. Une transition doit, pour être franchie, vérifier ses conditions de déclenchement, c'est à dire obtenir une marque de chaque place liée à un arc d'entrée de la transition. Les accès aux ressources étant bloquants, l'interblocage se produit si par exemple deux transitions, partageant les deux mêmes places, veulent accéder simultanément à la

marque que chacun d'elle contient (Figure 3). Chacune des transitions accapare une marque et se met en attente (infinie !) de la deuxième marque que l'autre possède.

Places -Transitions / Tâches - monitors **Figure 3**



Les interblocages nécessitent la mise en oeuvre des mécanismes similaires à ceux employés dans le domaine des Systèmes de Base de Données Réparties [Rosenkrantz 78]. Les actions nécessaires au déclenchement d'une transition doivent alors constituer une unité de traitement atomique (une transaction) et des mécanismes, tels que l'estampillage ou le verrouillage à deux phases, permettent d'assurer la cohérence nécessaire.

De tels mécanismes permettent cependant uniquement de conserver la sémantique de fonctionnement du schéma de contrôle associé au modèle de Réseau de Petri. Ils ne sont pas prévus pour traiter les interblocages génèrés par l'application elle-même telle qu'elle est modélisée, au contraire ils seront fidèlement implantés !

Le degré de parallélisme

Dans les applications complexes, éventuellement réparties, la performance est souvent directement liée au degré de parallélisme, c'est-à-dire à la capacité de la prise en charge de l'exécution simultanée de plusieurs opérations.

Le parallélisme du modèle est qualifié de **potentiel**; il fait référence aux opérations pouvant comportementalement s'exécuter en parallèle. Naturellement, il tient compte des synchronisations devant être mise-en-oeuvre pour gérer les contraintes de précédence et les conflits d'accès aux ressources partagées.

Le degré de parallélisme d'un support d'implémentation est naturellement limité à celui du modèle. Si le modèle ne comporte aucun degré de parallélisme (le modèle de l'application est purement séquentielle), le support de l'implémentation

sera restreint à une seule tâche.

L'objectif de notre travail est de faire tendre le degré de parallélisme de l'implémentation vers le degré potentiel du modèle (cf. § 4).

3.2. Techniques de Traduction et d'Analyse

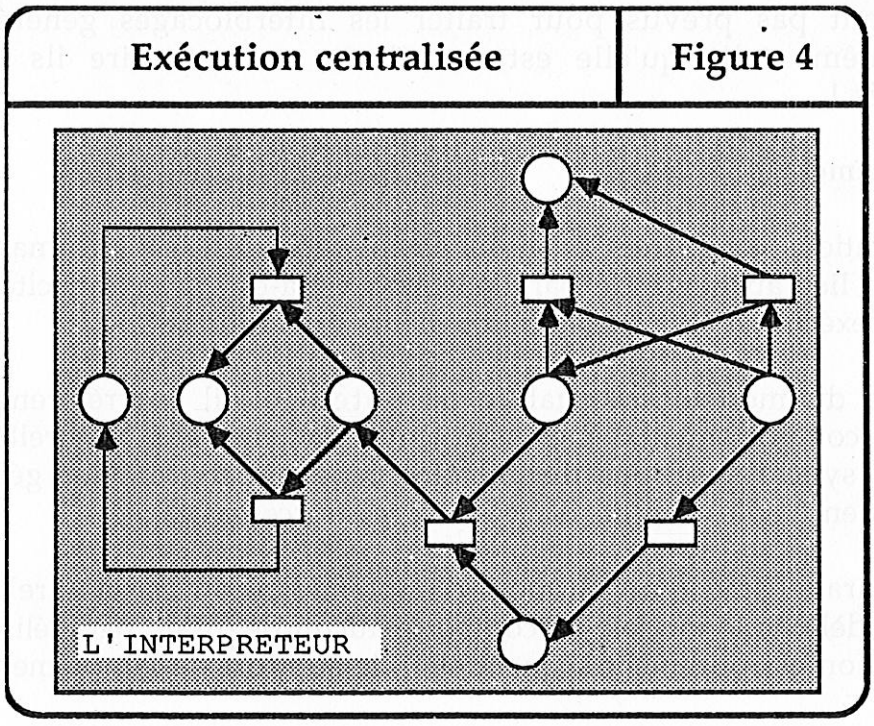
L'exécution des opérations est prise en charge par différents supports d'implémentation (tâche, unité mécanique dans les ateliers flexibles, ...).

Pour définir le nombre de supports nécessaires, quelques techniques de Traduction et d'Analyse de modèles spécifié par Réseau de Petri ont déjà été publiées [Bruno 86], [Taubner 87]. Nous allons présenter les principales techniques afin d'en dégager les différents caractéristiques essentielles (parallélisme, simplicité, etc...).

Technique A : à support minimal

La première de ces techniques est une technique séquentielle, dans laquelle une **tâche** interprète le modèle et centralise l'exécution du schéma de contrôle (Figure 4). L'interpréteur étant le seul à décider quelle opération doit être effectuée, les conflits d'accès aux ressources partagées (places ou transitions du modèle) sont trivialement résolus.

Cette technique convient parfaitement à des solutions centralisées et séquentielles dans lesquelles une opération au plus est exécutée simultanément. Son avantage primordial réside dans sa simplicité de mise en oeuvre. Néanmoins elle possède l'inconvénient majeur de ne pas permettre au modèle de bénéficier de la puissance de traitement des machines parallèles ou réparties. Or c'est justement l'une des justifications de l'utilisation de notre modèle formel (les RdP) que de gérer les applications distribuées.



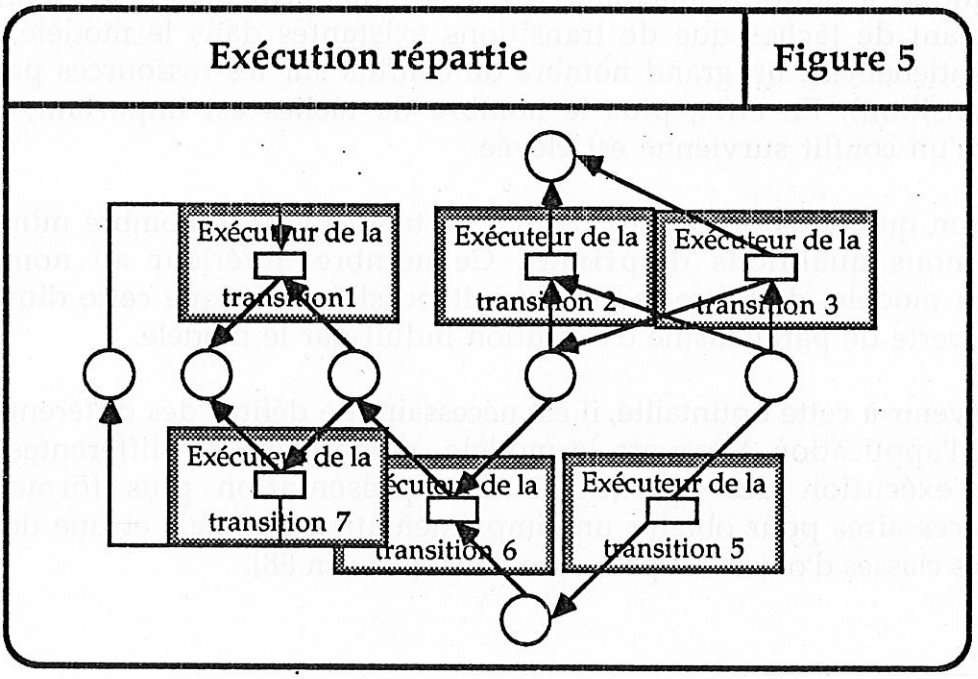
En résumé, cette technique est fondamentalement :
- simple (en conception et en implantation);
- inefficace (en temps d'exécution et en regard du critère de distribution).

Technique B : à support maximal

La deuxième technique est située à l'autre extrémité du spectre des implantations possibles. Elle consiste associer **une tâche par transition du modèle.** C'est le support d'implémentation maximale possible. Cette tâche a pour charge d'attendre que les conditions de franchissement de la transition associée soient remplies pour lancer l'exécution de l'opération correspondante (Figure 5).

Cette technique permet toujours d'atteindre le parallélisme potentiel induit par le schéma de contrôle, et donc de bénéficier pleinement de la puissance. Cependant, la résolution des problèmes d'accès aux ressources partagées (les places et les transitions du modèle) entre plusieurs tâches potentiellement actives simultanément est non triviale. Elle nécessite la mise en oeuvre une des solutions proposées dans le paragraphe précédent, pour résoudre les conflits d'accès et les interblocages. Ce nombre de conflits étant directement dépendant du nombre de tâches (de transitions), il est ici maximal !

En résumé, cette technique comporte les caractéristiques suivantes :
- complexe au niveau de la gestion des conflits d'accès aux places et aux transitions partagées.
- efficace dans la mesure où le parallélisme potentiel est atteint;



Technique C : à support optimal

Quelques travaux récents proposent des variantes des deux techniques précédentes. Elles ont généralement pour but de minimiser le nombre de conflits d'accès aux ressources partagées, tout en conservant un degré de parallélisme intéressant.

Dans [Colom 86], la notion de "local synchroniser" est introduite. Elle regroupe les places et les transitions partagées pour faciliter leur traitement, et permet résoudre trivialement le traitement des places et des transitions locales.

En prolongeant de cette même idée, nous proposons dans le chapitre suivant une méthode minimisant le nombre de tâches nécessaires au fonctionnement de l'application, sans toutefois diminuer le parallélisme potentiel du modèle. Pour cela, on associe **une tâche par ensemble de transitions** devant être exécutées séquentiellement. Il en découle une diminution du nombre de tâches, une simplification et une réduction du nombre de conflits à gérer.

L'optimalité de l'implémentation proposée est obtenue en minimisant le nombre de tâches nécessaires à l'exécution du système. L'optimalité est atteinte si la minimisation ne provoque pas une perte du degré de parallélisme de l'implantation proposée vis à vis du parallélisme potentiel du modèle.

4. Une Technique à support Optimal

La technique à support maximal (B) a comme principal inconvénient de nécessiter autant de tâches que de transitions existantes dans le modèle, ce qui entraîne potentiellement un grand nombre de conflits sur les ressources partagées (place ou transition). En effet, plus le nombre de tâches est important, plus la probabilité qu'un conflit survienne est élevée.

La solution que nous proposons permet de tendre vers le nombre minimal de tâches, que nous qualifions d'**optimal**. Ce nombre, inférieur au nombre de transitions du modèle, doit être le plus réduit possible sans que cette diminution entraîne une perte de parallélisme d'exécution induit par le modèle.

Pour parvenir à cette optimalité, il est nécessaire de définir des différents objets caractérisant l'application à travers le modèle, notamment les différentes tâches supportant l'exécution des opérations. Une présentation plus formelle des conditions nécessaires pour obtenir une implémentation optimale et une définition des différentes classes d'objets est présentée dans [Cousin 88].

4.1 Principe

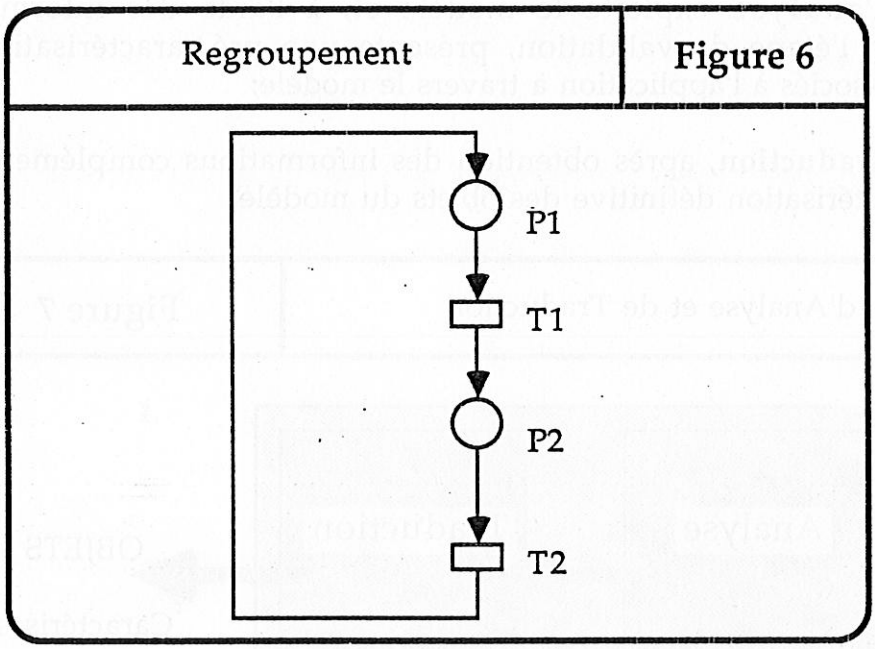
Nous proposons de regrouper plusieurs opérations qui seront gérées par une tâche unique. Pour conserver le parallélisme potentiel, le regroupement ne doit pas comporter des opérations susceptibles de s'exécuter simultanément. Les transitions regroupées sont alors en franchissement exclusif et leurs exécutions sont

séquentielles.

Un tel regroupement se justifie d'autant plus lorsque les opérations partagent des ressources. On observe alors une diminution du nombre de conflits d'accès à résoudre :

- Sur les ressources locales, une tâche accède à ses ressources de manière séquentielle (et préserve donc leur cohérence en réglant trivialement les conflits),
- Sur des ressources globales, le nombre total de tâches (potentiellement conflictuelles) étant diminué, le nombre de conflits à arbitrer est minimisé.

Notre proposition peut être illustrée par l'exemple de la figure 6. La marque occupe successivement les places P1 puis P2, en franchissant alternativement les transitions T1 et T2. Il est évident d'associer à la totalité de ce modèle une seule tâche d'exécution. Dans cette exemple, nous notons que tous les conflits d'accès aux places ont disparu par simple exécution séquentielle de la tâche associée.



4.2 Méthode

La détermination des regroupements à effectuer est un problème d'envergure lorsque des modèles quelconques doivent être examinés. Une analyse structurelle du modèle s'avère nécessaire pour pouvoir détecter les regroupements les plus performants.

Une solution [Colom 86] consiste à étudier, transition par transition et place par place, si les conditions requises pour le regroupement sont obtenues. Cette solution s'avère coûteuse et surtout ne prend en compte tous les conflits: elle élimine ceux qui sont structurels sans être effectifs.

Plutôt que de devoir procéder à cette analyse structurelle, il nous a semblé naturel d'utiliser les invariants produits par l'étape de validation du modèle. Chaque invariant du modèle constitue une composante conservative du système. On peut alors potentiellement lui associer une tâche.

Les places servant de support à un invariant constituent l'ensemble des états de

la tâche associée et les transitions reliant ces places représentent alors les opérations de cette tâche.

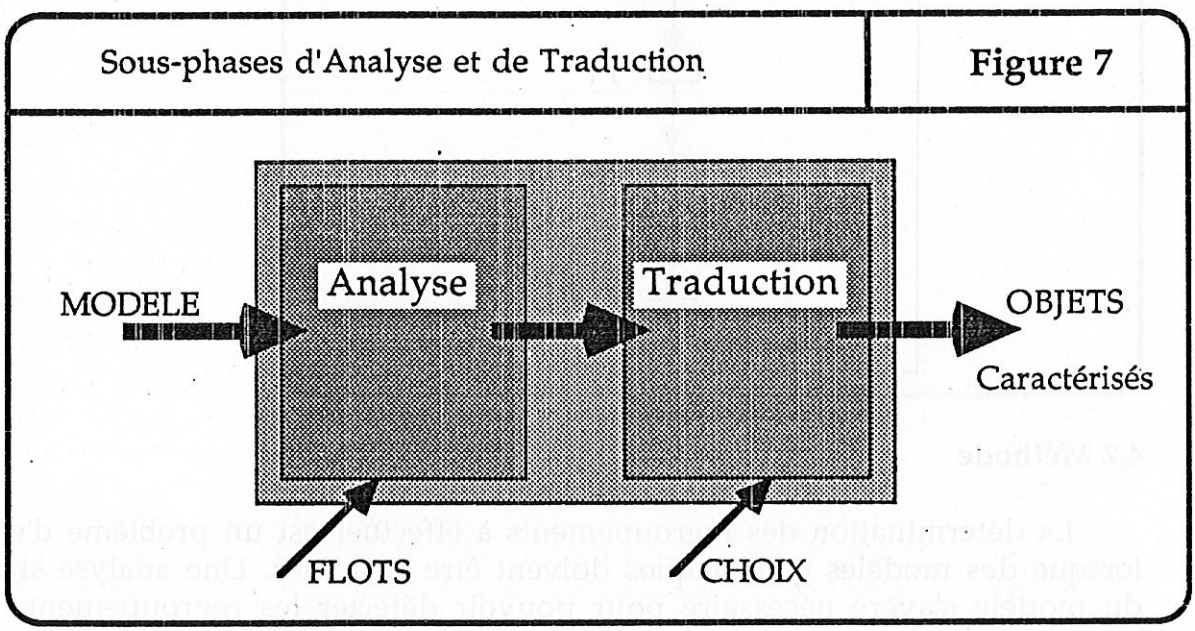
Un obstacle apparaît immédiatement : c'est la présence d'une même place parmi plusieurs flots.

Une solution peut y être apportée. Cette solution consiste simplement à éliminer un des flots, ce qui résout radicalement le problème soulevé. Néanmoins, il convient de déterminer le flot à éliminer. Ce choix n'est pas aisé à effectuer; il dépend de l'interprétation du modèle et de l'environnement de l'implémentation.

Il est clair que l'analyse du modèle, même bénéficiant des résultats de la validation ne suffit pas à la caractérisation des objets. Il est alors nécessaire d'obtenir des informations supplémentaires. C'est la raison de la structuration de la phase **d'analyse et de traduction** en deux sous-phases (Figure 7):

- La sous-phase d'**analyse** exploite le modèle et, à l'aide des informations disponibles après l'étape de validation, présente une pré-caractérisation des différents objets associés à l'application à travers le modèle;

- La sous-phase de **traduction**, après obtention des informations complémentaires, présente une caractérisation définitive des objets du modèle.



4.21. Sous-phase d'Analyse

Des techniques numériques, basées sur la résolution d'équations linéaires, permettent d'obtenir une base d'invariants linéaires d'un modèle quelconque décrit par Réseau de Petri [Memmi 82].

Les flots obtenus sont des combinaisons linéaires du marquage de certaine places. Ils sont égaux à la constante formée sur le marquage initial de la combinaison linéaire de ces mêmes places. Deux classes disjointes de flots sont discernées :

- Les flots positifs ne comportent que des coefficients de même signe.
- Les flots négatifs comportent des coefficients de signe opposé.

Nous proposons l'interprétation suivante de ces 2 classes:

- Un flot positif décrit l'ensemble des états d'une tâche. La constante représente le nombre d'exemplaire de tâches associés à ce flot.
- Un flot négatif décrit 2 sous-ensembles distincts de places ayant un comportement similaire. Le premier sous-ensemble est décrit par les places à coefficients négatifs, le deuxième par les places à coefficients positifs. Les 2 sous-ensembles ont un comportement similaire, car le marquage des deux sous-ensembles est toujours égal à la constante près. Si l'un des sous-ensembles est réduit à une place, cette place est dite implicite, car le marquage du sous-ensemble détermine le marquage de la place.

Il existe des catégories de réseaux (machines à états, graphes d'évènements, etc... [Hack 73]) qui possèdent intrinsèquement l'ensemble des caractéristiques nécessaires et pour lesquels l'analyse conduit facilement à la détermination des objets.

Malheureusement, il existe de nombreux modèles qui ne possèdent pas un ensemble adéquat d'invariants (voire aucun !). De tels modèles peuvent cependant toujours s'implémenter selon la technique à support maximal (une tâche par transition, cf les transitions isolées).

En général, l'expérience tend à confirmer que, dans les applications réelles, la validation du modèle permet souvent d'obtenir un ensemble intéressant d'invariants.

4.22. Sous-phase de Traduction

Pour compléter l'analyse et caractériser définitivement les objets, nous proposons 3 méthodes non-exclusives permettant d'obtenir les informations nécessaires:

- Une méthode automatique et heuristique fondée sur les faits les plus probables (une structure particulière modélise souvent un mécanisme déterminé), ou les gains les plus importants (éliminer le plus petit flots permet en général de créer moins de tâches). L'application performante de cette technique repose essentiellement sur l'expertise des règles d'implémentation.

- Une méthode interactive dans laquelle le concepteur est interrogé sur ses propres choix. En la combinant avec la méthode précédente, nous pouvons présenter au concepteur parmi l'ensemble des choix possibles, celui qui semble judicieux. Le concepteur se contente alors de valider l'un deux.

- Une méthode "à-priori", dans laquelle le concepteur a déjà précisé ses décisions durant la définition du modèle (ainsi les notions de places-ressources ou de places-tâches peuvent figurer dans le modèle). Néanmoins, nous pensons que cette dernière méthode peut être contraignante et dangereuse, le concepteur faisant éventuellement des choix non optimaux dans l'ignorance de l'exact comportement de son système. Il convient de s'en assurer à l'aide des deux méthodes précédentes.

4.3. Caractérisation des objets

La phase d'analyse et de traduction permet de présenter une caractérisation des types d'objets suivants :

Tâche :

Potentiellement, la phase d'analyse associe une tâche à chaque flots du modèle. Les places formant le flot constitue le support des tâches.
Dans la figure 8, il a potentiellement 3 tâches car 3 flots :
Pc0+Pc1=1 ; Pp0+Pp1=1 ; et Pt0+Pt1+Pt2=1.

Transition :

Une transition (l'opération associée) appartient à un flot (à une tâche), si la transition est parcourue par le flot. La transition possède alors au moins un arc entrant et un arc sortant vers des places de ce flot.
Dans la figure 8:

Les transitions Tc, Tc1, Tc2 appartiennent à la tâche du premier flot.
Les transitions Tp, Tp1, Tp2 appartiennent à la tâche du deuxième flot.
Les transitions Tp1, Tp2, Tc1, Tc2 appartiennent à la tâche du troisième flot.

Transition partagée :

Les transitions partagées font simultanément partie d'ensembles de transitions reliant les places appartenant à des tâches distinctes.
Dans la figure 8, les transitions Tp1, Tp2, Tc1, Tc2 sont des transitions partagées.

Elles peuvent être implantées sous la forme de rendez-vous entre les tâches. Par exemple, dans la figure 9, la transition T est partagée entre des deux tâches formées sur les invariants suivants :

$F1 = P1 + P2$ et $F2 = P3 + P4 + P5$

provoquent une synchronisation par rendez-vous entre les deux tâches.

L'opération associée à une transition partagée peut être effectivement partagé. Ceci est possible, car les contraintes de modélisation produisent naturellement un code réentrant :

- Les variables globales (sur lesquelles portent les synchronisations) sont externes au code (figurent explicitement dans la partie de contrôle du modèle);
- Les variables locales sont internes au code (n'apparaissent pas dans la partie de contrôle).

Donc le code de l'opération associé à une transition partagée appartient à toutes les tâches qui partagent la transition. Dans le cas d'une transition de synchronisation le code n'est exécuté qu'une fois pour chaque synchronisation (par définition du synchronisme). Dans le cas d'une transition commune à plusieurs tâches : chaque tâche a la propre maîtrise de son exécution. Le code est réentrant et l'exécution des différentes tâches est asynchrone (Figure 10).

Transition isolée :

Une transition est isolée si elle n'appartient à aucune tâche (aucun flot la parcourant n'a été retenu). Alors, aucun de ses arcs n'est relié à une place constituant le support d'une tâche.

A toute transition isolée, nous associons une tâche (Figure 11). Le support de cette tâche peut être constitué d'une place implicite bouclant sur la transition isolée. L'ajout de cette place ne modifie en rien le comportement du modèle (par construction puisqu'elle est implicite). Cette place exhibe le comportement de la tâche associée; elle teste en permanence la franchissabilité de la transition.

Flot-message

Un flot est potentiellement un flot-message, si toutes ses transitions sont des transitions partagées. L'ensemble potentiel de ses opérations peut être exécuté par les tâches qui partagent chacune de ses transitions. Il est donc inutile de leur attribuer une tâche d'exécution, ce qui n'aurait pour effet que de compliquer la synchronisation nécessaire sur les transitions partagées. Les flots conservés deviennent des **Flot-tâches**.

Ce cas est illustré dans la figure 8 par le troisième flot et ses quatre transitions qui sont deux à deux partagées par les deux autres tâches.

Cette technique est naturelle si on s'aperçoit que la figure 8 modèlise un producteur (places Pp0 et Pp1) /consommateur (places Pc0 et Pc1) communiquant à travers un double tampons (places Pt0 tampon vide/Pt1 un tampon plein/ Pt2 les deux tampons pleins).

Flot parallèles

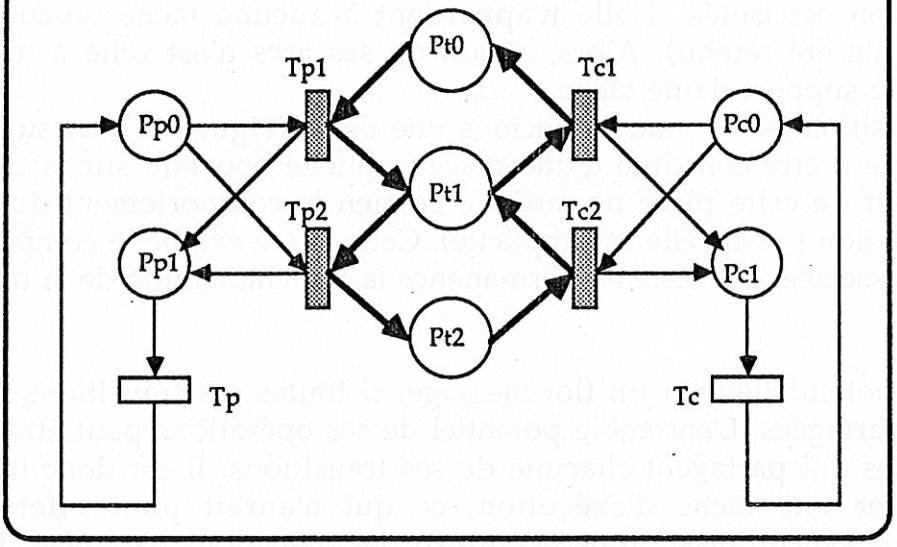
Les flots parallèles sont des flots ayant comme support un sous-ensemble de place identiques.

Ressources

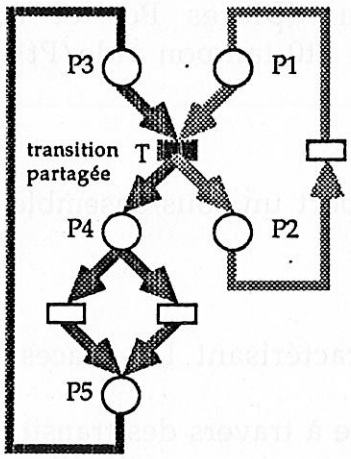
Nous définissons deux classes permettant de caractérisant les places du modèle:

- Les places locales ne sont reliés qu'à une seule tâche à travers des transitions non-partagées; On ne leur associe donc pas une structure particulière de contrôle.
- Les places globales ont un marquage modifiées par plus d'une tâche. Ils convient de les protéger pour assurer leur intégrité.

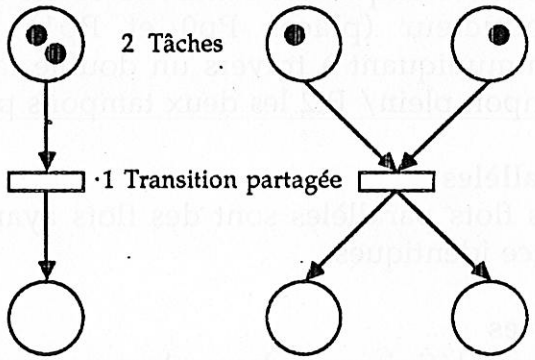
Un Flot message : 4 transitions partagées — Figure 8



Synchronisation entre tâches — Figure 9



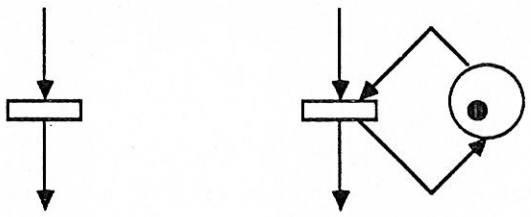
Transitions partagées — Figure 10



2 exécutions :
1 exécution asynchrone
pour chacune des 2 tâches
(2 franchissements)

1 exécution :
1 seule exécution
synchrone des 2 tâches
(1 franchissement)

Transition isolée : une place implicite — Figure 11



5. Conclusion

Nous avons étudié dans cet article l'implémentation des systèmes parallèles spécifiés par des modèles utilisant le réseau de Pétri pour schéma de contrôle. Cette étape d'implantation est située en aval des étapes de modélisation et de validation. Elle est constituée de trois phases : Analyse et Traduction, Implantation et Génération.

Grace à cette structuration, la première phase est libérée des contraintes de l'environnement et du langage d'implémentation. Elle permet d'optimiser le nombre de tâches nécessaires à l'exécution du système, ce qui contribue à la diminution de la complexité de l'implémentation.

L'étude structurelle du modèle, concrétisée par les flots trouvés durant l'étape de validation, apporte les informations nécessaires pour atteindre le nombre optimal de tâches.

Cette étude originale nous permet de faire apparaître de manière systématique les caractéristiques des objets composant un modèle donnée (les tâches et leurs états, les messages et ses tampons, les synchronisations par rendez-vous, les ressources locales ou partagées, etc...).

Cette caractérisation des objets du modèle permet d'optimiser et de faciliter les deux phases suivantes, pour aboutir finalement à un code adéquat du système.

REMERCIEMENTS

Nous tenons tout particulièrement à remercier nos collègues de l'université de Saragosse qui nous ont permis, durant un trop bref mais fructueux séjour dans leur locaux, grâce à leurs travaux antérieurs et à leur nombreuses discussions, de faire aboutir notre recherche sur ce sujet.

BIBLIOGRAPHIE

[Ayache 85] J.M.Ayache, J.P.Courtiat, M.Diaz, G.Juanole, *Utilisation des réseaux de Petri pour la modélisation et la validation de protocole,* TSI vol 4 n°1 pp51, janvier 1985.

[Bernard 88] J.M.Bernard, J.L.Mounier, N.Beldiceanu, S.Haddad, *Un Atelier de Modélisation Interactif,* rapport MASI n°211, 1988.

[Bruno 86] G.Bruno, G.Marchetto, Process-translatable Petri nets fot the rapid prototyping of process control systems, IEEE transaction on Software Engineering, vol 12 pp346, February 1986.

[Colom 86] J.M.Colom, M.Silva, J.L.Villarroel, *On software implementation of Petri nets and colored Petri nets using high-level concurent languages,* Proceding on 7th Petri nets Workshop, Oxford - England, 1987.

[Cousin 87] B.Cousin, *Méthodologie de validation des systèmes structurés en couches,* Thése de l'universite Pierre et Marie Curie, Avril 1987.

[Cousin 88] B.Cousin,P.Estraillier *Le projet TAGADA : Traduction, Analyse et Génération de code ADA* Rapport MASI à paraitre - janvier 1988.

[Hack 73] M.Hack, *Extended State-Machine Allocatable Nets an extension of free-choice Petri nets results* M.I.T Cambridge Mass. Project MAC. MAC-TR 94 (Septembre 1972)

[Haddad 87] S.Haddad, *Une catégorie régulière de réseau de Petri de haut-niveau: définition, propriétés et réduction,* Thése de l'universite Pierre et Marie Curie, Juin 1987.

[Jensen 81] K.Jensen, *Coloured Petri nets and the Invariant method,* Theorical Computer science, vol14 n°3, June 1981.

[Memmi 83] G.Memmi, *Méthode d'analyse de réseaux de Petri, réseaux à files ,* et applications aux systèmes temps réels Thése de doctorat d'état,l Université Pierre et Marie Curie, Juin 1983.

[Taubner 87] D.Taubner, *On the implementation of Petri nets,* 8th European Workshop on application and theory of Petri nets, Zaragoza - Espagne, juin 1987.

[Rosenkrantz78] D.J.Rosenkrantz, R.E.Stearns, P.W.Lewis, *System level concurrency control for distrubuted database systems,* ACM Transactions on Database Systems, vol 2 n°3 pp233, Sept. 1979.

[Shatz 85] S.M.Shatz, W.K.Cheng , *An appoach to automated static analysis of distributed software,* Proc. of the first international conference on Supercomputing Systems, pp 377, December 1985.