Méthodologie de développement de protocoles de communication et des applications réparties. Vers une approche de synthèse

Hakim Kahlouche

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

Numéro d'ordre : 160 ID

THESE

Présentée par

Hakim KAHLOUCHE

Pour obtenir le titre de

DOCTEUR en Informatique

de l'Ecole Nationale Supérieure des Mines de Saint-Etienne
et de l'Université de Saint-Etienne

Méthodologie de développement des protocoles de communication et des applications réparties. Vers une approche de synthèse

Soutenue à Saint-Etienne, le 18 Juillet 1997

Composition du Jury

Messieurs	Gérard	BERTHELOT	Président
	Claude	JARD	Rapporteurs
	Guy	JUANOLE	
	Jean-Jacques	GIRARDOT	Examinateurs
	Michel	HABIB	
Madame	Christine	LARGERON	

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

Numéro d'ordre : 160 ID

THESE

Présentée par

Hakim KAHLOUCHE

Pour obtenir le titre de

DOCTEUR en Informatique

de l'Ecole Nationale Supérieure des Mines de Saint-Etienne
et de l'Université de Saint-Etienne

Méthodologie de développement des protocoles de communication et des applications réparties. Vers une approche de synthèse

Soutenue à Saint-Etienne, le 18 Juillet 1997

Composition du Jury

Messieurs	Gérard	BERTHELOT	Président
	Claude	JARD	Rapporteurs
	Guy	JUANOLE	
	Jean-Jacques	GIRARDOT	Examinateurs
	Michel	HABIB	
Madame	Christine	LARGERON	

Remerciements

Je tiens à exprimer mes plus vifs remerciements à tous les membres du jury ainsi qu'à toutes les personnes qui, de près ou de loin, m'ont aidé durant ces années de thèse :

Monsieur Jean-Jacques Girardot, Maître de Recherche à l'Ecole Nationale Supérieure des Mines de Saint-Etienne, pour m'avoir accueilli dans son équipe et a permis la réalisation de ce travail dans les meilleures conditions. Je tiens à lui témoigner ma sincère gratitude.

Monsieur Gérard Berthelot, Professeur au Conservatoire National des Arts et Métiers, pour l'honneur qu'il me fait en présidant le jury de ma thèse.

Messieurs Claude Jard, Chargé de recherche CNRS à l'Institut de Recherche en Informatique et Systèmes Aléatoires de Rennes, et Guy Juanole, Professeur à l'Université Paul Sabatier de Toulouse, pour l'intérêt qu'ils ont porté à mon travail en acceptant la charge d'en être les rapporteurs.

Madame Christine Largeron, Maître de conférence à l'Université des Sciences de Saint-Etienne, et Monsieur Michel Habib, Professeur à l'Université des Sciences de Montpellier, pour avoir accepté de faire partie du jury.

Tous les membres du centre SIMADE (ex-département informatique), permanents et thésards, pour l'ambiance amicale qu'ils ont su créer.

Mesdames Marie-Line Barneoud, Liliane Brouillet, Zahia Mazer et Hélène Sayet pour leur disponibilité et leur gentillesse.

Le personnel du service de reprographie pour avoir assuré la reproduction de cet ouvrage.

Mes Parents, mes frères et soeurs pour leur affection et leur soutien moral indéfectible.

Table des matières

Table des figures

Liste des tableaux

Résumé

Nous proposons dans notre thèse une nouvelle méthodologie pour le développement des protocoles de communication et des applications réparties en explorant une approche de synthèse automatisée.
Après avoir effectué un état de l'art sur les techniques de synthèse de protocoles, nous présentons, dans la première partie de la thèse, une nouvelle méthode de synthèse automatique de spécifications de protocoles à partir de spécifications de services dans un modèle de réseaux de Petri interprétés. Notre approche de synthèse est basée sur un nouveau concept de raffinement de réseaux de Petri qui assure la correction des protocoles construits de façon incrémentale. L'intérêt principal de cette méthode par rapport aux techniques classiques basées sur l'analyse est qu'elle évite la phase supplémentaire de validation du protocole qui passe souvent par une analyse exhaustive dont le problème fondamental est l'explosion combinatoire. De plus, le coût de construction des protocoles est considérablement diminué. Partant d'une description de service, notre technique de synthèse permet de dériver automatiquement au niveau protocole l'ensemble des fonctionnalités suivantes : (1) traitements répartis, (2) flot de contrôle, (3) flot de données, (4) contrôle de la cohérence des données, (5) et choix distribué. Elle constitue, de ce fait, un "bon" compromis entre le pouvoir de synthèse et le pouvoir d'expression des services relativement aux techniques existantes.
Dans la deuxième partie de la thèse nous présentons une application réelle de notre méthode de synthèse à la conception du protocole de transport ISO de base (ISO 8073), le point de départ de la conception étant la spécification du service de transport ISO 8072.
Dans la troisième partie de la thèse, nous présentons un ensemble d'outils logiciels pour la synthèse automatique de protocoles que nous appelons STEPS (Software Toolset for automatEd Protocol Synthesis). Nous avons développé cet ensemble d'outils afin de démontrer l'utilité et la faisabilité de l'approche de synthèse que nous proposons.
La sémantique de notre modèle de spécification étant assez proche de celle du langage Estelle, nous proposons une extension de notre démarche pour la synthèse de spécifications de protocoles dans le langage Estelle. L'intérêt est que ces spécifications peuvent être directement exploitées par des outils existants afin de générer automatiquement du code exécutable ou d'effectuer des simulations et des évaluations de performances.

Mots clés : *Protocole de communication, service de communication, application répartie, ingénierie des protocoles, synthèse de protocoles, réseaux de Petri, vérification des protocoles.*

Introduction

L'évolution de l'informatique répartie en général et des réseaux de communication en particulier est liée d'une part au développement de protocoles de communication réalisant des services de plus en plus sophistiqués (transmission fiable, divers sortes de traitements répartis, échange de données multimédia, etc.), et d'autre part, à l'évolution technologique des supports physiques de transmission (fibres optiques, canaux satellites, etc.). Ainsi, une telle évolution a permis, entre autres, l'intégration de la voix, des données et des images dans des réseaux à haut débit tels que FDDI, DQDB et ATM.
La tâche qui consiste à développer les protocoles de communication (appelés encore partie logiciel des réseaux) apparaît donc comme une phase cruciale pour une exploitation efficace et performante des réseaux de communication. L'incapacité de contrôler la complexité et la correction de tels logiciels est particulièrement nuisible pour le développement de systèmes distribués modernes et à grande échelle. En effet, des erreurs non détectées lors de la phase de conception ou de spécification peuvent causer de grandes difficultés dans le fonctionnement et la maintenance du logiciel développé.
Bien que l'architecture multi-couches des protocoles de communication telle que celle proposée par l'ISO [ISO 7498] simplifie le développement des protocoles, chaque couche reste cependant complexe. Cette complexité est due à plusieurs facteurs:

- La nature distribuée des protocoles de communication. Il n'existe pas d'état global perceptible par une entité du système.

- Les services sous-jacents à une couche peuvent être non fiables. Cette non fiabilité devrait donc être prise en compte par la couche en question.

- Les systèmes interconnectés sont souvent hétérogènes, ce qui exige la mise au point de convertisseurs de protocoles.

- Les protocoles doivent être tolérants aux fautes. Le recouvrement des situations de panne est souvent complexe à réaliser.

La nature complexe des spécifications de protocoles favorise l'apparition d'une variété d'erreurs de conception. Ces erreurs peuvent être propagées vers la phase d'implantation si elles ne sont pas détectées et corrigées le plus tôt possible dans le processus de développement du protocole. Ces erreurs peuvent être classées en deux catégories: *les erreurs logiques* et *les erreurs sémantiques*. La correction logique d'un protocole concerne la structure logique des échanges de messages entre les entités du protocole. Parmi ces erreurs, nous pouvons citer: la réception non spécifiée signifiant que l'arrivée d'un message dans un état donné

du protocole n'a pas été prévue et donc ne déclenche aucun traitement, l'interblocage et le débordement des canaux de communication. D'autre part, un protocole possède des erreurs sémantiques s'il ne fournit pas tous les services attendus. Ce type d'erreur est spécifique au protocole.
Au vu de ces problèmes, liés d'une part à la complexité des logiciels de communication et d'autre part aux différentes erreurs qui peuvent survenir, on sent la nécessité d'une *méthode formelle* pour la production de logiciels de communication corrects. Une telle méthode devrait s'appuyer sur un *modèle de spécification formelle* et être supportée par ce qu'on appelle un *cycle de vie du logiciel.*
Une méthode formelle devrait tenir compte d'un certain nombre d'exigences afin qu'elle soit admise dans un milieu industriel [VIS 93] :

(1) Elle devrait être spécifique à un domaine d'application bien précis. Dans notre cas, il s'agit du développement des protocoles de communication et des applications réparties.

(2) Elle devrait fournir une démarche à suivre pour développer un système, en démarrant à partir d'un niveau d'abstraction élevé avec les besoins des utilisateurs et en aboutissant à un bas niveau d'abstraction avec l'implantation de la spécification. Pour être capable de supporter une telle méthodologie le modèle utilisé devrait supporter des techniques de transformations entre les différents niveaux d'abstraction.

(3) Elle devrait finalement être supportée par des outils logiciels.

A la lumière de l'étude bibliographique que nous avons effectuée au chapitre 1, nous pouvons dégager trois autres exigences qui nous semblent également essentielles :

(4) La méthode formelle devrait combiner plusieurs formalismes afin de répondre à divers besoins requis dans le cycle de vie d'un système complexe. Il est difficile, voir impossible, de trouver un formalisme servant pour toutes les étapes du cycle de vie d'un logiciel.

(5) L'intérêt de la méthode devrait être démontré sur des protocoles réels.

(6) Enfin, la méthode devrait répondre au problème fondamental de la complexité des systèmes réels.

Le but principal de notre thèse est de proposer une méthodologie formelle pour le développement des protocoles de communication et des applications réparties qui satisfasse au maximum les exigences précédemment citées. En effet, les méthodes existantes ignorent le plus souvent quelques points essentiels à savoir les exigences (2), (5) et (6) qui les rendent quasiment non applicables dans un cadre réel. Cette motivation a d'ailleurs été confirmée récemment dans [COU 96].

La réalisation des protocoles peut être considérée comme un problème technique relativement nouveau faisant partie d'une discipline distincte d'ingénierie des logiciels à savoir *l'ingénierie des protocoles* [LIU 89]. L'ingénierie des protocoles est basée sur la définition d'un cycle de vie passant par plusieurs phases : de la définition des besoins au test de l'implantation. Nous distinguons deux approches différentes dans la définition du cycle de vie :

l'approche basée sur *l'analyse* et celle basée sur la *synthèse*. Les techniques analytiques prennent en entrée les spécifications des services et du protocole et vérifient d'une part, la conformité des services rendus vis-à-vis des services attendus et d'autre part, l'absence d'erreurs logiques. Le cycle "détection d'erreurs et correction" est répété jusqu'à ce que le protocole soit valide. Ces techniques sont ainsi très coûteuses en temps. De plus, la phase d'analyse passe généralement par une exploration exhaustive de l'espace des états accessibles dont le problème fondamental est l'explosion combinatoire. Bien que l'approche analytique soit utile dans la validation de protocoles existants, elle ne fournit, néanmoins, aucune démarche pour la conception de nouveaux protocoles. La synthèse consiste à construire de manière systématique ou interactive un protocole sans erreurs partant des spécifications formelles des besoins requis et/ou d'un ensemble de modules pré-définis du protocole. L'objectif de la synthèse est d'augmenter la productivité des protocoles tout en diminuant le coût de leur production et en garantissant leur correction. De ce point de vue, les méthodes de synthèse sont très intéressantes, surtout en comparaison avec les méthodes analytiques. L'une des difficultés de la synthèse est qu'il n'est pas facile de couvrir toutes les fonctionnalités utilisées par les protocoles. Les méthodes ayant été proposées dans cette optique font souvent des restrictions qui les rendent quasiment non applicables.

Les objectifs de notre thèse se résument donc comme suit :

1. Proposer une méthodologie de développement de protocoles basée sur une approche de synthèse. Partant de notre recherche bibliographique, il apparaît que l'approche basée sur la synthèse satisfait mieux les exigences précédemment citées que l'approche analytique. C'est la raison pour laquelle nous nous sommes orientés vers la synthèse de protocoles. L'approche que nous avons adoptée démarre de la spécification des services requis et consiste à produire systématiquement les spécifications des entités de protocole implantant ce service. L'efficacité d'une telle approche peut se mesurer essentiellement par la puissance de son modèle de spécification de service et par son pouvoir de synthèse (c'est-à-dire, l'ensemble des fonctionnalités que l'on peut dériver au niveau protocolaire).

Dans une architecture distribuée, il est souhaitable que le modèle de spécification des services permette d'exprimer : (a) des événements parallèles de façon explicite, puisque le parallélisme est inhérent aux architectures distribuées, (b) des primitives de service paramétrées, (c) des variables virtuellement globales afin de fournir un niveau d'abstraction élevé dans la description des flots de données, (d) des accès concurrents aux variables globales, (e) la duplication de données pour augmenter le degré de disponibilité des données, et (f) des structures sélectives distribuées afin de spécifier des choix déterministes ou non déterministes entre plusieurs primitives de service en faisant abstraction de leur localisation.

Cet ensemble de caractéristiques définit une classe importante de services permettant d'exploiter efficacement un système distribué à un niveau d'abstraction élevé. Par conséquent, une méthode de synthèse prenant en compte cette classe de services devrait permettre de synthétiser simultanément : (1) le flot de contrôle avec prise en compte de la concurrence et du choix distribué, et (2) le flot de données avec prise en compte du contrôle de la cohérence. Pour la classe des services souhaitables (a)-(f), aucune des méthodes existantes ne fournit

d'algorithme permettant de dériver automatiquement un protocole correct. L'objectif est donc de proposer une méthodologie pour cette classe de services.
L'une des difficultés principales réside dans l'opposition de la puissance d'expression du modèle de spécification des services et le pouvoir de synthèse. En effet, le modèle d'expression doit être riche afin de pouvoir spécifier toutes les fonctionnalités nécessaires d'un service ou d'une partie de protocole. Alors qu'une méthode de synthèse exige l'utilisation d'un modèle mathématique relativement simple (automate à états finis, réseaux de Petri non interprétés, etc.) afin de conduire à des techniques de transformation de spécifications automatisables et préservant les critères de correction.

2. Démontrer l'intérêt de la méthodologie en proposant au moins une application réelle. Cette objectif est très important, car il correspond à l'une des exigences essentielles pour que la méthodologie soit acceptée dans un monde réel. Les méthodes existantes se contentent le plus souvent d'applications très simplifiées.

3. Démontrer la faisabilité de la méthodologie par un outil logiciel la supportant. De nombreux outils logiciels ont été mis en oeuvre pour faciliter le développement des protocoles et ainsi diminuer leur coût de production. Une grande partie de ces outils est basée sur des techniques analytiques telles que l'analyse d'accessibilité et la simulation [CHA 93]. Par ailleurs, très peu d'outils sont basés sur des techniques de synthèse. L'objectif étant de développer un outil qui met en oeuvre notre méthodologie de synthèse de protocoles.

4. Permettre la production du code des protocoles dans un langage de spécification standard et orienté implantation tel que Estelle. L'avantage est que le code obtenu peut être directement exploité par des outils existants tels que EDT[BUD 92], afin de générer automatiquement du code exécutable ou d'effectuer des simulations et des évaluations de performances.

Cette thèse s'articule autour de cinq chapitres. Des annexes viennent compléter certains points. Le chapitre 1 présente un état de l'art sur les méthodes de synthèse de protocoles de communication. Dans la classe des protocoles, nous incluons également les applications réparties et les convertisseurs de protocoles. Nous présentons des généralités sur le développement de protocoles en soulignant les motivations et intérêts de la synthèse par rapport à l'analyse. Nous y trouvons également une vision unifiée des étapes de développement d'un protocole selon une approche de synthèse ainsi que les difficultés qui s'y rattachent. Nous présentons aussi une classification et une comparaison des différentes méthodes de synthèse existantes en nous appuyant sur un ensemble de critères. Nous proposons ensuite, sur cette base, les directions qui nous semblent importantes à explorer dans le domaine de la synthèse de protocoles.
Les chapitres 2, 3, 4 et 5 correspondent respectivement aux objectifs que nous nous sommes fixés.
Dans le chapitre 2, nous proposons une nouvelle méthode de synthèse de spécifications de protocoles à partir de spécifications de service dans un modèle de réseaux de Petri interprétés. Notre approche traite à la fois le flot de contrôle, le flot de données, le choix distribué et les contraintes de cohérence de données. En d'autres termes, nous proposons un "bon"

compromis entre le pouvoir d'expression du modèle de spécification de service et le pouvoir de synthèse.
Le chapitre 3 présente une application réelle de notre approche de synthèse à la conception du protocole de transport ISO classe 0 [ISO 8073]. Le point de départ de la conception étant la spécification du service de transport ISO [ISO 8072].
Le chapitre 4 est consacré à la présentation et à l'expérimentation d'un ensemble d'outils logiciels pour la synthèse automatique de protocoles que nous appelons STEPS (Software Toolset for automatEd Protocol Synthesis).
Nous proposons au chapitre 5 une extension de notre méthode de synthèse pour la dérivation de spécifications de protocoles dans le langage Estelle.
L'annexe A fournit des définitions importantes liées à l'agglomération de transitions dans un réseau de Petri. L'annexe B présente la grammaire du langage de spécification des services et des protocoles utilisée par les outils STEPS. Enfin, les annexes C et D présentent les résultats des expérimentations effectuées au chapitre 4.

Chapitre 1

Méthodes de synthèse de protocoles de communication : état de l'art

Résumé

Le développement de protocoles de communication selon une approche de synthèse représente sans aucun doute une alternative importante aux approches analytiques dont le problème fondamental est l'explosion combinatoire de l'espace des états analysés. Le principal avantage de l'approche de synthèse est que la correction du protocole est assurée "par construction" lors du processus de synthèse. Depuis les premiers pas vers la synthèse de protocoles [ZAF 80], de nombreux travaux se sont intéressés à cette nouvelle direction. Nous proposons donc, à travers ce chapitre, une vision unifiée de la synthèse de protocoles ainsi qu'un état de l'art sur les techniques et les outils de synthèse existants.

1.1 Introduction

Les protocoles de communication jouent un rôle de plus en plus important dans les systèmes distribués. Un protocole de communication est la partie dite *logiciel de communication* dont le but est d'assurer le dialogue entre un ensemble de machines géographiquement réparties. Plus précisément, le protocole désigne l'ensemble des règles et des procédures à suivre pour réaliser ce dialogue. Il constitue en tant que tel un élément crucial pour une exploitation efficace et performante d'un système distribué. Il arrive malheureusement assez souvent que des erreurs surviennent en cours d'exploitation sur des protocoles considérés comme opérationnels. Parmi ces erreurs, nous pouvons citer la réception non spécifiée, signifiant que l'arrivée d'un message dans un état donné du protocole n'a pas été prévue et donc ne déclenche aucun traitement. La réalisation des protocoles peut être considérée comme un problème technique relativement nouveau faisant partie d'une discipline distincte d'ingénierie des logiciels à savoir *l'ingénierie des protocoles* [LIU 89]. Les techniques propres à cette discipline sont caractérisées principalement par leur capacité de produire des protocoles satisfaisant les besoins des utilisateurs. Ces besoins peuvent être classés en deux catégories : d'une part les besoins *qualitatifs* qui expriment qu'une chose indésirable ne doit pas se produire et que les propriétés attendues doivent se réaliser, et d'autre part les besoins

quantitatifs qui expriment les performances attendues du protocole en terme de vitesse et de taux d'utilisation des ressources. Les méthodes formelles et les outils logiciels qui les mettent en œuvre présentent actuellement un intérêt certain pour la production de protocoles satisfaisant les besoins des utilisateurs [CHA 93]. L'ingénierie des protocoles repose en général sur la définition d'un cycle de vie passant par plusieurs phases : de la définition des besoins au test de l'implantation. Nous distinguons deux approches différentes dans la définition du cycle de vie : l'approche basée sur *l'analyse* et celle basée sur la *synthèse*.
L'analyse consiste en général à décomposer le système et à examiner ses composants. Dans le cas des protocoles de communication, les techniques analytiques prennent en entrée les spécifications des services et du protocole et vérifient d'une part, la conformité des services rendus vis-à-vis des services attendus et d'autre part, l'absence d'erreurs de spécification, telles que les interblocages et les réceptions non spécifiées. Le cycle "détection d'erreurs et correction" est répété jusqu'à ce que le protocole soit valide. Ces techniques sont souvent très coûteuses en temps. De plus, la phase d'analyse passe généralement par une exploration exhaustive de l'espace des états accessibles dont le problème fondamental est l'explosion combinatoire. Un tour d'horizon sur les techniques analytiques est présenté dans [YUA 88]. Bien que l'approche analytique soit utile dans la validation de protocoles existants, elle ne fournit, néanmoins, aucune démarche pour la conception de nouveaux protocoles.

La synthèse est une méthode de réalisation des systèmes à partir de spécifications de haut niveau et de composants réutilisables [SET 93]. Dans le cas des protocoles, la synthèse consiste à construire de manière systématique ou interactive un protocole sans erreur en partant des spécifications formelles des besoins requis et/ou d'un ensemble de modules prédéfinis du protocole. L'objectif de la synthèse est d'augmenter la productivité des protocoles tout en diminuant le coût de leur production et en garantissant leur correction. De ce point de vue, les méthodes de synthèse sont très intéressantes, surtout en comparaison avec les méthodes analytiques. L'une des difficultés de la synthèse est qu'il n'est pas facile de couvrir toutes les fonctionnalités utilisées par les protocoles. Les méthodes ayant été proposées dans cette optique font souvent des restrictions qui les rendent quasiment non applicables. La première méthode orientée vers la synthèse de protocoles a été proposée en 1980 par ZAFIROPULO [ZAF 80]. Depuis, diverses approches sont apparues dans la littérature.

L'objectif de ce chapitre est de présenter un état de l'art sur les méthodes de synthèse de protocoles de communication. Dans la classe des protocoles, nous incluons également les applications réparties et les convertisseurs de protocoles. Nous présentons ainsi une classification et une comparaison des différentes méthodes de synthèse existantes en s'appuyant sur un ensemble de critères. Le critère clé qui distingue les méthodes de synthèse de protocoles est, outre leur point de départ [PRO 91] (c'est-à-dire, les spécifications initiales du processus de synthèse), la puissance d'expression du modèle de spécification et le contexte de synthèse (c'est-à-dire, les fonctionnalités qui peuvent être synthétisées). A chaque point de départ correspond une vision particulière du problème de synthèse pouvant avoir une application distincte. C'est ainsi que le problème de conversion de protocoles, qui est un problème d'interopérabilité très important dans les réseaux hétérogènes, peut-être formulé en termes de synthèse de protocoles. Le point de départ du processus de synthèse inclut entre autres les spécifications des protocoles que l'on souhaite faire interagir. Dans ce cas, il

s'agit de synthétiser un module dit *convertisseur de protocoles* jouant un rôle de médiateur entre les entités de protocole devant interagir.
L'originalité de cette étude réside dans le rapprochement entre deux axes, a priori indépendants, à savoir : la synthèse de protocoles et la synthèse de convertisseurs de protocoles, mais qui, à notre sens, résultent fondamentalement de la même problématique. L'avantage de notre état de l'art est qu'il présente d'une part une vision globale de la synthèse de protocoles au sein du domaine de l'ingénierie des protocoles, et d'autre part, il fournit une classification beaucoup plus fine et beaucoup plus large relativement à [PRO 91].

La suite de ce chapitre est organisée comme suit : dans la section 1.2, nous présentons des généralités sur le développement de protocoles en soulignant les motivations et intérêts de la synthèse par rapport à l'analyse. Nous y trouvons également une vision unifiée des étapes de développement d'un protocole selon une approche de synthèse ainsi que les difficultés qui s'y rattachent. La section 1.3 présente, selon un ensemble de critères, une classification des méthodes de synthèse de protocoles. Cette classification nous permet d'effectuer ensuite une étude comparative des principales méthodes de synthèse existantes. Ainsi, les sections 1.4, 1.5 et 1.6 présentent respectivement les trois classes principales, à savoir les méthodes orientées service, les méthodes non orientées service et les méthodes mixtes. En notant que la synthèse de convertisseurs de protocoles n'est rien d'autre qu'un problème particulier de synthèse de protocoles, nous consacrons la section 1.7 aux méthodes de synthèse de convertisseurs de protocoles. La conclusion présente les directions qui restent ouvertes dans le domaine de la synthèse de protocoles.

1.2 Développement de protocoles : vers une approche de synthèse

La complexité croissante des logiciels de communication exige une méthodologie de développement simple, efficace et qui permet de surmonter cette complexité. Depuis les années 80, un clivage entre deux approches de conception est né. D'une part, une approche analytique caractérisée par une validation postérieure et explicite du logiciel conçu, et d'autre part, une approche de synthèse caractérisée par une validation implicite et intégrée au sein même du processus de développement. Nous présentons ici, dans un premier temps, un cadre architectural multi-couches pour les protocoles de communication en mettant en évidence le problème de complexité des protocoles. Les critères définissant la correction d'un protocole sont développés ensuite dans la section 1.2.2. Nous présentons dans la section 1.2.3 les différents modèles permettant la spécification des services et des protocoles. Partant de cette base, nous décrivons dans la section 1.2.4 le cycle de vie d'un protocole selon une approche de synthèse que nous comparons au cycle de vie classique basé sur l'analyse.

1.2.1 Architecture des protocoles

Pour maîtriser la complexité du développement des logiciels de communication, les réseaux sont en général organisés en *couches*. Une couche donnée rend des *services* à la couche immédiatement supérieure en ajoutant certaines fonctionnalités aux services qui lui sont fournis par la couche immédiatement inférieure. De cette façon, les services fournis aux

utilisateurs (qui se trouvent au dessus de toutes les couches) sont les plus riches en fonctionnalités et cela sans que chaque couche soit trop complexe. Ce principe de structuration en couches a été adopté en particulier par le modèle de référence de l'ISO [ISO 7498] qui compte sept couches. Les couches inférieures (physique, liaison de données, réseau et transport) résolvent des problèmes de communication ; à savoir le transfert fiable des données. Les couches supérieures (session, présentation et application) mettent en œuvre des mécanismes de synchronisation qui permettent de développer de véritables applications réparties.

Nous présentons maintenant les principaux concepts relatifs à une architecture de communication multi-couches. Nous appelons "service N" le service fourni par la couche N, "protocole N", le protocole de la couche N et "entité N", une entité de protocole de la couche N. Le fonctionnement d'une couche N doit être vu à travers le triplet (service N, protocole N, service N-1).
Une couche N offre un service N à la couche N+1. Le service N est fourni par la coopération entre les entités N au moyen du protocole N (cf. Figure 1.1b). Le protocole N utilise le service N-1. Le service N est vu, de la couche N+1, comme une abstraction du comportement de la couche N et des couches inférieures à cette dernière (cf. Figure 1.1a). Il est caractérisé par un ensemble de *primitives de service* qui représentent les interactions entre les entités N+1 et les entités N. Ces interactions sont effectuées dans les entités N à travers des *point d'accès au service* de la couche N, appelées SAPs (Service Access Points). Les SAPs sont des identificateurs des entités N+1 dans le monde de la couche N. Le protocole N est caractérisé par un ensemble de messages ou PDUs (Protocol Data Units) échangés entre des entités N distantes. Cet échange est mis en œuvre au moyen de primitives de service de la couche N-1 effectuées à travers les SAPs de cette couche.

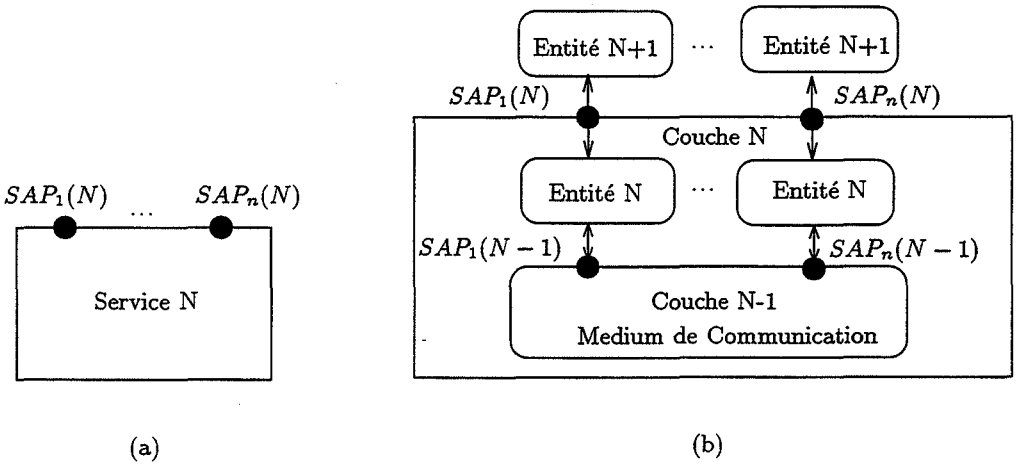

FIG. 1.1 – *Système de communication (a) Vue abstraite d'un service de couche N (b) Implantation distribuée d'un service de couche N*

Service et protocole sont deux concepts distincts mais étroitement liés. Le service définit les opérations qu'une couche est prête à effectuer mais ne dit rien sur la façon dont ces opérations sont réalisées. En revanche, un protocole est un ensemble de règles et de conven-

tions que les entités d'une même couche utilisent pour mettre en œuvre leur spécification de service. Une interaction entre une entité N et une entité N+1 se traduit par l'exécution d'une primitive de service. Les primitives de service ne doivent pas être exécutées dans un ordre arbitraire et avec des valeurs arbitraires des paramètres (même si ces valeurs sont comprises dans le domaine des valeurs admissibles). La primitive autorisée et les valeurs autorisées de ses paramètres dépendent de l'histoire antérieure des opérations. La spécification du service doit refléter ces contraintes. D'autre part, la spécification d'un protocole de couche N doit décrire le comportement de chaque entité de cette couche en réponse aux interactions initiées par ses utilisateurs, aux messages provenant des autres entités et aux événements locaux tels que le "timeout".

Par ailleurs, le besoin d'interconnexion de réseaux hétérogènes a donné naissance au problème de conversion de protocoles. Des réseaux sont dits hétérogènes quand leurs architectures de communication n'ont pas le même nombre de couches, quand des couches de même niveau sont caractérisées par des couples (service, protocole) différents, ou encore quand les adressages de même niveau sont différents [JUA 91]. Nous nous intéressons dans ce chapitre aux différences dans les couples (service, protocole) et particulièrement à la conversion de protocoles.

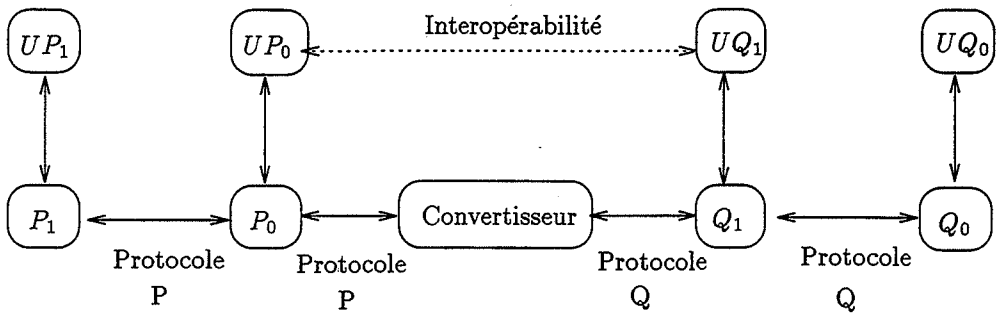

FIG. 1.2 – *Système de conversion de protocoles*

La conversion de protocoles intervient entre les PDUs échangés entre les entités des protocoles P et Q comme indiqué dans la Figure 1.2. La conversion est effectuée par un module appelé convertisseur. Ce module reçoit des PDUs du protocole P et les transforme en PDUs du protocole Q (ou inversement) tout en respectant les règles d'échanges des deux protocoles. La conversion est appliquée, en général, à des protocoles hétérogènes fournissant des services équivalents pour ne pas appauvrir de façon considérable le dialogue.
La première étape d'une méthodologie de conversion consiste à identifier les services communs entre les protocoles de même couche que l'on souhaite faire interopérer, puis à les comparer avec les services demandés par les utilisateurs. Une conversion sera possible lorsque l'ensemble des services communs contient tous les services demandés par les utilisateurs. La deuxième étape consiste à établir les relations entre les PDUs reçues et émises par le convertisseur afin de fournir tous les services communs et demandés par les utilisateurs.

1.2.2 Complexité des protocoles et critères de correction

Bien que l'architecture multi-couches des protocoles simplifie leur développement, chaque couche reste cependant complexe. Cette complexité est due à plusieurs facteurs :

- La nature distribuée des protocoles de communication. Il n'existe pas d'état global perceptible par une entité du système.
- Les services sous-jacents à une couche peuvent être non fiables. Cette non fiabilité devrait donc être prise en compte par la couche en question.
- Les systèmes interconnectés sont souvent hétérogènes, ce qui exige la mise au point de convertisseurs de protocoles.
- Les protocoles doivent être tolérants aux fautes. Le recouvrement des situations de panne est souvent complexe à réaliser.

La nature complexe des spécifications de protocoles favorise l'apparition d'une variété d'erreurs de conception. Ces erreurs peuvent être propagées vers la phase d'implantation si elles ne sont pas détectées et corrigées le plus tôt possible dans le processus de développement du protocole. Ces erreurs peuvent être classées en deux catégories : *les erreurs logiques* et *les erreurs sémantiques.*

Erreurs logiques :
La correction logique d'un protocole concerne la structure logique des échanges de messages entre les entités du protocole. Parmi ces erreurs, nous pouvons citer :

(1) *L'interblocage*: un protocole entre dans un état (autre que l'état final) dans lequel un ensemble d'entités de protocole est interbloqué. Chaque entité de l'ensemble attend l'occurrence d'un événement qui ne peut être produit que par une entité du même ensemble, et les canaux de communication sont vides.

(2) *La réception non spécifiée*: il y a réception d'un message auquel aucun traitement n'est associé.

(3) *Le bouclage*: le protocole entre dans un état dans lequel des cycles infinis non productifs sont possibles.

(4) *Les transitions non exécutables*: il existe des transitions du protocole qui ne sont jamais exécutées.

(5) *Le débordement des canaux de communication*: un protocole de communication est dit non borné s'il peut atteindre un état dans lequel un canal de communication liant deux entités de protocole contient un nombre de messages qui excède la capacité pré-définie du canal.

(6) *Terminaison non propre*: un protocole non cyclique (resp. cyclique) possède un problème de terminaison si, partant de son état initial, il n'atteint jamais son état final (resp. initial).

Erreurs sémantiques :
Un protocole possède des erreurs sémantiques s'il ne fournit pas tous les services attendus. Ce type d'erreur est spécifique au protocole. Par exemple, pour un protocole de contrôle d'accès à une section critique, s'il existe un état où deux processus entrent simultanément en section critique, cela voudrait dire que le protocole est sémantiquement incorrect. En effet, le service "un processus au plus dans la section critique" n'est pas fourni.

Un protocole correct doit satisfaire trois propriétés:

(a) *Correction logique*: cette propriété assure que le protocole est libre de toute erreur logique.

(b) *Correction sémantique*: cette propriété assure que le protocole est libre de toute erreur sémantique. En d'autres termes, le protocole fournit tous les services attendus.

(c) *Correction des performances*: cette propriété garantit que le protocole satisfait les performances requises.

Dans notre travail, nous nous intéressons aux propriétés qualitatives, c'est à dire les propriétés (a) et (b).
Au vu de ces problèmes, liés d'une part à la complexité des logiciels de communication et d'autre part aux différentes erreurs qui peuvent survenir, on sent la nécessité d'une méthode formelle pour la production de logiciels de communication corrects. Une telle méthode devrait s'appuyer sur un *modèle de spécification formelle* et être supportée par ce qu'on appelle *cycle de vie du logiciel.*

1.2.3 Modèles de spécification des services et des protocoles

Un modèle de spécification des services et des protocoles se doit de vérifier un certain nombre de critères:

(1) Pouvoir d'expression : il doit permettre de spécifier toutes les fonctionnalités des protocoles et des services.

(2) Bonne définition : il doit se fonder sur un modèle mathématique formel non ambigu permettant la validation automatique et/ou des transformations automatiques de spécifications.

(3) Structuration fonctionnelle : il doit fournir un moyen de structurer les spécifications afin qu'elles soient compréhensibles et faciles à maintenir.

(4) Abstraction : il doit permettre de décrire un protocole à des niveaux d'abstraction différents ; de la description du service (qui est le niveau d'abstraction le plus haut) à la description détaillée des entités de protocole. Il doit être indépendant des aspects d'implantation.

Un grand nombre de modèles de spécification ont été introduits dans la littérature. Ces modèles peuvent être classés en deux catégories: les modèles opérationnels et les modèles logiques. Les modèles opérationnels permettent la description de la partie comportementale

d'un système. Dans cette classe, nous distinguons d'une part les modèles à transition d'états tels que les automates à états finis [BOC 78, YUA 88] et les réseaux de Petri [MUR 89], et d'autre part, les modèles basés sur l'algèbre des processus, tels que CSP [HOA 85] et CCS [MIL 89]. Les modèles logiques permettent d'exprimer les propriétés d'un système sans spécifier comment ces propriétés doivent être réalisées. La logique temporelle [SCH 82] en est un exemple typique.
Par ailleurs, trois techniques de description formelles (TDFs) ont été adoptées par l'ISO, à savoir : Estelle [ISO 9074], SDL [CCITT 88] et LOTOS [ISO 8807], chacune étant basée sur l'un des modèles précédents. La TDF Estelle est fondée sur la notion d'automates communicants étendue avec le langage Pascal. SDL est une norme graphique reposant sur les automates à états finis et les types abstraits de données. LOTOS est fondée sur le modèle CCS étendu par les types abstraits axiomatiques issus du modèle ACT-ONE.

La spécification d'un protocole doit définir les spécifications de toutes les entités le composant. On utilise en général un modèle opérationnel pour la spécification d'un protocole ou plus précisément l'une des TDFs normalisées. D'autre part, le concept de service constitue l'un des plus importants concepts architecturaux des logiciels de communication [VIS 86]. La spécification de service doit définir seulement le comportement externe du protocole fournissant ce service. Deux approches sont possibles pour spécifier les services [GOT 90b] :

(a) L'ordonnancement explicite des primitives de service. LOTOS, CSP, les automates à états finis et les réseaux de Petri constituent des exemples de modèles pour cette approche.

(b) Les propriétés qui restreignent les ordonnancements possibles aux seuls ordonnancements corrects. Il s'agit d'une approche non constructive. La logique temporelle est un exemple de modèle pour cette approche.

L'approche explicite conduit facilement à l'implantation. Elle est utilisée dans les méthodes ayant pour but l'implantation d'un service, alors que l'approche implicite est plus adaptée à la vérification de la conformité du service rendu vis-à-vis du service attendu.

1.2.4 Cycle de vie d'un protocole

Le cycle de vie d'un protocole est la suite de phases qui conduit à sa production. Nous distinguons deux approches différentes : une approche basée sur l'analyse et une approche basée sur la synthèse.

1.2.4.1 Approche analytique

Dans cette approche, le développement d'un protocole passe par les phases suivantes :

1. Définition informelle des besoins
2. Description formelle
3. Analyse
4. Implantation

5. Test

La première phase consiste à établir un cahier de charges résumant les différents besoins des utilisateurs. Partant de cette base, et utilisant un modèle de spécification donné, la deuxième phase consiste à décrire formellement le protocole et éventuellement les services attendus. La troisième phase consiste à vérifier la correction logique et sémantique de la spécification du protocole. La technique de vérification utilisée dépend du modèle de spécification utilisé dans la phase 2. La technique la plus utilisée est l'analyse d'accessibilité. Elle consiste à construire le graphe de tous les états accessibles par le protocole, et de vérifier ensuite la validité des propriétés logiques et sémantiques en utilisant par exemple la logique temporelle. Cette technique a été mise en œuvre par certains outils logiciels tels que XEXAR [RIC 87] et VESAR [ALG 93]. Si des erreurs sont détectées dans cette phase, le concepteur doit reprendre la spécification du protocole pour la modifier en conséquence et refaire ensuite l'analyse. La phase 4 consiste à produire, à partir de la spécification du protocole, son code d'implantation sur une architecture donnée. Cette phase peut être supportée par des outils logiciels tels que EDT [BUD 92] qui permet de produire du code exécutable à partir de spécifications Estelle. La dernière phase consiste à tester la conformité de l'implantation par rapport à la spécification en utilisant des séquences de tests. Une étude détaillée sur le test peut être trouvée dans [ISO 9646][SID 89]. L'approche analytique a fait l'objet de beaucoup de travaux de recherche et a donné naissance à une panoplie d'outils logiciels la supportant [CHA 93]. Néanmoins, elle est fortement limitée par les inconvénients suivants :

(1) Bien qu'elle soit utile dans la validation de protocoles existants, elle n'offre cependant aucune démarche pour la conception de nouveaux protocoles.

(2) La phase 3 est appliquée itérativement par le concepteur jusqu'à ce que le protocole soit libre de toute erreur. Il s'agit donc d'une approche très coûteuse en temps de conception.

(3) La phase 3 passe généralement par une exploration exhaustive de l'espace des états accessibles dont le problème fondamental est l'explosion combinatoire [LIN 87].

1.2.4.2 Approche de synthèse

La synthèse de logiciels en général est une idée très ancienne ; les compilateurs en sont les premiers exemples. Actuellement, la puissance croissante des machines, la diminution de leur coût et l'avènement de l'informatique répartie rendent cette discipline plus que jamais attractive [SET 93]. A notre avis, pour être efficace, une technique de synthèse doit être ciblée sur un domaine d'application bien précis. La synthèse automatique de programmes scientifiques sur des architectures parallèles ou la synthèse automatique de protocoles en sont des exemples d'application.
La synthèse de protocoles est une approche systématique pour la conception de protocoles dont la correction est assurée "par construction". Ainsi, il ne serait plus nécessaire de passer par une phase d'analyse pour s'assurer de la validité du protocole. De cette manière, les problèmes (1), (2) et (3) de l'approche analytique seraient évités.
Le cycle de vie orienté synthèse d'un logiciel de communication peut être schématisé par la Figure 1.3.

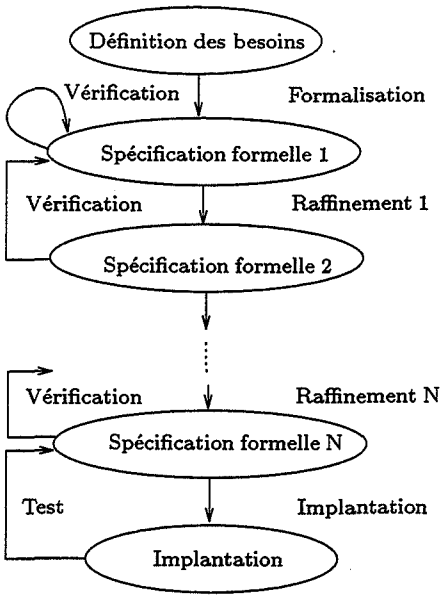

FIG. 1.3 – *Cycle de vie d'un protocole selon une approche de synthèse*

La première étape est la définition informelle des besoins en termes de fonctionnalités requises du protocole. L'étape suivante est la formalisation des besoins introduits dans la première étape. Ceci résulte soit en une spécification partielle du protocole (par exemple une entité du protocole), soit en une spécification complète du service de communication. Cette spécification est ensuite raffinée en une ou plusieurs étapes pour construire la spécification du protocole. Le raffinement doit être guidé par un ensemble de règles de transformations qui préservent la correction logique et sémantique des spécifications. Le raffinement peut être supporté par des outils logiciels. Cependant, d'après [SCH 92], les outils développés à cet égard, tels que [EIJ 89], sont souvent très complexes.
D'une manière générale, le raffinement peut être soit automatique soit interactif. Le processus de raffinement a pour but d'introduire des fonctionnalités protocolaires de façon incrémentale. Le résultat est que le produit final sera complexe comme tout système réel mais la manière de le construire garantit sa correction. La dernière étape (l'étape N+1) du raffinement consiste à produire une implantation du protocole à partir des spécifications formelles de l'étape N. Cette étape est similaire à la phase 4 de l'approche analytique.
En conclusion, cette approche garantit que les spécifications produites à l'étape N sont correctes à condition que celles de l'étape 1 le soient. Notons que la spécification d'un service définie à l'étape 1 est en général largement moins complexe que celle du protocole correspondant. Par conséquent, la vérification de sa correction devrait être moins difficile à réaliser. Une telle approche nous paraît très intéressante surtout en comparaison avec la précédente. Mais sa faisabilité doit être démontrée par des outils logiciels pouvant la supporter, ce qui reste très faible par rapport à l'approche analytique. Parmi les outils existants nous pouvons citer [RAM 85, SHI 91, CHA 94b] dont le fonctionnement et les limites sont détaillés dans la section 1.5.

L'une des difficultés principales de la synthèse automatique de protocoles réside dans l'opposition de la puissance d'expression du modèle utilisé dans l'étape 2 et le pouvoir de synthèse. Le modèle d'expression doit être assez riche afin de pouvoir spécifier toutes les fonctionnalités nécessaires d'un service ou d'une partie de protocole. Alors qu'une méthode de synthèse exige l'utilisation d'un modèle mathématique relativement simple (automates à états finis, réseaux de Petri non interprétés, etc.) afin de conduire à des techniques de transformation de spécifications automatisables et préservant le critère de correction.

1.3 Taxinomie

Pour pouvoir étudier et comparer les méthodes de synthèse de protocoles existantes, il est nécessaire de définir un ensemble de critères permettant de les évaluer. Pour ce faire, nous nous inspirons de certains critères présentés dans [PRO 91], à savoir: le point de départ de la méthode, le modèle utilisé pour décrire les protocoles et les services, les contraintes sur le modèle de communication, le mode d'interaction avec le concepteur et les propriétés garanties par la méthode. Nous rajoutons, cependant, un critère appelé *contexte de la synthèse* relatif aux fonctionnalités pouvant être synthétisées. Et nous développons davantage le critère modèle de spécification afin de dégager les principales fonctionnalités que doit pouvoir exprimer un modèle de spécification de service.

a) Point de départ de la méthode: D'une manière générale, le processus de synthèse peut commencer par l'un des points suivants: (a) à partir de spécifications informelles, (b) à partir de spécifications formelles des services, (c) à partir de spécifications formelles d'une (ou plusieurs) entité(s) de protocole, (d) à partir des deux points précédents.

b) Modèle de spécification : Il est utilisé pour spécifier les services de communication et les protocoles. Nous pouvons classer les aspects importants pris en compte (ou devant être pris en compte) dans un modèle de spécification de service en 4 catégories: flot de contrôle, flot de données, aspects temporels et aspects structurels.
b.1) Flot de contrôle: Ces aspects sont relatifs aux structures de contrôle pouvant être définies entre les primitives de service. Ils se résument comme suit:

C1: Exécution séquentielle de primitives de service.

C2: Choix, déterministe ou non-déterministe, entre plusieurs primitives de service liées à un SAP unique (il s'agit d'un *choix centralisé*).

C3: Exécution parallèle de primitives de service.

C4: Point de synchronisation entre plusieurs primitives de service.

C5: Choix, déterministe ou non-déterministe, entre plusieurs primitives de service liées respectivement à des SAPs distants l'un de l'autre (il s'agit d'un *choix distribué*).

C6: Primitive de service gardée par une clause conditionnelle.

C7: Primitive de service avec priorité d'exécution.

b.2) Flot de données : Ces aspects sont relatifs à la définition de mécanismes permettant la circulation de données entre les primitives de service. Ils se résument comme suit :

D1 : Spécification de primitives de service avec paramètres.

D2 : Spécification de données virtuellement globales aux primitives de service.

D3 : Duplication des données globales à travers les points d'accès au service.

D4 : Accès concurrents des primitives de service aux données globales.

Ainsi, des schémas élaborés de circulation de données entre les primitives de service peuvent être réalisés soit par passage de paramètres entre les primitives de service [GOT 90a], soit par accès des primitives de service aux données globales [HIG 93].
b.3) Aspects temporels : Ces aspects définissent des contraintes de temps sur les primitives de service. Ils permettent ainsi de spécifier des services temps-réel [KHO 94a].
b.4) Aspects structurels : Ces aspects concernent la structure d'une spécification de service. On distingue la modularité et la réutilisation de services existants [GOT 90a].

c) Contraintes sur le modèle de communication : Elles définissent l'architecture cible sur laquelle doit être implanté le protocole ainsi que des hypothèses sur cette architecture. Parmi ces contraintes, nous pouvons citer : *le mode d'interaction* entre un utilisateur de service et les entités de protocole (RendezVous ou mode asynchrone), *le nombre d'entités de protocoles* pouvant être synthétisées et *les caractéristiques du canal de communication* (ordre des messages, degrés de fiabilité, etc.).

d) Mode d'interaction avec le concepteur : Le protocole de communication peut être produit soit *automatiquement*, sans intervention du concepteur, soit de manière *interactive.*

e) Propriétés garanties par la méthode : Elles concernent la correction logique et sémantique du protocole construit.

f) Contexte de la synthèse : Il définit les fonctionnalités pouvant être synthétisées par la méthode au niveau protocole. Elles se divisent en fonctionnalités dérivées du service et fonctionnalités ajoutées par la méthode (ou valeur ajoutée). Parmi les fonctionnalités dérivées du service nous citons : *le flot de contrôle*, *le flot de données*, *les aspect temporels* et *les aspects structurels.* Ces dernières ont la même sémantique que celles définies dans le service (voir le second critère) sauf qu'elles doivent tenir compte de l'environnement distribué du protocole. D'autre part, les fonctionnalités ajoutées concernent entre autres : *le recouvrement d'erreurs* (si le médium est non fiable), *le contrôle du flux de données*, et *le contrôle de la cohérence des données.* Le contexte de la synthèse est un critère permettant d'évaluer le *pouvoir de synthèse* d'une méthode. Plus le contexte est riche, plus le domaine d'application de la synthèse est large ce qui signifie que le pouvoir de synthèse est grand.

Selon leur point de départ, les méthodes de synthèse peuvent se diviser en trois grandes catégories : les méthodes *orientées service*, les méthodes *non orientées service* et les méthodes *mixtes.*

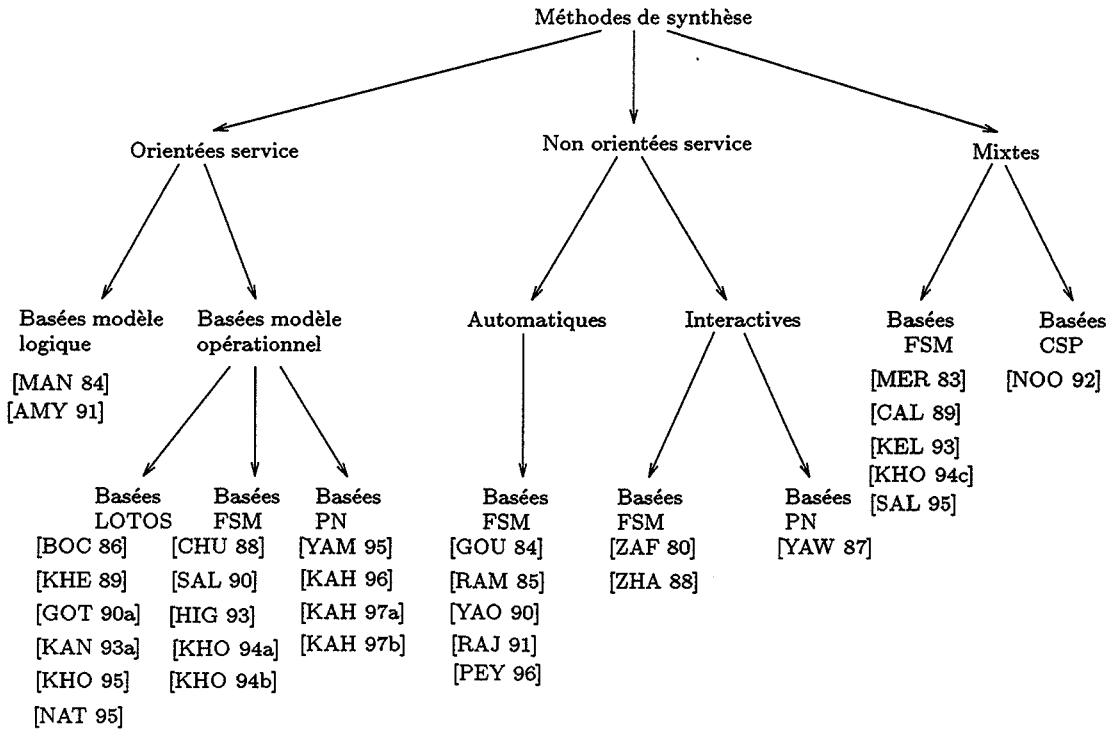

FIG. 1.4 – *Classification des méthodes de synthèse de protocoles existantes*

Les méthodes orientées service partent d'une description complète du service de communication requis et consistent à dériver de manière automatique un protocole correct. Dans ce contexte, nous distinguons les méthodes basées sur LOTOS [BOC 86] [KHE 89] [GOT 90a] [KAN 93a][KHO 95][NAT 95], les méthodes basées sur un modèle d'automates à états finis [CHU 88] [SAL 90] [HIG 93] [KHO 94a][KHO 94b] [KAK 94], celles basées sur un modèle logique [MAN 84][AMY 91] et celles basées sur un modèle de réseaux de Petri [YAM 95][KAH 96][KAH 97a][KAH 97b]. La dernière catégorie, faisant partie de notre travail, est abordée en détail dans les chapitres suivants.

Les méthodes non orientées service sont basées sur la description partielle ou complète d'une (ou plusieurs) entité(s) de protocole pour construire la spécification duale de l'autre (ou des autres) entité(s) de protocole. Parmi les principales méthodes ayant été proposées dans cette optique, nous pouvons citer les méthodes de synthèse automatique de protocoles [GOU 84] [RAM 85], les méthodes de synthèse automatique de convertisseurs de protocoles [YAO 90][RAJ 91][PEY 96], et les méthodes de synthèse interactive de protocoles [ZAF 80] [YAW 87] [ZHA 88]. Notons que les méthodes de synthèse de convertisseurs de protocoles considèrent, en plus, des spécifications propres au contexte de la conversion.

Les méthodes mixtes prennent en entrée à la fois une spécification complète du service et des spécifications d'un ensemble d'entités de protocole et consistent à dériver les spécifications des entités restantes. Nous distinguons la méthode de Merlin [MER 83] et les méthodes de synthèse de convertisseurs de protocoles [CAL 89] [NOO 92] [KEL 93][SAL 95].

La Figure 1.4 présente notre classification des principales méthodes de synthèse de protocoles et de convertisseurs de protocoles existantes. Ces méthodes sont abordées dans les sections suivantes.

1.4 Approche orientée service

Nous présentons dans cette section les principales méthodes de synthèse basées sur le concept de service. Nous terminons par une comparaison selon les critères décrits dans la section 1.3.

1.4.1 Synthèse de protocoles dans le langage LOTOS

1.4.1.1 Problématique

LOTOS est un langage de spécification des systèmes distribués développé par l'ISO. L'idée de base à partir de laquelle LOTOS a été développé est qu'un système peut être spécifié en définissant une relation temporelle entre les interactions qui constituent le comportement externe observable du système. Cette description est basée sur l'algèbre de processus issue du modèle CCS [MIL 89]. Plusieurs niveaux d'abstraction peuvent ainsi être décrits avec LOTOS. La problématique de la synthèse de protocoles dans ce langage consiste à trouver une stratégie permettant de transformer automatiquement une expression LOTOS d'un niveau d'abstraction élevé spécifiant un service, en une expression LOTOS d'un niveau d'abstraction plus bas spécifiant le protocole qui implante ce service. La stratégie en question doit garantir la correction logique et sémantique du protocole construit.

1.4.1.2 Méthodes existantes

La dérivation de protocoles de communication à partir de spécifications complètes de service a été étudiée pour la première fois par BOCHMANN [BOC 86]. Le langage de spécification des services est un sous-ensemble très simplifié de LOTOS. Un service est défini par une expression algébrique selon la grammaire suivante :

$$e \longrightarrow e;e \mid e|||e \mid e[]e \mid x$$

où x représente une primitive de service associée à un point d'accès donné. Les opérateurs ";", "|||" et "[]" spécifient respectivement la séquentialité, le parallélisme et le choix non déterministe. Le symbole "|" délimite les alternatives de la règle. Le protocole de communication synthétisé est décrit dans le même langage étendu par des primitives d'émission et de réception de messages.
Le principe de base de la méthode est d'obtenir la spécification de chaque entité de protocole par la projection de la spécification du service sur chaque point d'accès au service de cette spécification. La projection sur un point d'accès consiste à éliminer toutes les primitives de service qui ne se rapportent pas à ce point d'accès. Les spécifications obtenues par projection sont complétées avec des éléments de synchronisation (émission/réception de messages) pour assurer que l'ordre temporel des opérations exécutées aux SAPs respectifs soit conforme à l'ordre pré-défini par la spécification du service.
Cette méthode suppose que le médium de communication est fiable. L'algorithme de dérivation considère l'arbre syntaxique attribué correspondant à l'expression spécifiant le service.

Les attributs sont utilisés pour préciser les points d'accès à travers lesquels doivent s'échanger des messages de synchronisation. Si on considère l'expression "$e_1; e_2$", il y a besoin de synchronisation entre certains points d'accès de e_1 et e_2.

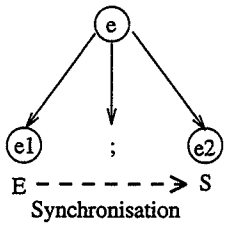

FIG. 1.5 – *Arbre attribué associé à l'expression* $e_1; e_2$

En effet, l'arbre associé à cette expression est donné par la Figure 1.5. L'attribut E de e_1 définit l'ensemble des SAPs où les dernières primitives de e_1 sont exécutées et l'attribut S définit l'ensemble des SAPs où les premières primitives de e_2 sont exécutées. Par conséquent, il doit y avoir synchronisation entre les deux ensembles de points d'accès. Cette synchronisation se fait par des émissions de messages des SAPs de l'ensemble E vers les SAPs de S. Basé sur ce concept d'arbre attribué, l'algorithme de dérivation consiste à définir pour chaque SAP (noté p) une fonction de projection (notée Tp) qui, appliquée à une expression de service (notée e), fournit une expression de l'entité de protocole associée à ce point d'accès p (notée $T_p(e)$).
A titre d'exemple, on considère un service $e = a^1; b^2$ où a^1 est une primitive localisée au point d'accès 1 et b^2 au point d'accès 2. L'algorithme de dérivation appliqué au service e donne les deux entités de protocole suivantes :

- au point d'accès 1 : $T_1(e) = a^1; send(2, m)$
- au point d'accès 2 : $T_2(e) = receive(1, m); b^2$

Le paramètre m désigne un message de synchronisation. La correction logique et sémantique des protocoles construits n'a pas été prouvée. D'après [KHE 89], la méthode engendre quelquefois des messages de synchronisation non nécessaires. Le langage de spécification utilisé étant très simplifié, il ne permet pas de spécifier des structures répétitives, la récursivité, le choix distribué et la modularité. De plus, la méthode de BOCHMANN ne permet pas d'exprimer l'échange de données, elle est limitée à la synthèse du flot de contrôle.

La méthode de BOCHMANN a été étendue dans [GOT 90a] par l'expression de primitives de service paramétrées dans une spécification de service. L'exemple le plus typique est le paramètre qui définit, dans une primitive initiée par un utilisateur, des données à transmettre à un autre utilisateur. Dans ce cas, les entités de protocole doivent s'échanger non seulement des messages de synchronisation mais aussi des messages de données. Une primitive de service possède la forme suivante :

$$a^p(x_1, ..., x_m / x_{m+1}, ..., x_n)$$

où a représente le nom de la primitive, p dénote son point d'accès, $x_1, ..., x_m$ désignent des paramètres formels d'entrée et $x_{m+1}, ..., x_n$ sont des paramètres formels de sortie.

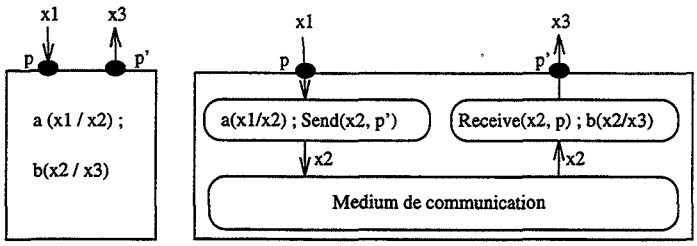

FIG. 1.6 – *Spécifications d'un service et du protocole correspondant*

Dans la Figure 1.6, la primitive a au point d'accès p est suivie par b au point p'. La valeur du paramètre x_1 doit être fournie par l'utilisateur au point d'accès p. Le paramètre x_2 est en sortie pour la primitive a, et il est utilisé en entrée par la primitive b à travers le point p'. Ce paramètre peut, ainsi, définir un échange de données. Le paramètre x_3 est en sortie pour la primitive b à travers le point d'accès p'. La Figure 1.6 représente aussi les entités du protocole de communication implémentant le service donné. La méthode de GOTZHEIN génère quelquefois des messages non nécessaires. Par exemple, pour un service $a(x_1/x_2); b(x_2/x_3)$, le paramètre x_2 doit servir à la fois de message de synchronisation et de données. Il n'est donc pas nécessaire d'avoir un autre message de synchronisation. De plus, cette méthode impose deux restrictions majeures dans l'utilisation des paramètres: d'une part, les paramètres doivent être à assignation unique et d'autre part, ils ne peuvent pas faire l'objet d'accès concurrents.

Une autre extension de la méthode de BOCHMANN à des spécifications très proches du langage LOTOS de base [BOL 87] a été présentée dans [KAN 93a] [KAN 93b]. Cette extension introduit de nouveaux opérateurs, tels que *l'interruption* et le *parallélisme avec synchronisation.* Elle améliore aussi l'expression de la récursivité qui était de type "récursion terminale" dans [KHE 89], et fournit dans [KAN 93b] une preuve formelle de l'équivalence entre spécifications de service et spécifications du protocole correspondant.

Néanmoins, [KAN 93a] impose deux restrictions sur les structures sélectives de la forme $e_1 [] e_2$:

- R1 : $S(e_1) = S(e_2)$ (S fournit l'ensemble des SAPs où les premières primitives de service sont exécutées).
- R2 : $E(e_1) = E(e_2)$ (E fournit l'ensemble des SAPs où les dernières primitives de service sont exécutées).

Ces restrictions excluent, donc, les structures sélectives distribuées de la classe des services que l'on peut spécifier. D'autre part, deux autres restrictions sont imposées sur les structures d'interruption de la forme $e_1 [> e_2$:

- R2 : $E(e_1) = E(e_2)$
- R3 : $E(e_1) \subseteq S(e_2)$

Dans [KAN 93b], cette méthode est appliquée pour la synthèse d'une version simplifiée du protocole de transport ISO [ISO 8073]. Considérant une partie de la spécification du service qui concerne la phase de fermeture de la connection de transport:

```
SPEC DISC = A(1,2) [] A(2,1) WHERE
     A(i,j) = dsreq_i ; ((dsind_j ; exit) [] (dsreq_j ; C))
     C = ((dsind_1 ; exit) ||| (dsind_2 ; exit)
```

Où `dsreq_i` désigne une requête de déconnexion émanant du SAP i et `dsind_i` dénote une indication de déconnexion au SAP i. Dans cette expression, la restriction R1 n'est pas respectée, ceci a pour conséquence que le protocole dérivé de cette spécification contient un interblocage. En effet, les spécifications des deux entités de protocole sont données comme suit:

```
SPEC DISC_i = A_i [] B_i WHERE
     A_i = (dsreq_i ; send(m_i, j) ; exit)
           >> ((receive(p_i, j) ; exit) [] ((receive(q_i, j) >> C_i))
     B_i = receive(m_j, j) ; exit >> ((dsind_i ; exit >> send(p_j, j) ; exit)
           [] (dsreq_i ; (send(q_j, j) ; exit) >> C_i))
     C_i = dsind_i ; exit
```

L'interblocage apparaît quand les deux entités `DISC_1` et `DISC_2` s'engagent simultanément dans une déconnexion: dans l'entité `DISC_1` le processus `A_1` exécute `dsreq_1`, envoie un message `m_1` et se met en attente du message `p_1` ou `q_1`. Dans l'entité `DISC_2`, le processus `A_2` exécute `dsreq_2`, envoie un message `m_2` et se met en attente du message `p_2` ou `q_2`. Les messages attendus par les deux entités ne sont jamais envoyés, d'où la situation d'interblocage. Une solution a été proposée à ce problème dans [KHO 95]. Elle consiste à faire exécuter `A_1` et `A_2` en exclusion mutuelle. Pour ce faire, les auteurs ont étendu le langage de spécification par des variables globales utilisées pour synchroniser l'exécution des primitives de service dans les cas où la méthode [KAN 93a] ne marche pas. Pour réaliser ces synchronisations, les primitives de service sont encapsulées dans des transactions. Le contrôle des transactions est réalisé implicitement par le principe du verrouillage à deux phases. Cette méthode est limitée à la synthèse du flot de contrôle. En effet, les variables globales utilisées n'ont aucune sémantique liée au protocole, elles servent simplement dans la spécification de certaines structures de contrôle qui ne sont pas supportées par [KAN 93a]. De plus, les problèmes du choix et de l'interruption distribués ne sont pas totalement transparents au spécifieur. Les deux méthodes [KAN 93a] et [KHO 95] ne prennent pas en compte le paramétrage des primitives de service, ce qui peut être gênant pour leur application à des protocoles réels.

Pour résoudre le problème du choix distribué, [KAN 93b] propose l'utilisation de la solution de LANGERAK [LAN 90]. Cette solution consiste à remplacer une expression de choix par une expression équivalente (de manière observationnelle) qui fait intervenir des événements internes non observables. Soit par exemple une expression $e_1 [] e_2$ où e_1 et e_2 ont lieu respectivement aux SAPs 1 et 2. Supposons que le fournisseur de service commence arbitrairement par offrir e_1 au SAP 1. Si l'utilisateur attaché au SAP 1 accepte l'offre, le choix est terminé. Sinon, par une action non observable par les utilisateurs, le fournisseur du service commute

sur le SAP 2 pour offrir e_2. De façon similaire, si l'utilisateur attaché au SAP 2 accepte l'offre, le choix est terminé, sinon le choix est commuté sur le SAP 1 et ainsi de suite. Le choix est décrit par l'expression suivante :

$$B = e_1 [] i_1; (e_2 [] i_2; B)$$

Où i_1 et i_2 sont deux actions non observables associées respectivement aux SAPs 1 et 2. Cette solution introduit à chaque choix distribué des actions internes non productives.
Enfin, une dernière extension de [KAN 93a] par des contraintes de temps a été apportée dans [NAT 95]. Ceci a nécessité une extension du langage LOTOS par des aspects temporels.

Les méthodes basées sur l'approche de BOCHMANN sont destinées plus spécifiquement aux modèles de communication synchrones, ce qui limite leur application à des systèmes synchrones. Au vu de ces méthodes, nous pensons qu'il serait intéressant de combiner les avantages des méthodes présentées dans [GOT 90a] et [KAN 93a]. Ceci permettrait d'étendre le langage de spécification défini dans [KAN 93a], qui est le plus proche de LOTOS de base, par l'expression de primitives paramétrées telles qu'elles sont définies dans [GOT 90a]. La même extension serait également envisageable pour les méthodes [KHO 95] et [NAT 95]. Ces méthodes de synthèse trouvent leur application directe dans la conception de protocoles de communication. Les auteurs affirment aussi leur utilité dans les systèmes où les demandes de service changent fréquemment, ce qui est le cas des systèmes de gestion de bases de données réparties. Une autre application possible concerne les problèmes qui demandent la répartition d'une activité sur plusieurs sites d'exécution, par exemple, un processus de contrôle réparti.

1.4.2 Synthèse de protocoles dans un modèle d'automates

1.4.2.1 Problématique

Les automates à états finis constituent l'un des premiers modèles utilisés pour la description des protocoles [BOC 78]. Chaque entité communicante est décrite par un automate. Les automates communiquent entre-eux en échangeant des messages. Les interactions peuvent s'effectuer soit par files d'attente soit par rendez-vous. Les automates peuvent être étendus par un jeu de variables locales (automates d'états finis étendus). Ces variables peuvent être transportées dans des messages. Elles permettent par exemple de numéroter les messages. Elles ont pour rôle de réduire notablement la complexité de l'automate en diminuant le nombre des états et des transitions. D'autres extensions peuvent aussi être considérées, telles que les délais d'exécution des transitions et les priorités des transitions. Les problèmes qui doivent être abordés dans cette approche concernent d'une part la description d'un service dans le modèle d'automate et d'autre part la construction à partir de cette description d'un ensemble d'automates communicants spécifiant le protocole dérivé. La problématique introduite dans cette approche diffère essentiellement de la précédente par le modèle utilisé dans la technique de synthèse.

1.4.2.2 Méthodes existantes

Une première technique ayant été proposée dans ce contexte consiste à synthétiser des protocoles à deux entités [CHU 88]. Cette méthode poursuit l'approche précédente en ce

sens que le principe est de dériver automatiquement une spécification de protocole à partir de spécifications complètes de service. Cette méthode utilise le modèle des automates à états finis pour la spécification des services et du protocole correspondant. Un service est décrit par l'entrelacement des primitives de services localisées à différents points d'accès distribués. Une transition de l'automate modélise l'exécution d'une primitive de service pouvant être activée soit par l'utilisateur soit par le fournisseur de service. Une exécution concurrente de plusieurs primitives de service peut aussi être exprimée à l'aide de transitions particulières dites concurrentes. Un protocole est spécifié par un ensemble d'automates communicants. Chaque automate modélise une entité du protocole.
Le principe de cette méthode est de synthétiser deux automates, communiquant à travers un médium fiable, à partir d'un automate spécifiant le service du protocole. Pour ce faire, la méthode consiste en trois étapes :

1. Pour chaque état de l'automate modélisant les services,
   - partitionner les transitions passant par cet état en 4 groupes selon le point d'accès utilisé (SAP_1 ou SAP_2) et le sens de la transition,
   - décomposer l'état courant en 4 sous-états, chacun étant associé à une partition de transitions,
   - et ajouter des transitions de synchronisation entre les sous-états de telle sorte que la sémantique de l'automate initial soit maintenue.
2. Projeter l'automate obtenu sur chaque point d'accès. Les transitions non nécessaires[1] sont remplacées par des transitions spontanées.
3. Supprimer les transitions spontanées des deux automates obtenus en utilisant l'algorithme de suppression de transitions spontanées décrit dans [BAR 79].

Cette méthode présente quelques limitations : (a) les entités construites sont fortement synchronisées, (b) elle ne traite pas les protocoles à plus de deux entités, (c) d'après une étude faite dans [SAL 91], l'algorithme de dérivation peut engendrer des bouclages infinis, (d) elle ne considère pas l'aspect "données" dans la description des services et des protocoles, elle exclut donc une grande partie des protocoles, (e) et la spécification de la concurrence au niveau service nécessite l'introduction d'un concept de transition concurrente qui n'existe pas dans le modèle d'automate.

Cette méthode a été ensuite généralisée dans [SAL 90] pour la synthèse de protocoles à plus de deux entités. Le principe est le suivant : Lorsque dans la spécification du service deux primitives de service A et B sont invoquées séquentiellement à des points d'accès SAP_1 et SAP_2,

- après exécution de A par l'entité de protocole associée à SAP_1, celle-ci envoie un message de synchronisation à l'entité associée à SAP_2 (notée EP_2),
- après réception du message de synchronisation par EP_2, celle-ci exécute B.

1. Transitions qui ne font pas référence au SAP sur lequel se fait la projection

Les étapes de cette méthode peuvent se résumer comme suit :

1. Projeter la spécification de service sur les points d'accès au service (de la même manière que la méthode de CHU).

2. Rajouter des transitions d'émission et de réception de messages de synchronisation à chaque automate obtenu selon un ensemble de règles. Par exemple, si une primitive de service d'une entité de protocole E_1 est suivie par une transition d'une entité de protocole E_2, alors une émission d'un message de E_1 vers E_2 est insérée entre les deux transitions.

3. Supprimer les transitions spontanées par l'algorithme décrit dans [BAR 79].

Cette méthode peut être vue comme une généralisation de la méthode de CHU pour la dérivation de protocoles ayant plus de deux entités. Les protocoles produits sont corrects logiquement et sémantiquement, mais le modèle de spécification des services n'est pas assez puissant puisqu'il ne permet pas l'expression de la concurrence et du flot de données.

Plus tard, KHOUMSSI *et. al.* [KHO 94a] ont repris l'approche de SALEH [SAL 90] en apportant une extension du modèle de spécification des services par des contraintes temporelles. A chaque transition de l'automate spécifiant le service est associé un ensemble d'intervalles de temps. Un intervalle de temps d'une transition t dépend de la transition t' par laquelle on peut atteindre t. Une transition doit être exécutée dans son intervalle de temps. De plus, une transition sortante doit obligatoirement être exécutée après avoir atteint un état donné. Ceci permet d'éviter les blocages. Le principe de la dérivation des contraintes temporelles sur les entités de protocole est le suivant : *connaissant les contraintes de temps sur les transitions de la spécification de service, il s'agit de déterminer, pour chaque entité de protocole, les intervalles de temps associés aux transitions d'émission et de réception de messages de synchronisation de telle sorte que les contraintes sur le service soient respectées.* Deux cas sont principalement considérés par la méthode :

- Cas statique : les messages de synchronisation ne contiennent aucune information sur le temps.

- Cas dynamique : les messages sont émis avec des informations temporelles telles que l'estimation de la durée de transit et le temps qui sépare l'exécution d'une transition de service de l'émission d'un message de synchronisation. Ceci permet aux entités réceptrices de mieux exploiter le temps qui leur a été réservé afin de satisfaire le service désiré.

Dans les deux cas, la méthode suppose la connaissance du plus petit intervalle de temps contenant le temps de transit d'un message. Cette méthode peut être appliquée aux cas où plusieurs systèmes doivent interagir pour accomplir des tâches avec des délais fixés. Cependant, elle exclut l'aspect "donnée" et quelques aspects importants du flots de contrôle tels que le choix distribué et la concurrence. Les auteurs ont apporté deux améliorations dans [KHO 94b], d'une part en considérant les systèmes concurrents et d'autre part en affinant les contraintes temporelles à des transitions qui ne sont pas nécessairement consécutives.

Une première méthode de synthèse qui considère l'aspect "donnée" dans le modèle d'automates a été proposée dans [HIG 93]. Cette méthode propose une approche de dérivation basée sur un modèle d'automates à états finis étendus par des données globales appelées *registres*. Une transition du modèle représente une primitive de service en faisant la distinction entre une primitive en entrée et une primitive en sortie à travers un point d'accès donné. Une transition représentant une primitive en entrée s'exprime sous la forme suivante :

$$< a?(x_1, ..., x_n), C, R >$$

L'expression $a?(x_1, ..., x_n)$ dénote un événement en entrée à travers le point d'accès a, C représente la condition d'exécution de la transition et R dénote un ensemble d'actions à entreprendre quand l'événement en entrée est activé et la condition C est vérifiée. Une transition représentant une primitive de service en sortie s'exprime ainsi :

$$< a!(E_1, ..., E_n), C, R >$$

Elle signifie que si la condition C est vérifiée, l'ensemble d'actions R est exécuté et les valeurs des expressions $E_1, ..., E_n$ deviennent disponibles à travers le point d'accès a. L'expression $a!(E_1, ..., E_n)$ dénote dans ce cas un événement en sortie.
Le problème consiste à *dériver k entités de protocole connaissant la spécification des services ainsi que l'allocation des registres et des points d'accès aux différentes entités de protocole.* Le principe de la méthode est le suivant : chaque transition du service est remplacée par une séquence de transitions incluant des émissions et des réceptions de messages de synchronisation et de données. Les messages de synchronisation assurent l'ordonnancement spécifié dans le service tandis que les messages de données sont utilisés pour gérer la modification de la valeur d'un registre éventuellement dupliqué ou l'accès à un registre distant. Les transitions qui ne participent pas à une entité de protocole sont supprimées par un algorithme de suppression des transitions vides. Une phase d'optimisation du nombre de messages produits vient compléter la méthode. Le problème d'optimisation a été ramené à un problème de programmation linéaire.
L'avantage de cette méthode par rapport aux précédentes est l'introduction du concept de registre qui permet d'exprimer le flot de données. Cette méthode a été ensuite étendue par l'expression du parallélisme dans la spécification de service en utilisant les réseaux de Petri à choix libre [YAM 95]. Néanmoins, son application reste assez limitée puisqu'elle ne traite pas les structures sélectives distribuées et interdit les accès concurrents aux registres.

Une autre vision du problème a été proposée dans [CAI 97]. Elle consiste à synthétiser un programme réparti à partir d'un programme séquentiel. Le programme séquentiel est modélisé par un automate réactif (i.e., incluant des réactions à un environnement externe) étiqueté par les instructions atomiques du programme source. Cette méthode va dans le même sens que celle proposée dans [SAL 90]. Elle propose en plus un mécanisme de gestion du flot de données.

1.4.3 Synthèse de protocoles basée sur un modèle logique

1.4.3.1 Problématique

Nous avons mentionné dans la section 1.2.3 qu'un service peut être spécifié dans un modèle logique. En effet, certains comportements externes du systèmes sont plus simples à exprimer

dans un modèle logique que dans un modèle opérationnel. Par exemple, il est plus simple d'exprimer un invariant à l'aide d'une propriété de logique temporelle qu'en utilisant les automates à états finis. De ce point de vue, il serait intéressant qu'une méthode de synthèse prenne comme point de départ des spécifications de service décrites dans un modèle logique. Dans ce cas, la difficulté principale réside dans la transformation d'une expression décrite dans le modèle logique en expression d'un modèle opérationnel.

1.4.3.2 Méthodes existantes

Très peu de travaux ont été consacrés à cette approche. L'une des premières méthodes proposées dans ce contexte est due à MANNA et WOLPER [MAN 84]. Cette méthode consiste à construire des processus CSP à partir de formules de logique temporelle.
D'autre part, AMYAY [AMY 91] a proposé une méthodologie de synthèse dont le point de départ est défini par une spécification de service en logique épistémique [HAL 86] et un ensemble de règles d'inférence. La spécification de service exprime des propriétés de connaissance entre un ensemble d'agents utilisateurs de service. Partant de ces spécifications, cette approche construit d'abord un graphe global modélisant tous les états de connaissance du système décrit, et produit ensuite un système de transitions étiquetées pour chaque agent fournisseur de service. Cependant, le graphe global des états de connaissance peut être très complexe pour des protocoles à grande complexité combinatoire. Par ailleurs, nous estimons qu'une telle approche serait plus réaliste si elle était combinée avec une approche basée sur un modèle opérationnel, car les comportements externes d'un système ne sont pas tous facilement exprimables dans un modèle logique.

1.4.4 Discussion sur les méthodes orientées service

Après avoir passé en revue les différentes méthodes de synthèse orientées service, nous constatons que le principe de base commun à toutes ces méthodes consiste à distribuer une spécification de service, faisant intervenir des primitives de service et des ressources distribuées, à travers un ensemble de points d'accès au service (qui définissent les sites participant à la réalisation du service). L'objectif est d'obtenir des spécifications communicantes, chacune localisée à un point d'accès particulier. Ces spécifications doivent réaliser le service requis. L'apport principal de ces méthodes est l'abstraction qu'elles offrent quant aux problèmes de synchronisation et de communication. Le concepteur peut spécifier de manière centralisée un service qui fait intervenir des composants répartis, sans se soucier des problèmes de communication et de synchronisation.
Nous présentons dans ce qui suit quelques observations sur les méthodes abordées et nous terminons par un tableau comparatif (cf. Tableau 1.1) :

- Nous estimons que l'efficacité d'une méthode de synthèse orientée service se mesure essentiellement par la puissance d'expression de son modèle de spécification des services et par son contexte de synthèse. Nous avons remarqué que le pouvoir d'expression des services va à l'encontre du pouvoir de synthèse des protocoles. Plus le modèle est riche, plus il est difficile de synthétiser des spécifications écrites dans ce modèle. Chacune des méthodes abordées prend en compte certains aspects dans la spécification des services et ignore d'autres aspects. Il serait donc intéressant de trouver un bon compromis.

	[BOC 86]	[GOT 90a]	[KAN 93a]	[AMY 91]	[CHU 88]	[SAL 90]	[KHO 94a]	[HIG 93]
Point départ	Spécif. service	Spécif. service	Spécif. service	Spécif. service	Spécif. service	Spécif. service	Spécif. service	Spécif. service
Contraintes sur le modèle de communication	Synchrone, N-entités, fiable	Synchrone, N-entités, fiable	Synchrone, N-entités, fiable ou non	(A)synchrone, N-entités, fiable ou non	Synchrone, 2-entités, fiable ou non	Asynchrone, N-entités, fiable ou non	Asynchrone, N-entités, fiable ou non	Synchrone, N-entités, fiable
Modèle de spécification	Sous-ensemble de Lotos	Sous-ensemble de Lotos	Très proche de Lotos de base	Logique épisté-mique/Automates	Automates à états finis	Automates à états finis	Automates temporisés	Automates à états étendus.
Correction du protocole	Logique et sémantique (sans preuve)	Logique et sémantique (sans preuve)	Logique et sémantique (avec preuve)	Logique et sémantique	Logique et sémantique (avec preuve)	Logique et sémantique (avec preuve)	Logique et sémantique (sans preuve)	Logique et sémantique (sans preuve)
Contexte de la synthèse	1. Flot de contrôle: C1 à C4	1. Flot de contrôle: C1 à C4 2. Flot de données: D1 3. Aspects structurels	1. Flot de contrôle: C1 à C4, Interruption et Récursivité	1. Flot de contrôle: C1 et C2 2. Aspects structurels	1. Flot de contrôle: C1 à C4	1. Flot de contrôle: C1 et C2	1. Flot de contrôle: C1 et C3 2. Aspects temporels	1. Flot de contrôle: C1, C2 et C6 2. Flot de données: D1 à D3
Particularités	1. Basée sur le concept d'arbre syntaxique attribué	1. Basée sur le concept d'arbre syntaxique attribué	1. Basée sur le concept d'arbre syntaxique attribué	1. Basée sur la construction du graphe global des états de connaissance	1. Basée sur la projection d'automates 2. Engendre des bouclages infinis	1. Basée sur la projection d'automates	1. Basée sur la projection d'automates	1. Basée sur un principe de simulation de transitions

TAB. 1.1 – *Comparaison des méthodes de synthèse orientées service*

- Parmi les méthodes que nous avons abordées, la majorité sont limitées à la synthèse du flot de contrôle. Seules les méthodes de GOTZHEIN [GOT 90a] et de HIGASHINO [HIG 93] introduisent la synthèse du flot de données respectivement par l'intermédiaire de primitives de service paramétrées et de données virtuellement globales.
- La méthode de HIGASHINO [HIG 93] se distingue des autres par l'introduction du concept de registre qui permet de modéliser, en plus du flot de données, des ressources disponibles à travers les différents points d'accès au service.
- Les méthodes [KHO 94a], [KHO 94b] et [NAT 95] sont caractérisées par la prise en compte de contraintes temporelles dans la spécification des services et des protocoles, mais ne traitent pas la synthèse du flot de données.
- Aucune des méthodes abordées ne traite les problèmes de panne, telle que la panne d'un site.
- Seules les méthodes de AMYAY [AMY 91] et de MANNA [MAN 84] prennent en entrée des spécifications écrites dans un modèle logique.
- Certaines méthodes supposent que le médium de communication est fiable. Ces méthodes sont adaptées à la construction de protocoles de couches hautes. Par contre d'autres méthodes synthétisent des fonctions de recouvrement d'erreurs de transmission. Celles-ci sont destinées à la construction de protocoles de couches basses.

1.5 Approche non orientée service

Dans cette section, nous présentons les principales méthodes de synthèse dont le point de départ est une spécification partielle du protocole. Ces approches ne considèrent pas le service fourni par le protocole.

1.5.1 Synthèse de protocoles à deux entités dans un modèle d'automates

1.5.1.1 Problématique

Le problème abordé dans cette approche consiste à trouver des règles permettant de construire deux automates communicants et logiquement corrects. Ces règles sont utilisées pour la conception de protocoles à deux entités.

1.5.1.2 Méthodes existantes

Une première méthode qui s'est penchée sur le problème est due à ZAFIROPULO *et. al.* [ZAF 80]. Ce travail constitue le premier pas qui a été effectué vers la synthèse de protocoles. Cette approche présente une méthodologie interactive pour la synthèse d'entités de protocole communiquant via des canaux FIFO (First In First Out) et fiables. Les entités de protocole sont modélisées par des machines à états finis communicantes. La synthèse est basée sur un ensemble de règles de production et s'effectue selon les étapes suivantes :

1. Le concepteur ajoute une transition d'émission de message à l'une des deux machines.

2. En utilisant les règles de production, l'algorithme de synthèse rajoute la transition de réception correspondante de telle sorte que les spécifications soient libres de toute réception non spécifiée ou de situation d'interblocage.

3. Le concepteur demande au système de vérifier si les transitions ajoutées ne conduisent pas à un dépassement de capacité du canal de communication.

Le processus de synthèse s'arrête lorsqu'il n'y a plus de transitions d'émission à ajouter pour l'une ou l'autre des deux machines. Dans ce cas, on considère que les deux machines sont complètes et que la communication entre elles est sans interblocage, sans réceptions non spécifiées et sans transitions non exécutables. Les principales limites de cette approche peuvent se résumer comme suit :

- les étapes de synthèse sont coûteuses en temps puisqu'elles sont basées sur une procédure d'approximations successives où des transitions sont ajoutées puis vérifiées.

- de plus, la vérification du dépassement de capacité de l'étape 3 est basée sur l'analyse d'accessibilité qui constitue une négation même des objectifs définis par la synthèse.

Afin d'éliminer les approximations successives de la méthode précédente, GOUDA propose dans [GOU 84] une technique permettant de synthétiser automatiquement une entité de protocole partant de la spécification de son entité homologue. Cette méthode démarre d'une machine à états finie incomplète modélisant une seule entité de protocole M et consiste à synthétiser deux machines à états M' et N' sans erreurs logiques et communiquant via deux FIFOs unidirectionnelles. La machine M est incomplète en ce sens qu'elle peut conduire à des erreurs logiques. M' est construite à partir de M en ajoutant des transitions de réception. L'algorithme de construction garantit que les spécifications M' et N' sont sans erreurs logiques. Par rapport à la méthode précédente, GOUDA propose une approche déterministe qui évite les approximations successives et les retours en arrière. Néanmoins, sa généralisation à plusieurs entités de protocole (plus de deux) n'est pas directe. De plus, comme dans la méthode précédente, on ne vérifie pas si l'ensemble des deux machines à états satisfait un comportement externe attendu. En d'autres termes, on ne tient pas compte du concept de service.

RAMAMOORTHY [RAM 85] a développé une méthode de synthèse qui va dans le même sens que celle de GOUDA [GOU 84] mais utilisant un modèle de réseaux de Petri et basée sur le graphe d'accessibilité. Le modèle choisi lui offre la possibilité de spécifier des comportements concurrents au sein d'une même entité de protocole. La synthèse consiste alors en cinq étapes :

1. Description d'une entité de protocole locale EP_1 dans le modèle des réseaux de Petri généraux.

2. Construction du graphe des états accessibles G_1 associé à EP_1.

3. Vérification de la correction logique de EP_1 sur le graphe G_1.

4. Construction du graphe des états accessibles G_2 de l'entité homologue EP_2 partant du graphe G_1. Ceci est fait en utilisant un ensemble de règles de transformation basées sur la dualité émission/réception de messages.

5. Construction de l'entité homologue EP_1 à partir du graphe G_2.

Cette approche a été étendue ultérieurement [RAM 86] pour inclure des mécanismes de contrôle d'erreurs dans le cas où les canaux de communication sont non fiables. L'avantage de cette approche est qu'elle produit automatiquement une entité paire correspondant à une entité locale donnée afin que leur interaction soit sans erreurs logiques. Toutefois, elle exige que l'entité locale donnée soit correcte en passant par une analyse exhaustive. De plus, elle est strictement limitée aux protocoles à deux entités. Cette méthode est supportée par un outil logiciel appelé APS (Automatic Protocol Synthesizer).

ZHANG *et. al.* [ZHA 88] ont proposé plus tard une approche de synthèse interactive basée sur le graphe d'accessibilité. Cette méthode est analogue à celle de ZAFIROPULO [ZAF 80] mais fournit une terminaison automatique de la synthèse ce qui n'est pas le cas pour [ZAF 80]. Partant de spécifications informelles, l'algorithme de synthèse construit d'abord, de manière interactive, un graphe d'états global du protocole, et produit ensuite les deux entités de protocole par décomposition de ce graphe. La construction du graphe est guidée par un ensemble de règles qui garantissent la correction logique. L'inconvénient majeur de cette approche réside dans le passage par la construction du graphe d'accessibilité associé au protocole. De plus, la généralisation à plus de deux entités de protocole n'est pas directe et risque d'augmenter encore la taille du graphe des états accessibles. Cette méthode a été reprise dans [SHI 91] pour la conception d'un environnement logiciel pour la synthèse de protocoles. Les auteurs ont utilisé deux approches différentes pour la conception d'un tel environnement : la première est basée sur des techniques d'intelligence artificielle, KSPS (Knowledge based System for Protocol Synthesis) et la seconde utilise un paradigme de programmation procédural, SEPS (Software Environment for Protocol Synthesis). Les auteurs ont ainsi montré que l'environnement KSPS possède des avantages sur SEPS en termes d'extensibilité et de maintenance.

1.5.2 Synthèse interactive par raffinements de réseaux de Petri

1.5.2.1 Problématique

Le modèle des réseaux de Petri est un outil mathématique très puissant pour la description des protocoles de communication et des systèmes distribués. Néanmoins, lorsque la description d'un système atteint une complexité assez élevée, il est souvent très difficile, voir impossible de vérifier les bonnes propriétés du système, telles que la vivacité et le caractère borné du réseau de Petri modélisant le système. Il est alors souhaitable de disposer de règles interactives permettant la construction de réseaux de Petri tout en garantissant les bonnes propriétés.

1.5.2.2 Méthodes existantes

Dans cette approche du problème de synthèse, [YAW 87] propose une nouvelle technique de synthèse interactive basée sur des règles spéciales de raffinement de réseaux de Petri appelées *knitting rules* (règles de tricotage). Partant de spécifications en réseaux de Petri d'un ou de plusieurs processus de base modélisant le protocole à un niveau d'abstraction très élevé, le concepteur introduit de nouveaux détails de manière incrémentale en utilisant

les règles de tricotage. Ces règles se divisent en deux types : règle TT et règle PP. La règle TT concerne l'ajout de chemins entre transitions seulement, par contre la règle PP concerne l'ajout de chemins entre places seulement. Ces règles garantissent qu'à chaque étape de raffinement, les spécifications obtenues conservent les bonnes propriétés telles que le caractère borné et la vivacité. Cela suppose, bien entendu, que les spécifications initiales possèdent ces propriétés. L'avantage principal de cette approche est qu'elle permet de concevoir des systèmes complexes en assurant la correction de manière incrémentale. Néanmoins, comme toutes les méthodes non orientées service, elle doit être complétée par une phase de vérification de la conformité des spécifications finales vis-à-vis des services attendus. Pour plus de détails sur le concept de règles de tricotage, se reporter aux travaux récents de CHAO et WANG [CHA 94a].

1.5.3 Discussion sur les méthodes non orientées service

L'inconvénient majeur des méthodes de synthèse non orientées service est qu'elles ne considèrent pas le comportement externe du système décrit. En d'autres termes, elles ne font aucune référence au concept de service. Par conséquent, elles doivent impérativement être complétées par une phase de vérification de la correction sémantique du protocole construit. A notre avis, l'utilisation de méthodes non orientées service serait plus adéquate dans un contexte où le protocole que l'on souhaite synthétiser possède certains composants prédéfinis et supposés corrects. C'est le cas du problème de la synthèse de convertisseur de protocoles.
Le Tableau 1.2 regroupe les principales distinctions entre les méthodes de synthèse non orientées service selon les critères définis dans la section 1.3.

1.6 Approche mixte

1.6.1 Problématique

Dans cette section, nous présentons une autre approche du problème de synthèse dont le point de départ regroupe à la fois les spécifications des services et les spécifications d'une (ou plusieurs) entité(s) de protocole. Le problème revient alors à synthétiser le ou les modules restants du protocole de telle sorte que l'ensemble de tous les modules soit conforme aux spécifications de service.

1.6.2 Méthodes existantes

Nous considérons dans ce contexte la méthode de Merlin [MER 83] qui est la première ayant introduit le concept de service dans le processus de synthèse. Cette méthode considère le problème de synthèse suivant : *Étant données n-1 machines à états finies communicantes ($n > 2$), telles que l'une d'elles modélise le médium de communication, et les spécifications de service, il s'agit de synthétiser la n^{me} machine de telle sorte que le service fourni par l'ensemble des n machines soit conforme aux spécifications de service.* Une hypothèse est faite sur le modèle de communication, à savoir que la communication entre les entités de protocole est fortement synchronisée ce qui implique que le délai entre l'émission et la réception d'un message est ignoré.

	[ZAF 80]	[GOU 84]	[RAM 85]	[ZHA 88]	[YAW 87]
Point départ	2 entités incomplètes	1 entité incomplète	1 entité complète et correcte	Spécif. informelles	Spécif. d'un ou plusieurs processus de base
Contraintes sur le modèle de communication	Asynchrone, 2-entités, fiable	Asynchrone, 2-entités, fiable	Asynchrone, 2-entités, fiable ou non	Asynchrone, 2-entités, fiable	Asynchrone, n-entités, fiable
Modèle de spécification	Automates à états finis	Automates à états finis	Automates à états finis	Automates états finis	Réseaux de Petri généraux
Mode d'interaction	Interactif	Automatique	Automatique	Interactif	Interactif
Correction du protocole	Logique (sans preuve)	Logique (sans preuve)	Logique (sans preuve)	Logique (sans preuve)	Logique (avec preuve)
Particularités	1. Basée sur la dualité émission/réception 2. Procédure par approximations successives 3. Vérification par analyse d'accessibilité	1. Basée sur la dualité émission/réception 2. Procédure déterministe 3. Strictement limitée à 2 entités de protocole	1. Basée sur la dualité émission/réception 2. Procédure déterministe 3. Entité en entrée doit être correcte 4. Strictement limitée à 2 entités	1. Basée sur la construction interactive du graphe d'accessibilité global 2. Strictement limitée à 2 entités	1. Basée sur des règles de raffinement de réseaux de Petri 2. Assure la correction logique de manière incrémentale

TAB. 1.2 – *Comparaison des méthodes de synthèse non orientées service*

La solution du problème consiste en une formule qui définit le module de communication recherché à partir des autres modules. Les auteurs font observer que le module résultant peut avoir des transitions superflues et par conséquent conduire à des blocages, et des transitions non exécutables. Pour y remédier, les auteurs proposent une procédure de révision du module construit. Cette révision est basée sur l'exploration des états accessibles afin de supprimer certaines transitions superflues d'où le long délai de traitement. Par ailleurs, cette méthode est limitée à la conception de systèmes fortement synchronisés. Un des principaux mérites de cette approche est qu'elle supprime la nécessité d'une phase postérieure de vérification sémantique, qui limite les méthodes non orientées service.

Cette approche a été appliquée dans [KEL 93] à la synthèse de convertisseurs de protocoles. Elle a été également reprise dans [KHO 94c] en utilisant la théorie du contrôle des systèmes à événements discrets. Ainsi, un module dit contrôleur est synthétisé à partir du système que l'on souhaite contrôler et de la spécification du service désiré. La méthode considère successivement les cas où le système est totalement et partiellement observable. Les auteurs font observer que la méthode de Merlin [MER 83] exige implicitement que les événements contrôlables doivent être observables, ce qui limite son application. Ils considèrent ainsi que leur méthode supprime cette restriction.

Comme pour les méthodes non orientées service, l'approche mixte est bien adaptée à des domaines de conception où certains modules du système sont connus a priori. C'est le cas de la synthèse de convertisseurs de protocoles que nous abordons dans la section suivante.

1.7 Synthèse de convertisseurs de protocoles

La construction de convertisseur de protocoles par les méthodes formelles a été suggérée par GREEN en 1986 [GRE 86]. L'objectif principal des méthodes formelles est la construction automatique des spécifications d'un convertisseur à partir de spécifications formelles des protocoles que l'on souhaite faire interopérer. L'intérêt de telles méthodes réside dans la construction de convertisseurs corrects à un coût faible.
Étant donnés deux protocoles à deux entités: $P = (P_0, P_1)$ et $Q = (Q_0, Q_1)$, le problème abordé dans cette section consiste à trouver, si elle existe, une entité C permettant le dialogue de P_0 avec Q_1 (ou de P_1 avec Q_0). Cette entité joue le rôle de convertisseur entre P_0 et Q_1. Le protocole à trois entités obtenu (P_0, C, Q_1) devra être correct logiquement et sémantiquement.
Les méthodes de construction de convertisseurs de protocoles peuvent être considérées comme des méthodes de synthèse de protocoles telles que le point de départ inclut les spécifications des protocoles que l'on souhaite faire coopérer, et éventuellement la spécification du service requis.

1.7.1 Méthodes existantes

CALVERT et LAM ont proposé dans [CAL 89] une technique basée sur le calcul du *quotient* de deux spécifications. Les protocoles à convertir $P = (P_0, P_1)$ et $Q = (Q_0, Q_1)$, et le service devant être fourni par la conversion S sont modélisés par des automates. Le problème est

de faire interopérer P_0 avec Q_1 pour fournir le service S. Le principe de cette méthode est d'interposer un convertisseur C entre les entités P_0 et Q_1 de telle sorte que le protocole final (P_0, C, Q_1) réalise le service S. Le résultat de l'interaction de deux automates étant défini par une opération de composition parallèle (notée ||). Cette opération construit tous les entrelacements possibles des séquences de transitions des deux automates. La première phase de cette méthode consiste à construire le plus grand automate C tel que la composition parallèle $P_0||C||Q_1$ puisse simuler le service S, c'est-à-dire que pour toute séquence de S, il existe une séquence équivalente (de manière observationnelle) dans $P_0||C||Q_1$. L'automate C est construit de manière inductive à partir des automates P_0, Q_1 et S. Dans la phase suivante, C subit les suppressions successives des états qui peuvent l'amener dans des situations irrégulières, telles que l'interblocage ou la réception non spécifiée.
Un convertisseur construit par cette méthode est correct, mais le calcul effectué pendant la construction de C est très complexe. Le nombre d'états de C croît de manière exponentielle en fonction du nombre d'états des automates P_0, Q_1 et S. Par ailleurs, cette méthode peut engendrer, lors de la première phase, des états et des transitions qui ne sont pas utiles pour la conversion mais qui peuvent, lors de la deuxième phase, conduire au cas dégénéré (i.e., automate sans transitions).

YAO *et. al.* ont tenté de diminuer la complexité de calcul d'un convertisseur en proposant une méthode dite *modulaire* [YAO 90]. Dans cette méthode, un protocole est divisé en plusieurs phases réalisant des fonctions différentes. Un convertisseur partiel est construit pour chaque couple de fonctions compatibles. Les convertisseurs partiels sont ensuite assemblés en un seul convertisseur. Un protocole est modélisé par un ensemble d'automates communicants. On considère deux protocoles à deux entités chacun $P = (P_0, P_1)$ et $Q = (Q_0, Q_1)$, deux fonctions compatibles $F = (F_0, F_1)$ et $G = (G_0, G_1)$ respectivement de P et Q, et une spécification N qui est un automate définissant la correspondance entre les PDUs des deux fonctions. Le convertisseur fonctionnel est construit en 3 étapes :

1. construire un convertisseur fonctionnel U sans restriction sur les séquences de PDUs de F_1 et G_0, c'est-à-dire : $U = F_1||G_0$,

2. combiner U avec la spécification de correspondance N. On obtient un automate T restreint aux correspondances définies dans N,

3. supprimer les états et transitions de T de telle sorte que le protocole (F_0, T, G_1) soit sans erreurs logiques (interblocage, bouclage, etc.).

Ces trois étapes permettent la construction d'un convertisseur correct T entre les entités F_0 et G_1 correspondant aux fonctions F et G. Ce traitement est réitéré pour tous les couples de fonctions compatibles des protocoles P et Q. Les convertisseurs fonctionnels obtenus sont ensuite combinés pour construire le convertisseur global entre les entités P_0 et Q_1.
Cette méthode tente de réduire la complexité de calcul par décomposition de la taille du problème. Néanmoins, la correction sémantique n'est pas intégrée au sein de la méthode, elle doit être, par conséquent, vérifiée après la construction du convertisseur.
Notons que l'idée de la décomposition du protocole peut être appliquée à d'autres méthodes.

RAJAGOPAL [RAJ 91] propose une méthode basée sur les traces des automates à états finis. Le problème étant de permettre l'interopérabilité de deux entités de protocole P_0 et Q_1 qui font partie de deux protocoles différents $P = (P_0, P_1)$ et $Q = (Q_0, Q_1)$. La démarche consiste dans un premier temps à établir une relation de correspondance entre les PDUs de P_0 et Q_1. A chaque réception de PDU dans P_0 correspond une émission de PDU dans Q_1 et vice versa. Il s'agit ensuite de construire un convertisseur H qui doit :

- simuler P_1 pour dialoguer avec P_0,
- simuler Q_0 pour dialoguer avec Q_1,
- et convertir tous les messages qu'il reçoit selon la correspondance établie dans la première phase.

La construction du convertisseur H commence par la détermination de l'espace de conversion maximal C, c'est-à-dire de tous les entrelacements possibles des traces d'exécution de P_1 et Q_0. L'automate C est donc obtenu par une opération similaire à la composition parallèle de la méthode quotient : $C = P_1 \| Q_0$. Le convertisseur H est un sous-automate de C contenant les séquences d'exécution dites *valides*, c'est-à-dire les séquences qui respectent la relation de correspondance établie précédemment. Cette méthode est complétée par une phase de vérification de la correction du convertisseur construit.
La synthèse du convertisseur H passe par la construction des traces de C et l'extraction des traces valides. Ceci rend la méthode très complexe à appliquer pour des protocoles réels. La complexité en temps de cette méthode augmente en factorielle par rapport au nombre de messages traités par une entité de protocole.

NOOSONG propose dans [NOO 92] une approche basée sur le modèle CSP et intégrant la spécification du service requis. Dans cette méthode, les objets du système de conversion sont modélisés par des processus du modèle CSP (Communicating Sequential Processes) [HOA 85]. La modélisation consiste à faire abstraction des détails internes de l'objet. Le comportement interne d'un objet est défini par l'ensemble des séquences d'actions pouvant être effectuées par cet objet (c'est-à-dire, l'ensemble des traces de l'objet). La spécification d'un service réalisé par un protocole à deux entités est définie par un processus S ayant deux points d'accès I_0 et I_1 ; chacun représente l'ensemble des primitives de service pouvant être activées à ce point. Un protocole à deux entités est modélisé par la composition de 4 processus P_0, P_1, L et L'. Les deux premiers processus représentent les deux entités du protocole et les deux derniers, un médium de communication bidirectionnel. Le modèle CSP permet de construire un nouveau processus en mettant en parallèle plusieurs processus. Ainsi, le protocole P peut être représenté par le processus composite $P = P_0 \| L \| L' \| P_1$. Par conséquent, le comportement de ce protocole est donné par l'ensemble des traces du processus P. Notons aussi que le modèle CSP permet de restreindre le comportement d'un processus aux événements que l'on souhaite observer. Ainsi, on peut comparer le comportement de P au service S en cachant les événements de communication dans P.
Le problème de conversion est énoncé de la manière suivante : *étant donnés deux protocoles* $P = (P_0, L, L', P_1)$ *et* $Q = (Q_0, R, R', Q_1)$ *et le service demandé* S, *il s'agit de trouver un processus* C *tel que le processus composite :*

$$X = P_0 \| L \| L' \| C \| R \| R' \| Q_1$$

supporte le service S (c'est-à-dire qu'il y a équivalence entre les séquences observables de S et celles admises par X). Partant des deux protocoles P et Q, le principe consiste à cacher les événements observables de P_1 et Q_0 dans X. On obtient, alors, deux processus : $P_0||L||L'||C_1$ et $C_2||R||R'||Q_1$, où C_1 et C_2 sont dits des *processus muets.* Les processus C_1 et C_2 sont construits à partir des traces des processus $P_0||L||L'$ et $Q_0||R||R'$. A notre avis, il serait plus simple de les construire directement à partir des processus P_1 et Q_0.
L'étape suivante consiste à synchroniser C_1 et C_2 pour obtenir le processus convertisseur. La synchronisation est faite de manière heuristique en respectant des règles de causalité entre les événements. Cette méthode est basée sur le calcul de traces d'un processus CSP qui peut être complexe pour des processus de grande taille. Elle a été étendue ensuite pour la construction de convertisseur à partir de spécifications LOTOS. La complexité de la méthode n'a pas été évaluée.

SALEH *et. al.* proposent dans [SAL 95] une technique qui va dans le même sens que [NOO 92], elle est basée sur la détermination du plus grand service commun fourni par les deux protocoles à convertir. Les protocoles et les services sont spécifiés par des automates à états finis. La méthode consiste en 4 étapes: (1) Calcul du plus grand service commun à partir des définitions des services des deux protocoles. (2) Construction de deux ensembles de traces décrivant les événements qui se produisent dans chaque protocole et qui contribuent à la réalisation du plus grand service commun. (3) Synchronisation des deux ensembles de traces en se basant sur l'heuristique définie dans [NOO 92]. (4) Synthèse d'une machine à états finie modélisant le convertisseur à partir des traces de l'étape précédente. Cette méthode fournit des convertisseurs corrects logiquement et sémantiquement. Cependant, l'étape (2) de l'algorithme de synthèse peut être très complexe pour des protocoles de taille réelle.

Récemment, PEYRAVIAN et LEA ont tenté de formaliser davantage les correspondances entre les messages à convertir, ils proposent ainsi dans [PEY 96] un algorithme de construction de convertisseurs logiquement corrects partant de descriptions formelles des correspondances entre les messages à convertir. Cette approche est très similaire à [RAJ 91]. Elle propose, en plus, une technique pour obtenir les correspondances entre les messages à convertir. Toutefois, elle ne considère pas la description du service requis et devrait donc être complétée par une phase de vérification de la correction sémantique du convertisseur.

1.7.2 Discussion

Après avoir passé en revue les principales méthodes de conversion existantes, nous présentons dans ce qui suit quelques observations générales sur les techniques de construction de convertisseur de protocoles, et nous terminons par un tableau comparatif (cf. Tableau 1.3).

- Relativement aux méthodes de synthèse de protocoles, les méthodes de synthèse de convertisseurs doivent considérer à la fois des contraintes externes (la spécification des services requis) et des contraintes internes (les spécifications des entités que l'on souhaite faire coopérer et éventuellement des informations de correspondances entres ces deux entités) ce qui augmente leur complexité en calculs.

- Nous constatons que les méthodes qui ne font pas intervenir des connaissances initiales sur la conversion engendrent une complexité en calculs très élevée, comme c'est le cas

	Quotient [CAL 89]	*Trace* [RAJ 91]	*Modulaire* [YAO 90]	*CSP* [NOO 92]
Point départ	Services +Protocoles	Protocoles +correspondances entre PDUs	Décomposition fonctionnelle des protocoles +correspondances entre fonctions	Services +protocoles
Modèle de spécification	Automates à états finis	Automates à états finis	Automates à états finis	CSP/LOTOS
Mode d'interaction	Automatique	Automatique	Automatique	Partiellement automatique
Correction	Par construction (logique et sémantique)	Postérieure (logique)	Par construction (logique)	Par construction (logique et sémantique)
Complexité	Exponentielle (nombre d'états des entrées)	Factorielle (nombre de PDUs)		
Particularités	1- Génère des états et transitions superflues 2- Convertisseur construit de zéro par induction	1- Convertisseur issu d'un espace de conversion maximal 2- Basée sur le calcul de traces d'un automate	1- Réduit la complexité par décomposition 2- Convertisseur fonctionnel issu d'un espace de conversion max.	1- Basée sur le calcul de traces de processus CSP 2- Partie heuristique

TAB. 1.3 – *Comparaison des méthodes de synthèse de convertisseurs de protocoles*

de la méthode quotient [CAL 89].

- Dans les méthodes de conversion existantes, nous pouvons distinguer trois principes de base :

  1. *La construction d'un espace de conversion maximal à partir des entités de protocole homologues à celles devant coopérer.* Le convertisseur est obtenu en réduisant, par une technique appropriée, l'espace de conversion maximal. C'est le cas des méthodes [YAO 90], [RAJ 91] et [PEY 96].
  2. *La construction heuristique d'un espace de conversion initial.* Dans ce cas, le convertisseur est obtenu en complétant cet espace de conversion. C'est le cas de la méthode d'OKUMURA [OKU 86] basée sur la construction d'un point de conversion initial dit "conversion seed".
  3. *La résolution d'équation d'équivalence entre spécifications de service et spécifications du protocole de conversion (constitué des entités à faire interopérer et du*

convertisseur). Dans ce cas, il s'agit de trouver un convertisseur tel que l'équivalence soit assurée ; c'est le cas des méthodes [CAL 89], [NOO 92] et [SAL 95].

A notre avis, il serait intéressant de combiner ces principes afin d'aboutir à une méthode qui garantisse les services requis et qui ne soit pas très complexe en calculs.

1.8 Conclusion

La synthèse de protocoles de communication est une nouvelle approche de conception dont l'objectif est de produire des protocoles corrects par construction. Cette approche représente une alternative importante aux approches classiques basées sur l'analyse. En effet, les méthodes analytiques sont destinées à la validation de protocoles existants et n'offrent, par conséquent, aucune démarche à suivre pour la conception de nouveaux protocoles. De plus, elles sont fortement limitées par le problème de l'explosion combinatoire. De ce fait, de nombreux travaux commencent à s'orienter vers cette nouvelle direction de recherche.

Dans ce chapitre, nous avons tenté de présenter un état de l'art sur les principales méthodes de synthèse de protocoles qui existent dans la littérature. Nous avons présenté dans un premier temps une vision unifiée des différentes étapes de développement d'un protocole selon une approche de synthèse que nous avons comparé à l'approche analytique. Nous avons dégagé ensuite un ensemble de critères d'évaluation d'une méthode de synthèse. Partant de ces critères, nous avons proposé une classification des principales méthodes existantes.
En considérant le point de départ du processus de synthèse comme critère clé, nous avons isolé trois approches différentes du problème de synthèse. La première approche consiste à dériver les spécifications du protocole en partant des spécifications complètes des services (synthèse orientée service). La deuxième approche suppose la connaissance d'une ou plusieurs entités de protocole et consiste à synthétiser le reste (synthèse non orientée service). La dernière approche considère à la fois des contraintes externes (spécifications de service) et des contraintes internes (spécifications d'une ou plusieurs entités de protocole) et consiste à dériver les entités de protocole manquantes (synthèse mixte).

A notre avis, l'efficacité d'une méthode de synthèse orientée service se mesure essentiellement par la puissance de son modèle de spécification de service et par son pouvoir de synthèse. L'une des difficultés principales de la synthèse automatique de protocoles réside dans l'opposition de la puissance d'expression du modèle de spécification des services et le pouvoir de synthèse. En effet, le modèle d'expression doit être riche afin de pouvoir spécifier toutes les fonctionnalités nécessaires d'un service ou d'une partie de protocole, alors qu'une méthode de synthèse exige l'utilisation d'un modèle mathématique relativement simple (automates à états finis, réseaux de Petri non interprétés, etc.) afin de conduire à des techniques de transformation de spécifications automatisables et préservant les critères de correction. Chacune des méthodes existantes prend en compte certains aspects dans la spécification des services et ignore d'autres aspects. Il serait intéressant de trouver un bon compromis. Par ailleurs, les méthodes non orientées service ne font aucune référence au concept de service et doivent, par conséquent, être complétées par une phase de correction sémantique.
Les méthodes mixtes constituent une amélioration des méthodes non orientées service en

considérant en plus le concept de service dans le processus de synthèse.
La synthèse non orientée service et la synthèse mixte sont bien adaptées à la conception de systèmes soumis à des contraintes internes. C'est le cas de la conception de convertisseurs de protocoles de communication. Nous avons étudié, dans ce contexte, les principales méthodes de synthèse de convertisseurs.

Pour terminer, nous citons certaines directions qui nous semblent intéressantes à aborder :

- Proposition d'un "bon" compromis entre la puissance d'expression des services et le pouvoir de synthèse des protocoles. Ceci permettra d'élargir le champ d'application d'une technique de synthèse.
- L'intégration d'une méthodologie de synthèse dans un environnement logiciel de développement de protocoles de communication. Certains logiciels existent mais sont toujours à l'état expérimental [SHI 91] [CHA 94b].
- Synthèse automatique de protocoles dans une TDF standard et orientée implantation telle que le langage Estelle. Ceci permettra de faciliter l'implantation des protocoles.
- Démontrer l'intérêt de la synthèse de protocoles en proposant des applications réelles.
- L'intégration d'une technique d'optimisation du nombre de messages échangés entre les entités de protocole synthétisées. Ceci concerne les performances des protocoles synthétisés.
- La synthèse de protocoles avec des contraintes de temps.
- La réduction de la complexité des calculs dans la synthèse de convertisseurs de protocoles.

Chapitre 2

Synthèse de protocoles par raffinement de réseaux de Petri

Résumé

Dans ce chapitre nous proposons une nouvelle méthode de synthèse de spécifications de protocoles à partir de spécifications de service dans un modèle de réseaux de Petri interprétés [KAH 96]. Notre approche traite à la fois le flot de contrôle, le flot de données, le choix distribué et les contraintes de cohérence de données. En d'autres termes, nous proposons un "bon" compromis entre le pouvoir d'expression du modèle de spécification de service et le pouvoir de synthèse.

2.1 Introduction

Les protocoles de communication jouent un rôle de plus en plus important dans les systèmes distribués. La construction de logiciels de communication fiables apparaît donc comme une phase cruciale dans l'exploitation efficace et performante d'un système distribué. Un protocole de communication est défini comme un ensemble de règles qui gouvernent l'échange de données entre un ensemble de composants répartis afin de fournir des services aux utilisateurs. Le concept de service constitue l'un des plus importants concepts architecturaux des logiciels de communication [VIS 86]. La spécification formelle des services de communication fournit une définition non ambiguë des besoins des utilisateurs. Cette spécification peut être utilisée par le concepteur pour vérifier si le protocole fournit les services pour lesquels il a été conçu. La relation entre protocole et service est définie comme suit : un système de communication peut être vu à un niveau d'abstraction élevé comme un fournisseur de service à des utilisateurs potentiels. Un utilisateur accède au système à travers des points d'accès au service (SAPs). A ce niveau, le système est vu comme une boîte noire. A un niveau d'abstraction plus bas, les services sont fournis par un ensemble d'entités de protocole (EPs) qui coopèrent en utilisant les services existants.

Parmi les méthodes formelles de développement de protocoles de communication, nous distinguons deux grandes catégories : les méthodes classiques basées sur l'analyse et les méthodes basées sur la synthèse.

Dans les techniques analytiques, le cycle *détection d'erreurs et correction* est répété jusqu'à ce que le protocole soit valide. Ces techniques sont souvent coûteuses en temps. De plus, la

phase de validation passe généralement par une analyse exhaustive dont le problème principal est l'explosion combinatoire (pour un tour d'horizon se reporter à [YUA 88]).
Les techniques de synthèse consistent à construire de manière directe et systématique un protocole de communication sans erreur. Nous distinguons selon le point de départ du processus de synthèse deux approches différentes: les approches non orientées service et les approches orientées service. Les approches non orientées service sont basées sur la description partielle ou complète d'une (ou plusieurs) entité(s) de protocole pour générer la spécification duale de l'autre (ou des autres) entité(s) de protocole. La synthèse est basée dans ce cas sur la dualité entre émission et réception des messages. Ces approches ne considèrent pas le service fourni par le protocole, et doivent par conséquent être complétées par une procédure de validation. Les approches orientées service partent d'une description complète du service de communication requis par les utilisateurs et consistent à dériver de manière automatique un protocole correct. Par conséquent, elles ne sont pas coûteuses en temps de conception et n'exigent pas de phase de vérification. Notre approche entre dans cette catégorie.

L'efficacité d'une approche de synthèse orientée service peut se mesurer essentiellement par la puissance d'expression de son modèle de spécification de service. Dans une architecture distribuée, il est souhaitable que le modèle de spécification des services permette d'exprimer :

(a) des événements parallèles de façon explicite, puisque le parallélisme est inhérent aux architectures distribuées,

(b) des primitives de service paramétrées.

(c) des variables "virtuellement" globales afin de fournir un niveau d'abstraction élevé dans la description des flots de données,

(d) des accès concurrents aux variables globales,

(e) la duplication de données pour augmenter le degré de disponibilité des données, et

(f) des structures sélectives distribuées afin de pouvoir spécifier des choix déterministes ou non déterministes entre plusieurs primitives de service en faisant abstraction de leur localisation.

Cet ensemble de caractéristiques définit une classe importante de services permettant d'exploiter efficacement un système distribué à un niveau d'abstraction élevé.

Dans le contexte de la synthèse orientée service, plusieurs méthodes ont été proposées dans la littérature, un état de l'art est présenté dans le chapitre 1. On distingue les approches basées sur LOTOS [BOC 86] [KHE 89] [GOT 90a] [KAN 93a] [KHO 95] [NAT 95] et les approches basées sur les automates à états finis [CHU 88] [HIG 93] [KHO 94a] [KAK 94] [SAL 90]. Les approches basées sur LOTOS permettent de décrire le parallélisme dans la spécification de service, mais la plupart sont limitées à la synthèse du flot de contrôle [BOC 86] [KHE 89] [KAN 93a] [KHO 95]. Seule la méthode de BOCHMANN et GOTZHEIN [GOT 90a] introduit des primitives de service paramétrées mais impose des restrictions fortes sur la manipulation des paramètres, telles que l'assignation unique et l'interdiction des accès concurrents.

Les approches basées sur les automates à états finis sont principalement limitées par leur caractère séquentiel dans la description d'un service. La plupart sont aussi limitées à la synthèse du flot de contrôle. La méthode de HIGASHINO *et. al.* [HIG 93] traite le flot de données dans la spécification de service en définissant un ensemble de variables globales (appelées registres) pré-allouées aux différentes entités de protocoles. Cette méthode a été ensuite étendue dans [YAM 95] par l'expression du parallélisme en utilisant un modèle de réseaux de Petri. Néanmoins, la puissance d'expression de son modèle de spécification de services reste limitée car elle ne supporte pas les structures sélectives distribuées, et elle impose une restriction sur la manipulation des registres, à savoir que les accès ne peuvent pas être concurrents.

Pour la classe des services souhaitables décrits ci-dessus, il n'existe pas d'algorithme de synthèse permettant de dériver automatiquement un protocole correct. Nous présentons alors dans ce chapitre une méthode de synthèse pour cette classe de service [KAH 96]. Notre approche est basée sur un concept de règles de raffinement qui lui offre un contexte de synthèse facilement extensible. En effet, l'extension du modèle de spécification de service entraîne simplement un ajout de nouvelles règles de synthèse sans remettre en cause celles qui existent. Nous définissons, dans ce chapitre, un modèle de réseaux de Petri interprétés (IPN) pour la spécification de services et de protocoles. Étant données une spécification de service et une architecture distribuée cible, le problème que nous nous proposons de résoudre est de dériver automatiquement les spécifications d'un protocole qui satisfassent d'une part les contraintes de correction et d'autre part les contraintes de cohérence sur les données globales.

Dans la section suivante nous présentons notre modèle de spécification de service et de protocole ainsi que le problème de dérivation, dans la section 3 nous présentons notre stratégie de synthèse. La section 4 donne des éléments de preuve de correction de notre algorithme de synthèse ainsi qu'une étude de sa complexité. Nous terminons par une conclusion et des perspectives.

2.2 Modèle de spécification des services et des protocoles

Nous avons choisi un modèle de réseaux de Petri interprétés pour la description des services et des entités de protocoles synthétisées. Notre choix est justifié principalement par :

(a) L'adaptation du modèle aux techniques de conception par raffinements [SUZ 83],

(b) l'existence de techniques d'analyse pour ce modèle,

(c) l'existence d'interfaces graphiques conviviales pour la description et l'analyse,

(d) la possibilité d'étendre le modèle par des contraintes quantitatives, telles que le temps,

(e) et le fait que ce modèle soit assez proche du langage de spécification Estelle normalisé par l'ISO [ISO 9074], ce qui faciliterait la translation à des spécifications Estelle. Une démarche inverse a été proposée dans [ZOU 91] pour la traduction de spécifications Estelle en réseaux de Petri, l'objectif étant l'analyse et la vérification de spécifications Estelle.

2.2.1 Définitions préliminaires

Rappelons deux définitions essentielles relatives aux réseaux de Petri [MUR 89].

Définition 1 (Réseau de Petri) *Un réseau de Petri est un graphe biparti orienté et valué ayant un état initial appelé marquage initial. Il est défini par un 5-uplet :*

$$R = \langle P, T, A, W, M_0 \rangle$$

- *P est un ensemble fini de places.*
- *T est un ensemble fini de transitions, $P \cap T = \phi$ et $P \cup T \neq \phi$.*
- *A est un ensemble d'arcs, $A \subseteq (P \times T) \cup (T \times P)$.*
- *W est une fonction de valuation ; $W : A \longrightarrow \{1, 2, ...\}$.*
- *M_0 est le marquage initial ; $M_0 : P \longrightarrow \{0, 1, 2, ...\}$.*

Si une place est marquée avec un nombre entier positif n alors on dit que cette place contient n jetons. ■

Pour un sommet $u \in P \cup T$, l'ensemble des successeurs est noté $u\bullet$, l'ensemble des prédécesseurs est noté $\bullet u$. On note aussi $u \bullet \bullet$, l'ensemble des successeurs des éléments de $u\bullet$, et u^+ l'ensemble de tous les sommets appartenant à la fermeture transitive de la relation successeur à partir de u (u non inclus). Une transition t est dite sensibilisée si et seulement si toute place $p \in \bullet t$ est marquée par au moins $W(p,t)$ jetons. Le tir de cette transition consiste à retrancher $W(p,t)$ jetons de toute place $p \in \bullet t$ et ajouter $W(t,p')$ jetons à toute place $p' \in t\bullet$.

Définition 2 (Réseau vivant et borné) *Un réseau de Petri R est dit vivant si et seulement si quel que soit le marquage M accessible à partir de M_0, et pour toute transition t, il existe une séquence de franchissements à partir de M qui conduit à un marquage permettant le franchissement de t. Un réseau est dit k-borné ssi le marquage de toute place p ne dépasse pas une constante donnée $k > 0$. Un réseau 1-borné est dit sauf.* ■

La propriété "vivant" d'un réseau de Petri assure l'absence d'interblocage et de transitions inaccessibles, et la propriété "bornée" garantit le non débordement des places.

2.2.2 Spécification de service

Les définitions suivantes vont nous permettre d'introduire un modèle de réseaux de Petri interprétés que nous appelons *modèle IPN* (Interpreted Petri Nets).

De manière informelle, une spécification de service consiste en deux parties : une partie contrôle et une partie opérative. La partie contrôle définit un réseau de Petri dont le rôle est de contrôler les séquences d'exécution des primitives de service définies dans la partie opérative. Ces primitives représentent les interactions potentielles entre les utilisateurs du service et le fournisseur du service, et ce à travers une interface définie par un ensemble de points d'accès au service (notées SAPs). Pour établir le lien entre la partie contrôle et

la partie opérative, chaque transition du réseau de Petri est étiquetée par une primitive de service.

Définition 3 (Spécification de service) *Une spécification de service $SPEC_s$ est définie par un triplet :*

$$SPEC_s = \langle PN, ENV, \mathcal{X} \rangle$$

où :

- *PN est un réseau de Petri représentant la partie contrôle ; $PN = \langle P, T, A, W, M_0 \rangle$*
- *$ENV = \langle D, SAP, PS \rangle$ désigne un environnement représentant la partie opérative tel que : $D = V \cup R$, désigne l'ensemble des données de la spécification, constitué d'un ensemble de variables V et d'un ensemble de ressources R ; on note E l'ensemble des états de D, SAP désigne l'ensemble des points d'accès au service, et PS dénote l'ensemble des primitives de service de $SPEC_s$.*
- *$\mathcal{X}$ est une fonction d'étiquetage servant à établir le lien entre la partie contrôle et la partie opérative ; $\mathcal{X} : T \longrightarrow PS$. Chaque transition est étiquetée par une et une seule primitive de service.* ■

Définition 4 (Primitive de service) *Une primitive de service ps est définie par un n-uplet où $1 \leq n \leq 5$ (les crochets délimitent les champs optionnels) :*

$$ps = \langle sap[, in][, cond][, act][, out] \rangle$$

où :

- *ps.sap représente le point d'accès de la primitive ps ($ps.sap \in SAP$).*
- *ps.in désigne un événement externe en entrée à travers ps.sap (on note "$\downarrow$ evt").*
- *ps.cond est une expression conditionnelle sur D représentant la condition d'exécution de la primitive ps ; $ps.cond : E \longrightarrow \{VRAI, FAUX\}$.*
- *ps.act est une action sur D ; $ps.act : E \longrightarrow E$.*
- *ps.out désigne un événement externe en sortie à travers ps.sap (on note "$\uparrow$ evt").*

■

Une spécification de service (Figure 2.1) est vue de l'extérieur comme une boîte noire accessible à travers l'ensemble de ses points d'accès SAP. Les événements d'entrée/sortie associés à une primitive de service servent à modéliser les interactions entre la spécification de service et son environnement externe (les utilisateurs du service).
Si la clause conditionnelle n'est pas définie dans une primitive de service alors sa valeur par défaut est la constante "VRAI".

Définition 5 (Événement) *Un événement externe en entrée e représente une réception de données de l'extérieur, on note $\downarrow e(x_1, x_2, ...)$, où $(x_1, x_2, ...)$ sont des variables de réception. Un événement externe en sortie s représente une émission de données vers l'extérieur, on note $\uparrow s(E_1, E_2, ...)$, où $(E_1, E_2, ...)$ sont des termes sur les données de l'environnement D.* ■

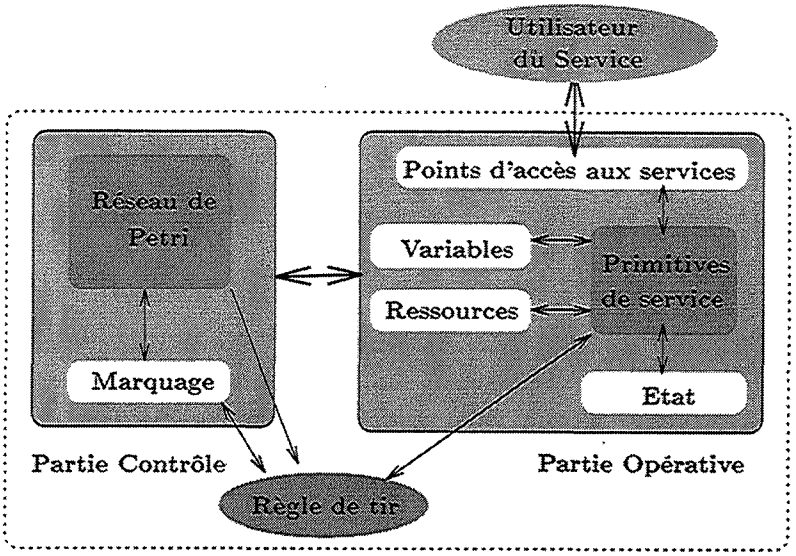

FIG. 2.1 – *Modèle de spécification de service*

Nous distinguons dans la partie opérative de $SPEC_s$ deux sortes de données globales: les variables et les ressources. Une variable est une donnée temporaire utilisée dans une primitive de service comme paramètre d'entrée ou de sortie, elle permet ainsi de modéliser le transfert de données entre des primitives liées à des points d'accès différents. En effet, si une primitive de service $ps_1 = \langle sap_1, \downarrow x \rangle$ est suivie d'une primitive de service $ps_2 = \langle sap_2, \uparrow x \rangle$ alors après exécution de ps_1, la valeur de la variable x doit être transférée de sap_1 à sap_2 afin de permettre l'exécution de ps_2. La sémantique d'une variable est donc similaire à celle donnée par BOCHMANN et GOTZHEIN [GOT 90a]. Toutefois, dans notre modèle, une variable peut être assignée plusieurs fois, ce qui n'est pas le cas dans la méthode de BOCHMANN et GOTZHEIN. La seule restriction imposée sur une variable est qu'elle ne peut pas faire l'objet d'accès concurrents.
Une ressource est une donnée globale persistante et pré-allouée à un ou plusieurs sites de l'architecture distribuée cible. Elle permet ainsi de définir certains éléments disponibles dans un système distribué, tels que: une base de donnée, un objet partagé, un serveur etc. Contrairement à une variable, une ressource peut faire l'objet d'accès concurrents.
Une primitive de service peut faire référence à toute ressource du système quelle que soit sa localisation, et à toute variable en respectant la restriction citée ci-dessus.

Définition 6 (Ressource) *Une ressource R_k est définie par un état interne et un ensemble d'opérations élémentaires sur cet état:*

$$R_k = \langle S, OP \cup \{Id(), St(x)\} \rangle$$

Soit E_{R_k} l'ensemble de tous les états de R_k, $R_k.S \in E_{R_k} \cup \{\perp\}$ désigne l'état interne de la ressource, et $R_k.OP$ représente un ensemble d'opérations. Le symbole $\perp$ représente un état interne indéfini utilisé pour désigner les ressources sans état.

Une opération de $R_k.OP$ peut être de deux types:

- *Une opération de type* **procédure**, $R_k.p(E_1, ..., E_n \mid x_1, ..., x_m)$, *où* $E_1, ..., E_n$ *sont des termes sur* D *définissant les arguments d'entrée et* $x_1, ..., x_m$ *sont des variables représentant les arguments de sortie. Cette opération peut modifier l'état interne de la ressource* R_k, *elle est donc caractérisée par sa nature: lecture ou écriture* ($R_k.p.nat \in \{Lecture, Ecriture\}$).

- *Une opération de type* **fonction**, $R_k.f(E_1, ..., E_n)$, *où* $E_1, ..., E_n$ *sont des termes sur* D *représentant les arguments d'entrée de* f. *Une fonction fournit une valeur et ne modifie pas l'état interne de la ressource.*

$R_k.Id()$ *désigne* **la fonction identité** *qui fournit l'état interne de la ressource, et* $R_k.St(x)$ *est* **la procédure d'initialisation** *qui permet d'initialiser l'état interne de* R_k *avec la valeur de la variable* x. ■

Exemple: Soit une base de données relationnelle:

$$B = \langle S, \{size(), append(x), Id(), St(y)\}\rangle$$

La base de données B définit une ressource disponible dans le système. La fonction $B.size()$ fournit la taille de la base de données. La procédure $B.append(x)$ consiste à ajouter dans B l'enregistrement défini par la variable x. Donc, $B.append.nat = Ecriture$. L'ensemble $B.S$ définit les tuples de la base de données. La fonction identité $B.Id()$ renvoie l'état interne de la base de données (i.e., $B.S$). La procédure $B.St(y)$ permet d'initialiser l'état interne de la base de données par la valeur de la variable y.

Définition 7 (Terme) *Un terme est défini comme suit:*

1. *une variable est un terme,*
2. *une constante est un terme,*
3. *si* $E_1, ..., E_n$ *sont des termes et* f *est une fonction d'arité* n *liée à une ressource* R_k, *alors* $R_k.f(E_1, ..., E_n)$ *est un terme.* ■

Définition 8 (Expression Conditionnelle) *Une expression conditionnelle est définie comme suit:*

1. *les constantes* $VRAI$ *et* $FAUX$ *sont des expressions conditionnelles*
2. *un terme de type booléen est une expression conditionnelle*
3. *si* E_1 *et* E_2 *sont deux termes de type arithmétique (i.e., entier ou réel) alors* $E_1 \alpha E_2$ *est une expression conditionnelle, où* $\alpha \in \{=, \leq, \geq, <, >\}$.
4. *si* C_1 *et* C_2 *sont deux expressions conditionnelles alors* $C_1 \vee C_2$, $C_1 \wedge C_2$, $\neg C_1$ *et* (C_1) *sont des expressions conditionnelles. Ces expressions définissent respectivement une conjonction, une disjonction, une négation et une expression parenthésée.* ■

Une action associée à une primitive de service permet de modifier l'état des données de la spécification de service, elle est définie comme suit :

Définition 9 (Action) *Une action associée à une primitive de service ps représente soit une procédure liée à une ressource, soit une opération quelconque modifiant l'état des variables de $SPEC_s$.*

$$ps.act = \begin{cases} R_k.p(E_1, ..., E_n \mid x_1, ..., x_m) \\ ou \\ p'(E_1, ..., E_n \mid x_1, ..., x_m) \end{cases}$$

Où les E_i (i allant de 1 à n) sont des termes sur D en entrée des opérations $R_k.p$ et p', et les x_i (i allant de 1 à m) sont des variables en sortie. ■

Exemple : Soient 3 bases de données relationnelles : $B = \langle S, \{size(), append(x), Id(), St(y)\}\rangle$, $B_1 = \langle S_1, \{Id(), St(y)\}\rangle$ et $B_2 = \langle S_2, \{Id(), St(y)\}\rangle$.

1. $B.append(x)$ est une action liée à la base de données B.
2. $Union(B_1.Id(), B_2.Id() \mid y)$ est une action qui consiste à faire l'union des deux bases de données B_1 et B_2 et à conserver le résultat dans la variable y.
3. $B.St(y)$ est l'action qui initialise la base de données B avec la valeur de y.

Définition 10 (Règle de franchissement) *Une transition t de $SPEC_s$ est dite franchissable à un état donné de la spécification si et seulement si :*

(a) elle est sensibilisée dans le réseau de Petri sous-jacent,

(b) la condition $\mathcal{X}(t).cond$ est satisfaite,

(c) les données attendues en entrée $\mathcal{X}(t).in$ sont disponibles (i.e., l'événement attendu entrée s'est produit).

Le franchissement de t consiste alors à :

(1) modifier le marquage du réseau de Petri sous-jacent,

(2) consommer les données disponibles en entrée (i.e., consommer l'événement en entrée $\mathcal{X}(t).in$),

(3) effectuer l'action définie par $\mathcal{X}(t).act$,

(4) sortir, à travers le point d'accès $\mathcal{X}(t).sap$, les données définies par les expressions de $\mathcal{X}(t).out$ (i.e., déclencher l'événement en sortie $\mathcal{X}(t).out$). ■

Quand un événement en entrée arrive alors que l'une des conditions de franchissement (a) et (b) n'est pas satisfaite, il est simplement ignoré.
Les champs d'une primitive de service (excepté "sap") étant optionnels, plusieurs types de primitives peuvent être définis :

a) *Une primitive avec événement externe en entrée* : $\langle sap, in[, cond][, act]\rangle$. A titre d'exemple,

$\langle TSAP_1, \downarrow TConRequest(cg, cd, qos, d)\rangle$ est une primitive du service de transport ISO qui définit une demande de connexion à travers le point d'accès $TSAP_1$. Les variables attendues en entrée désignent respectivement : l'appelant, l'appelé, la qualité de service désirée et une donnée de l'utilisateur.

b) *Une primitive avec événement externe en sortie* : $\langle sap[, cond][, act], out\rangle$. A titre d'exemple, $\langle TSAP_2, qos > val, \uparrow TConIndication(cg, cd, qos, d)\rangle$ est une primitive du service de transport définissant une indication d'une connexion de transport au $TSAP_2$. Cette indication n'est déclenchée que si la qualité de service *qos* est supérieure à la valeur *val*.

c) *Une primitive avec événement externe atomique en entrée/sortie* : $\langle sap, in[, cond][, act], out\rangle$. A titre d'exemple, la primitive $\langle TSAP_1, \downarrow TConRequest(cg, cd, qos, d), NbCon > MaxCon, \uparrow TDiscIndication(r, d)\rangle$ spécifie que si, lors de la réception d'une demande de connexion à travers $TSAP_1$, le nombre de connexions dépasse une limite définie par $MaxCon$, une indication de déconnexion est fournie à travers $TSAP_1$.

d) *Une primitive sans événement externe* : $\langle sap[, cond][, act]\rangle$. A titre d'exemple, on considère une base de données $B = \langle S, \{size(), append(x), Id(), St(y)\}\rangle$. La primitive $\langle SAP_1, B.size() < Tmax, B.append(x)\rangle$ spécifie que si la taille de la base de données B est inférieure à une limite $Tmax$, l'enregistrement défini par x est ajouté dans la base.

2.2.3 Spécification de protocole et problème de synthèse

2.2.3.1 Spécification de protocole

Définition 11 (Spécification de protocole) *La spécification d'un protocole de communication (on note $SPEC_p$) est définie par les spécifications des entités de protocoles (on note EP^i) qui le composent, $SPEC_p = \langle EP^1, ..., EP^n\rangle$.* ■

La spécification d'une entité de protocole est définie dans le même modèle que celui d'une spécification de service à savoir le modèle IPN. Cependant, une entité de protocole peut utiliser, en plus des primitives de service définies dans $SPEC_s$ (appelées *primitives de service externes*), des *primitives de service internes*. Une primitive de service interne est caractérisée par un événement interne qui représente en général une réception de message d'une entité EP^k (on note $r_k(msg)$) ou une émission de message vers une entité EP^k (on note $s_k(msg)$). On suppose qu'au sein de toute paire d'entités de protocole (EP^i, EP^j), il existe un lien de communication FIFO (First In First Out) sans perte de messages[1].

Avant de définir la spécification d'une entité de protocole, nous introduisons un concept de *sous-primitive de service* qui nous permet de distinguer entre les comportements observables et les comportements non observables.

Définition 12 (Sous-primitive de Service) *Soit une primitive de service ps définie par un n-uplet telle que $1 \leq n \leq 5$. Une sous-primitive de service (notée ps_i) extraite de ps est définie par un k-uplet telle que :*

(a) $k \leq n$,

1. Cette hypothèse est discutée à la fin du chapitre 4.

(b) et toute clause de ps_i est une clause de ps.

On note $\mathcal{P}$ l'ensemble de toutes les sous-primitives que l'on peut extraire des primitives de service de PS. ■

Définition 13 (Spécification d'une entité de protocole) *Une entité de protocole dérivée à partir d'une spécification de service $SPEC_s$ est définie par un triplet $\langle PN^i, ENV^i, \mathcal{X}^i \rangle$ où :*

- *PN^i est un réseau de Petri construit à partir de PN,*
- *ENV^i est un environnement représentant la partie opérative $ENV^i = \langle D^i, SAP^i, PS^i \cup IPS \cup \{\epsilon\} \rangle$ tel que :*
  - *$D^i = V^i \cup R^i$, où $V^i \subseteq V$ est l'ensemble des variables locales à EP^i et $R^i \subseteq R$ est l'ensemble des ressources utilisées par EP^i.*
  - *$SAP^i \subseteq SAP$ est l'ensemble des points d'accès servis par EP^i.*
  - *$PS^i \subset \mathcal{P}$ est l'ensemble des primitives de service observables dans EP^i.*
  - *IPS est un ensemble de primitives de service internes (i.e., non observables). Une primitive de service interne est définie par le n-uplet suivant :*

$$\langle [r_j(m),][cond,][act,][s_k(m')] \rangle$$

où $r_j(m)$ est une réception du message m de l'entité EP^j, "cond" est une expression conditionnelle sur D^i, "act" est une action sur D^i et $s_k(m')$ est une émission du message m' vers l'entité EP^k. Cette primitive spécifie que lors de la réception du message m de l'entité EP^j, si "cond" est vraie, l'action "act" est exécutée et le message m' est envoyé à l'entité EP^k. Les clauses "act" et "cond" sont définies de la même manière que dans une primitive de service externe.

- *$\mathcal{X}^i$ est une fonction d'étiquetage servant à établir le lien entre la partie contrôle et la partie opérative. $\mathcal{X}^i : T^i \longrightarrow PS^i \cup IPS \cup \{\epsilon\}$, où T^i est l'ensemble des transitions de PN^i. Chaque transition est étiquetée par une et une seule primitive de service interne ou externe.* ■

Une transition t d'une entité de protocole EP^i telle que toutes les clauses de $\mathcal{X}^i(t)$ sont vides est étiquetée par ϵ. Elle est dite une ϵ-transition.

2.2.3.2 Problème de dérivation

On considère une architecture distribuée définie par un ensemble de sites, chaque point d'accès au service étant alloué à un site unique, et chaque ressource étant allouée à un ou plusieurs sites. Le problème consiste à implanter une spécification de service sur cette architecture en construisant n entités de protocoles communicantes, chacune associée à un site particulier.

Dans une spécification de service il peut y avoir des accès concurrents à des ressources éventuellement dupliquées. Un conflit entre transitions est défini comme suit :

Définition 14 (Conflit) *On dit que deux transitions t_1 et t_2 de $SPEC_s$ sont en* **conflit** *si et seulement si elles sont concurrentes (i.e., elle peuvent s'exécuter en parallèle) et il existe au moins une ressource R_k telle que l'une au moins des assertions suivantes soit vraie :*

- *t_1 et t_2 accèdent à R_k en écriture,*
- *t_1 accède à R_k en lecture et t_2 accède à R_k en écriture, ou inversement.*

Une telle transition est appelée **transition conflictuelle.** ■

Deux contraintes de cohérence doivent, par conséquent, être respectées par les spécifications des entités de protocoles dérivées :

- **Contrainte C1 :** Les modifications successives effectuées sur des ressources dupliquées doivent être observées dans le même ordre par toutes les entités de protocole. L'ordre dans lequel s'effectuent ces modifications n'est pas déterminé a priori.
- **Contrainte C2 :** Si une transition t_1 effectue une modification sur une ressource R_k et une autre transition t_2 concurrente effectue plusieurs lectures sur la ressource R_k alors t_2 doit percevoir, pour toutes ses lectures, soit l'état de R_k avant l'exécution de t_1 soit celui d'après mais pas un mélange des deux états. En d'autres termes, une transition doit être atomique.

Par ailleurs, les entités de protocoles dérivées doivent être correctes *logiquement* et *sémantiquement*. La correction logique signifie : la vivacité et le caractère borné du réseau de Petri sous-jacent à la spécification. La correction sémantique signifie que la spécification du protocole est conforme à la spécification du service. Nous définissons cette conformité par une *relation d'équivalence* entre service et protocole. On considère que les primitives de service observables sont celles définies dans l'ensemble $\mathcal{P}$ et que toutes les autres primitives sont non observables.

Définition 15 (Équivalence) *On dit que $SPEC_s$ est équivalente à $SPEC_p$ si et seulement si, toute séquence d'exécution de primitives de service observable dans $SPEC_p$ est une séquence observable dans $SPEC_s$ et vice versa.* ■

L'équivalence entre service et protocole est étudiée en détail dans la section 2.4.

Définition 16 (Problème de synthèse) *Le problème de synthèse considéré est défini comme suit :*

- **Entrées** *:*
  1. *Une spécification de service $SPEC_s$*
  2. *Une architecture distribuée cible définie par un ensemble S de n sites et une fonction d'allocation, $Alloc : SAP \cup R \rightarrow S$, définie comme suit :*
     (a) $\forall s \in SAP$, il existe un unique $i \in S$ tel que $Alloc(s) = i$,
     (b) $\forall R_k \in R, Alloc(R_k) \subseteq S$.

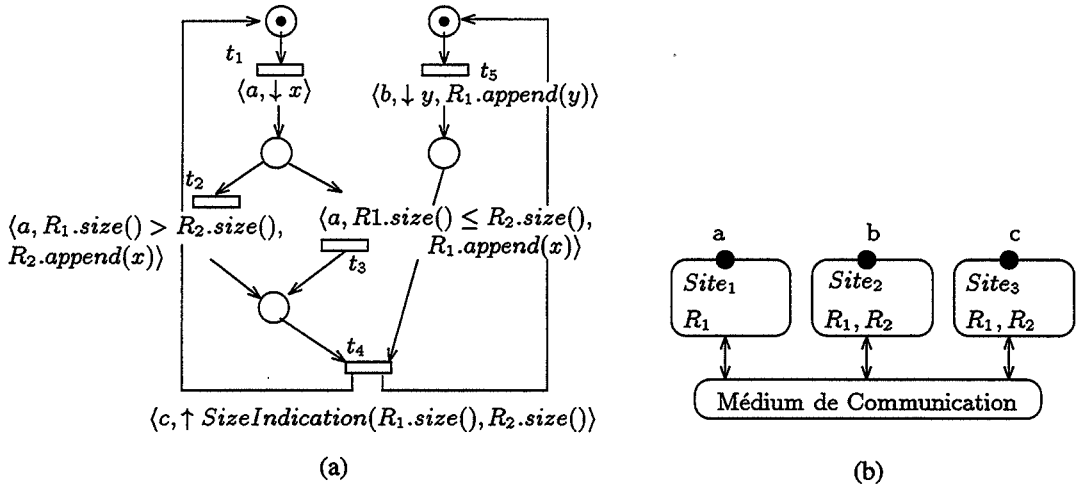

FIG. 2.2 – *Exemple de spécification de service*

La fonction Alloc est étendue aussi aux transitions de $SPEC_s$ *;* $\forall t \in T, Alloc(t) = Alloc(\mathcal{X}(t).sap)$.

- **Sorties** : *Une spécification de protocole* $SPEC_p = (EP^1, ..., EP^n)$, *telle que chaque* EP^i *soit associée à un seul site* i. *La spécification* $SPEC_p$ *doit être correcte logiquement et sémantiquement et doit satisfaire les contraintes* $C1$ *et* $C2$. ■

2.2.4 Exemple de spécification de service

Considérons un service de saisie sur deux bases de données dupliquées R_1 et R_2. Le service est constitué de trois points d'accès a, b et c. Les deux premiers points servent à la saisie et le troisième point est utilisé pour contrôler la taille des bases de données.
La Figure 2.2a représente la spécification du service dans le modèle IPN. Deux données peuvent être simultanément saisies à travers les points d'accès a et b (transitions t_1 et t_5). La valeur entrée par b est systématiquement ajoutée à la base de donnée R_1, alors que celle saisie par le point a est ajoutée à R_1 si la condition $R_1.size() \leq R_2.size()$ est vraie (transition t_3), sinon elle est ajoutée à la base de données R_2 (transition t_2). Ce service de saisie doit maintenir la taille de R_1 toujours plus grande que celle de R_2. Les tailles des bases de données R_1 et R_2 sont ensuite fournies à travers le point d'accès c (transition t_4). La fonction $size()$ et l'opération $append()$ sont des opérateurs internes aux deux ressources R_1 et R_2. L'architecture distribuée cible considérée est définie par trois sites S_1, S_2 et S_3 auxquels sont respectivement alloués les points d'accès a, b et c. Par ailleurs, la ressource R_1 est allouée aux trois sites, et la ressource R_2 est allouée à S_2 et S_3 (cf. Figure 2.2b). Ceci représente un exemple d'entrée pour le problème de dérivation de protocole défini précédemment.

2.3 Stratégie de Synthèse

L'idée de base de notre stratégie de synthèse est de raffiner une place ou une transition u de la spécification de service en utilisant un ensemble de règles afin de construire n sous-réseaux de Petri coopérants $(SP^1(u), ..., SP^n(u))$. Chaque sous-réseau $SP^k(u)$ définit l'implantation de u au sein de l'entité de protocole EP^k. Une entité de protocole EP^k est ensuite construite en remplaçant chaque place ou transition u de $SPEC_s$ par $SP^k(u)$. Les problèmes que nous nous proposons de résoudre dans ce chapitre se résument comme suit :

1. *Synthèse des traitements répartis engendrés par une transition :* La partie action d'une transition peut exiger une coopération entre plusieurs entités de protocole. Cette coopération doit être construite à l'aide d'un ensemble de sous-réseaux de Petri, chacun étant associé à une entité de protocole.

2. *Synthèse du flot de contrôle :* un sous-réseau de Petri $SP^k(t)$ dérivé à partir d'une transition t de $SPEC_s$ ne peut être mis dans son état initial que lorsque tous les sous-réseaux correspondants aux transitions prédécesseurs de t dans $SPEC_s$ ont atteint leurs états finaux.

3. *Synthèse du flot de données :* Quand un ensemble de sous-réseaux de Petri implémentant une transition de $SPEC_s$ atteint son état final, certaines données doivent être transférées afin de permettre l'exécution des sous-réseaux implémentant les transitions successeurs.

4. *Gestion des conflits :* les sous-réseaux de Petri dérivés à partir de transitions conflictuelles doivent respecter les contraintes de cohérence $C1$ et $C2$.

5. *Choix distribué :* il est possible de décrire dans $SPEC_s$ un choix (déterministe ou non) entre plusieurs transitions allouées à des sites différents. Le problème est de trouver une implantation distribuée de ce type de structure sélective.

6. *Réduction des entités de protocole :* si une entité de protocole EP^k ne participe pas à l'implantation d'une transition t de $SPEC_s$, $SP^k(t)$ est définie par une transition vide ; le problème se ramène à trouver des règles de suppression des transitions vides dans un réseau de Petri.

7. *Preuve de correction :* il faut prouver que la méthode de synthèse génère des protocoles corrects logiquement et sémantiquement.

La section 2.3.4 présente un ensemble de règles de synthèse traitant simultanément les problèmes (1), (2), (3) et (4). Les problèmes (1) et (2) sont directement résolus par les règles de synthèse alors que les problèmes (3) et (4) sont d'abord traités dans les sections 2.3.3 et 2.3.2. Dans la section 2.3.5, l'ensemble des règles est étendu afin de supporter la synthèse des structures sélectives distribuées (i.e., problème (5)). Les problèmes (6) et (7) sont traités, respectivement, dans les sections 2.3.6 et 2.4.
Pour des raisons de simplicité nous posons, les restrictions suivantes :

- R1 : On suppose que le service $SPEC_s$ est correct, c'est à dire que le réseau de Petri sous-jacent est vivant et sauf.

- R2 : Une transition de $SPEC_s$ qui est franchissable à l'état initial doit être non conflictuelle et doit disposer de toutes les ressources nécessaires à l'évaluation de sa partie condition.
- R3 : Une structure sélective de $SPEC_s$ doit être à choix libre ; c'est-à-dire que pour toute place p de $SPEC_s$, on a : $\bullet(p\bullet) = \{p\}$.
- R4 : Une variable de $SPEC_s$ ne doit pas faire l'objet d'accès concurrents.

La restriction R1 est intrinsèque aux approches de synthèse. En effet, pour pouvoir dériver des spécifications correctes, il est nécessaire que les spécifications de départ soient correctes. Par ailleurs, nous imposons la contrainte "sauf" pour simplifier les règles de synthèse. La restriction R2 peut être levée d'une part en introduisant une phase initiale dans le protocole qui consiste à rapatrier les valeurs de ressources nécessaires à l'exécution des transitions initiales, et d'autre part en ajoutant des transitions initiales vides afin de rendre non initiales les transitions conflictuelles qui le sont. La restriction R3 est faite pour simplifier la synthèse du flot de données et des structures sélectives distribuées. La restriction R4 est liée à la sémantique que nous avons attribuée au rôle d'une variable, à savoir le transfert de données entre transitions (ou primitives de service). Ces restrictions sont discutées encore en conclusion de ce chapitre.

2.3.1 Le concept de règle de synthèse

Afin d'offrir à notre approche un contexte de synthèse[2] extensible, nous distinguons deux catégories de règles de synthèse : *les règles contextuelles* et *les règles de base*. Les règles contextuelles dépendent du contexte de la synthèse, c'est à dire, des aspects que l'on souhaite synthétiser (flot de données, flot de contrôle, etc.), alors que les règles de base dépendent du modèle de spécification et servent de constructeurs pour les règles contextuelles.
Soit u une place ou une transition de $SPEC_s$, la forme générale d'une règle contextuelle se représente ainsi :

$$\{SP_e^1(u), ..., SP_e^k(u)\} \xrightarrow{(P,\tau)} \{SP_{e+1}^1(u), ..., SP_{e+1}^k(u)\}$$

Si le prédicat P est vrai, la règle transforme le membre gauche, en utilisant une séquence de règles de base τ, afin de construire le membre droit. Le membre gauche désigne un ou plusieurs sous-réseaux de Petri associé(s) à un objet (place ou transition) u de $SPEC_s$ à une étape e de la synthèse, et le membre droit désigne le ou les sous-réseau(x) obtenu(s) après application de la règle (étape $e+1$).

Pour un sous-réseau donné à une étape e de la dérivation (SP_e^i), on distingue trois transformations de base (dites règles de base) :

- **la T-substitution**, consiste à remplacer une transition de SP_e^i par un bloc de réseaux de Petri BP, on note : $SP_{e+1}^i = [tr/BP]SP_e^i$. La transition tr est une transition particulière qui définit en fait un point de raffinement. Nous appelons ce type de transition : *méta-transition*. Une méta-transition possède un ensemble d'attributs, tels

2. Le contexte de synthèse est un critère d'évaluation des méthodes de synthèse de protocoles, il est introduit au chapitre 1.

que la transition de $SPEC_s$ à synthétiser, une place particulière de $SPEC_s$, etc. Ces attributs sont utilisés dans la construction des blocs de réseaux de Petri à substituer. Un bloc peut contenir à son tour des méta-transitions, ceux-ci vont alors hériter de tous les attributs de la méta-transition qui a été substituée.

- **la P-substitution**, consiste à remplacer une place pr de SP_e^i par un bloc de réseaux de Petri BP, on note: $SP_{e+1}^i = [pr/BP]SP_e^i$. La place pr définit également un point de raffinement attribué que nous appelons *méta-place.*

- et **l'Addition** consiste à injecter dans SP_e^i un bloc de réseaux de Petri donné BP, on note: $SP_{e+1}^i = SP_e^i + BP$.

La T-substitution et l'Addition sont utilisées pour la synthèse de transitions. Les blocs qui y sont utilisés sont constitués par un ensemble de transitions de début (i.e., sans places d'entrée) et un ensemble de transitions finales (i.e., sans places de sortie). La P-substitution est utilisée pour la synthèse de places dites à choix distribué; c'est-à-dire, des places ayant plusieurs transitions en sortie parmi lesquelles au moins deux font référence à deux points d'accès distants. Les blocs utilisés par une P-substitution sont constitués par une place initiale (i.e., place marquée à l'état initial du bloc) et une ou plusieurs places finales (i.e., places potentiellement marquées à l'état final du bloc). Parmi les places finales, une et seulement une doit être marquée à l'état final du bloc. Des détails sur les règles de base ainsi que leur correction sont donnés dans la section 2.4.1

Une règle contextuelle, définie par un couple (P, τ), est applicable pour un objet u de $SPEC_s$ à une étape e de la synthèse si et seulement si:

1. pour chaque T-substitution de τ de la forme: $SP_{e+1}^i(u) = [tr/BP]SP_e^i(u)$, la méta-transition tr apparaît dans $SP_e^i(u)$,

2. pour chaque P-substitution de τ de la forme: $SP_{e+1}^i(u) = [pr/BP]SP_e^i(u)$, la méta-place pr apparaît dans $SP_e^i(u)$, et

3. le prédicat P est vrai.

L'application de la règle contextuelle consiste alors à effectuer les transformations définies par τ.

2.3.2 Gestion des conflits entre transitions

Dans une spécification de service $SPEC_s$, il peut y avoir des transitions conflictuelles. Nous associons, de ce fait, à toute spécification de service un graphe modélisant tous les conflits de cette spécification.

Définition 17 (Graphe des conflits) *On appelle graphe des conflits associé à une spécification de service $SPEC_s$, le graphe non orienté $GC = \langle T, U \rangle$ tel que:*

- *T est l'ensemble des transitions de $SPEC_s$ désignant les sommets du graphe,*

- *$U = \{(t_1, t_2) \in T \times T \mid t_1$ et t_2 sont en conflit $\}$.* ■

Nous présentons dans ce qui suit deux approches différentes pour la gestion des conflits. Les deux approches sont comparées dans la section 2.3.2.3. Le problème de construction du graphe des conflits est abordé en section 2.3.2.4.

2.3.2.1 Première approche

Nous supposons dans cette approche que les spécifications de service sont indépendantes l'une de l'autre. En d'autres termes, il n'existe pas de ressources communes entre plusieurs spécifications de service différentes. Dans ce cas, le graphe des conflits associé à une spécification de service (graphe GC) donne une description complète des conflits qui existent dans cette spécification. Nous utilisons alors ce graphe pour dériver automatiquement un protocole de contrôle de cohérence entre les transitions conflictuelles.
Dans le graphe GC chaque transition est en conflit avec toutes les transitions adjacentes. Les transitions non adjacentes ne sont pas en conflit et peuvent donc s'exécuter en parallèle indépendamment l'une de l'autre.

La solution que nous adoptons pour respecter les contraintes $C1$ et $C2$, consiste à assurer une exclusion mutuelle entre toute transition t et ses transitions adjacentes dans le graphe GC. Ceci constitue une condition suffisante pour les contraintes $C1$ et $C2$. En effet, la contrainte $C2$ est évidemment respectée par définition de l'exclusion mutuelle. Par ailleurs, supposons qu'il existe un ensemble de transitions qui modifient concurremment une ressource R_k dupliquée ; ces transitions sont deux à deux adjacentes dans GC (d'après la Définition 17) et doivent donc s'exécuter en exclusion mutuelle. Comme l'exclusion mutuelle assure un ordre total entre les transitions conflictuelles, toutes les entités de protocole détenant une copie de R_k observeront les modifications dans le même ordre.

Nous distinguons dans le graphe GC deux types de conflits : les *conflits globaux* et les *conflits locaux*. Un conflit entre deux transitions est dit global si les deux transitions appartiennent respectivement à deux sites différents, il est dit local dans le cas contraire. Le protocole d'exclusion mutuelle que nous voulons dériver doit assurer deux niveaux de contrôle de cohérence : le premier niveau concerne les transitions qui appartiennent à des sites différents et le deuxième niveau concerne les transitions d'un même site. Pour distinguer ces deux niveaux, nous subdivisons le graphe GC en composantes connexes C_i, telles que toutes les transitions (i.e., sommets) d'une même composante soient allouées à un même site ; nous appelons ces composantes des *composantes locales*. Les transitions qui appartiennent à des composantes différentes (i.e., niveau global) sont contrôlées par un algorithme d'exclusion mutuelle distribué, alors que celles qui sont dans une même composante (i.e, niveau local) sont contrôlées par un algorithme d'exclusion mutuelle centralisé. Nous présentons dans ce qui suit l'algorithme de contrôle qui est dérivé pour une transition t_n de GC. Cet algorithme est une adaptation de l'algorithme d'exclusion mutuelle distribuée de LAMPORT [LAM 78].

Algorithme de contrôle pour une transition t_n
debut
soit C_n la composante locale de t_n;
soient $t_0, ..., t_{n-1}$ les transitions adjacentes à t_n mais qui ne sont pas dans C_n;
type

horloge : 0..+∞;
type_msg : {req, ack, rel};
message : (msg : type_msg, h : horloge, t : 0..n);
privilege : {VRAI, FAUX};
var
h_n : horloge;
F : **tableau** [0..K] de message;
(** où K est le nombre total de transitions conflictuelles dans GC **)
initialisation ∀ j∈{0..K}, F[j]:=(rel,0,t_j);
lors de décision d'entrer en section critique **faire**
Diffuser(req, h_n, t_n) $\bigcup_{k=0}^{k=n-1}\{Alloc(t_k)\}$;
soit k tel que F[k].t=t_n;
F[k]:=(req, h_n, t_n);
h_n:=h_n+1;
Attendre : $\forall j \neq n$ Estamp(F[n])<Estamp(F[j]);
(* Où Estamp(m) donne l'estampille d'un message m *)
Demander SC locale;
(** Demander l'entrée en section critique localement à Alloc(t_n),
ceci concerne les transitions adjacentes à t_n et qui sont dans C_n **)
Attendre : Autorisation locale d'entrée en SC;
privilege:=VRAI;
fait;

lors de réception de (req,h_m,t_m) d'un site Alloc(t_m) **faire**
Traiter_requete(h_m,t_m) **debut**
MAJ(h_n, h_m);
soit k tel que F[k].t=t_m;
F[k]:=(req,h_m,t_m);
fin;
Envoyer(ack,h_n,t_n) à Alloc(t_m);
fait;

lors de réception de (ack,h_m,t_m) d'un site Alloc(t_m) **faire**
Traiter_ack(h_m,t_m) **debut**
MAJ(h_n, h_m);
soit k tel que F[k].t=t_m;
si F[k].msg≠req **alors** F[k]:=(ack,h_m,t_m) **fsi**;
fin
fait;

lors de sortie de la section critique **faire**
Traiter_sortie(h_n,t_n) **debut**
Libérer la section critique localement à Alloc(t_n);
(* concerne les transitions adjacentes à t_n et qui sont dans C_n *)
F[n]:=(rel, h_n, t_n);

```
            privilege:=FAUX;
        fin;
        Diffuser(rel, h_n, t_n) ⋃_{k=0}^{k=n-1} {Alloc(t_k)};
    fait;

    lors de réception de (rel,h_m,t_m) d'un site Alloc(t_m) faire
        Traiter_rel(h_m,t_m) debut
            MAJ(h_n, h_m);
            soit k tel que F[k].t=t_m;
            F[k]:=(rel,h_m,t_m);
        fin
    fait;

    procedure MAJ (h_n, h_m : horloge);
    debut
        si h_n < h_m alors h_n:=h_m fsi;
        h_n:=h_n+1;
    fin
fin.
```

Le raffinement d'une transition conflictuelle doit donc commencer par une demande d'entrée en section critique et se terminer par la libération de la section critique. En plus, un prédicat noté "Privilege" est utilisé pour savoir si l'autorisation d'entrée en section critique est accordée ou non. Les demandes d'entrée et de sortie sont à leur tour raffinées selon l'algorithme de contrôle adopté, à savoir dans notre cas, l'algorithme dérivé du graphe GC. Les raffinements nécessaires pour une demande d'entrée en section critique sont les suivants (cf. Figure 2.3a) :
$SP^i_{e+1}(t_n) = [tr_E(t_n)/BP1]SP^i_e(t_n)$ (où $i = Alloc(t_n)$);
$SP^j_{e+1}(t_n) = SP^j_e(t_n) + BP2$ (pour tout EP^j qui reçoit un message de type "ack");
$SP^k_{e+1}(t_n) = SP^j_e(t_n) + BP3$ (pour tout EP^k qui reçoit un message de type "req");
Par ailleurs, une sortie de la section critique requiert les transformations suivantes (cf. Figure 2.3b) :
$SP^i_{e+1}(t_n) = [tr_S(tn)/BP1]SP^i_e(t_n)$; (où $i = Alloc(t_n)$);
$SP^j_{e+1}(t_n) = SP^j_e(t_n) + BP2$ (pour tout EP^j qui reçoit un message de type "rel").

Les transitions $tr_E(t_n)$ et $tr_S(t_n)$ désignent des points de raffinement utilisés pour construire les blocs de réseaux de Petri correspondant respectivement à la demande d'entrée en section critique et la libération de la section critique. Dans la Figure 2.3, les primitives étiquetant les transitions correspondent directement aux traitement définis dans l'algorithme de contrôle d'une transition t_n.

2.3.2.2 Seconde approche

Dans cette approche, nous supposons qu'il peut y avoir des ressources communes entre des spécifications de service différentes. Dans ce cas, les informations données par un graphe des conflits deviennent incomplètes car elles ne tiennent pas compte des conflits engendrés

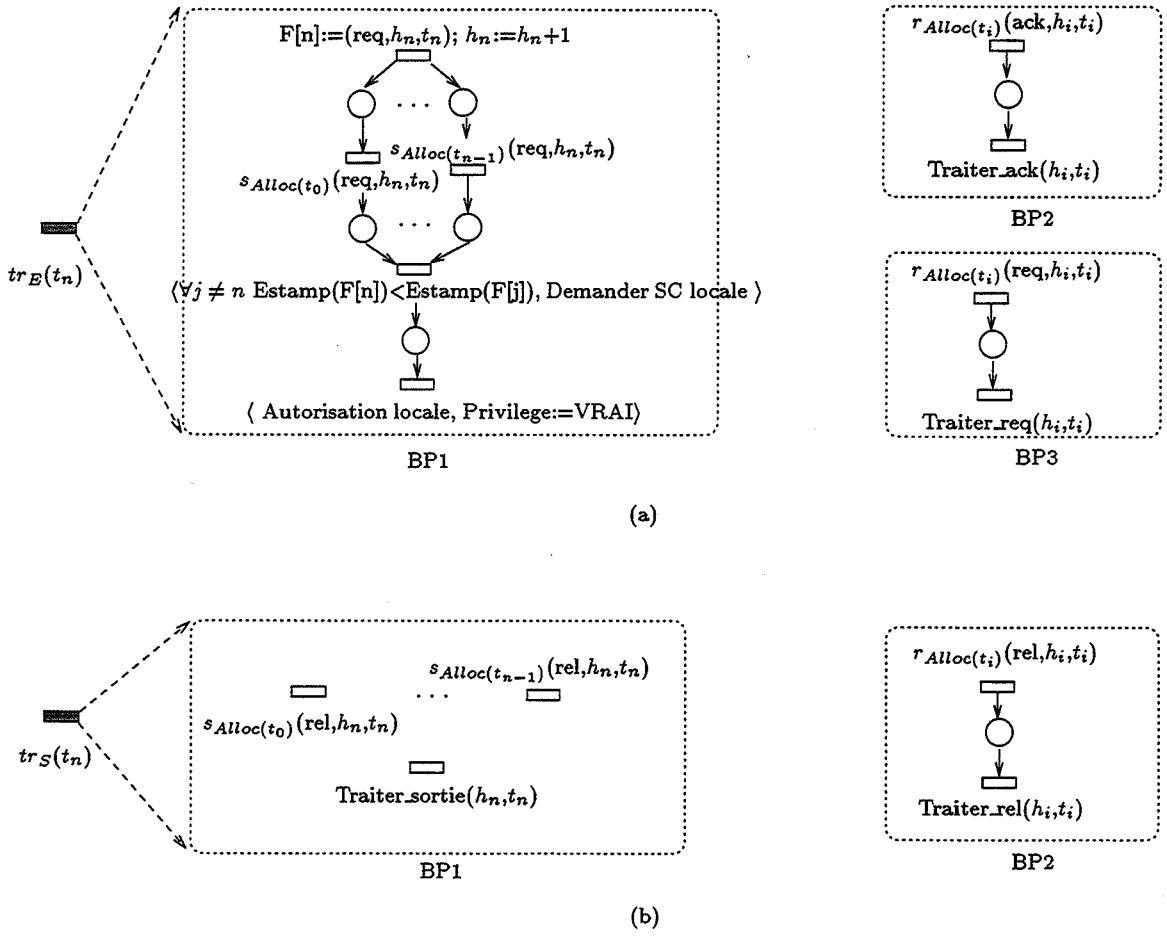

FIG. 2.3 – *Synthèse du contrôle de cohérence pour une transition conflictuelle* t_n

par d'autres spécifications de service. De ce fait, on considère chaque transition d'une spécification de service comme une transaction. L'exécution d'une transition de service doit respecter les contraintes C1 et C2, ce qui justifie l'analogie avec une transaction. Le respect des contraintes C1 et C2 est un problème qui a été abordé par plusieurs chercheurs, un tour d'horizon sur le contrôle de concurrence est présenté dans [BER 87]. Dans cette approche nous proposons l'utilisation du protocole de verrouillage à deux phases dont la correction par rapport aux contraintes C1 et C2 a déjà été prouvée. Chaque transaction doit d'abord verrouiller toutes les ressources qu'elle utilise avant d'entamer son exécution. Une fois son exécution terminée, toutes les ressources verrouillées sont libérées. Deux opérations sont alors définies :

- *lock(R_k, x)* : permet de verrouiller la ressource R_k en lecture (x=read) ou en écriture (x=write).

- *unlock(R_k)* : permet de déverrouiller la ressource R_k.

Le principe de verrouillage à deux phases se présente ainsi :

1. *Quand une transaction accède à une ressource R_k :*
   - *a) Si R_k n'est pas encore verrouillée alors elle est verrouillée et la transaction continue son exécution.*
   - *b) Si R_k a été déjà verrouillée par une autre transaction de façon conflictuelle (i.e., verrouillages write/write ou write/read ou read/write) alors la transaction doit attendre jusqu'à ce que la ressource soit déverrouillée.*
   - *c) Si la ressource a été déjà verrouillée par une autre transaction de façon non conflictuelle (i.e., verrouillages read/read) alors le verrouillage peut être partagé et la transaction peut continuer.*
   - *d) Si R_k a déjà été verrouillée par la même transaction alors elle garde le verrouillage le plus fort*[3] *et la transaction continue.*
2. *Quand une transaction est terminée ou annulée, toutes les ressources qu'elle a verrouillées sont déverrouillées.*

Puisque les ressources sont éventuellement dupliquées, un *lock(R_k,x)* consiste à diffuser une demande de verrouillage de type x à toutes les copies de R_k et à attendre les réponses. Celles-ci sont envoyées selon le principe du verrouillage décrit précédemment. Quand une transaction obtient toutes les réponses à ses requêtes de verrouillage, elle peut s'exécuter. L'exécution d'un *unlock(R_k)* consiste à diffuser une demande de déverrouillage à toutes les copies de R_k.
Le verrouillage à deux phases assure que si une transaction est terminée, son exécution est correcte (i.e., les contraintes sont respectées). Cependant, il n'assure pas que la transaction se termine. Il peut y avoir *interblocage* dû au verrouillage des ressources. Une solution consiste à utiliser un mécanisme d'estampillage [BER 87] : les requêtes de verrouillage sont estampillées et l'ordre total défini par les estampilles représente les priorités entre les transactions. La transaction ayant la plus faible priorité sera suspendue[4].

Pour intégrer l'approche du verrouillage à deux phases dans la stratégie de synthèse, il suffit de considérer que toutes les transitions de service qui font accès à des ressources sont a priori conflictuelles. Ainsi, par rapport à l'approche précédente, il suffit de remplacer les demandes d'entrée en section critique et la sortie de la section critique par respectivement les deux phases du verrouillage.
Une application de cette approche concerne les systèmes où les demandes de service arrivent de manière dynamique, comme c'est le cas des systèmes de gestion des bases de données réparties et dupliquées.

2.3.2.3 Discussion

La deuxième approche est plus coûteuse en messages que la première. En effet, si on considère que les ressources sont dupliquées sur tous les sites de l'architecture cible et que chaque

3. Un verrouillage en écriture est plus fort qu'un verrouillage en lecture.

4. L'intégration de la gestion des interblocages ne doit pas remettre en cause les contraintes de cohérence C1 et C2.

transition effectue au moins un accès à une ressource alors la première approche requiert pour chaque transition conflictuelle $3(N-1)$ messages, et la deuxième approche requiert pour toute transition $3N$ messages, où N est le nombre de sites. En plus, la gestion des interblocages augmente le nombre de messages nécessaires dans la deuxième approche.

2.3.2.4 Construction du graphe des conflits

Il est facile de voir que le problème de construction du graphe des conflits associé à une spécification de service $SPEC_s$ est équivalent au problème de construction de la relation de concurrence dans un réseau de Petri [KOV 92]. Un algorithme polynômial pour le calcul de la relation de concurrence s'avère ainsi très important pour la mise en pratique de la première approche. Pour la classe des réseaux de Petri à choix libre, vivants et bornés, il existe des algorithmes polynômiaux permettant le calcul de cette relation [YEN 91][KOV 92][KOV 95]. Nous adoptons un algorithme dont la complexité est en $O(N^3)$ où N est le nombre total des places et transitions dans le réseau de Petri [KOV 95].

Algorithme de calcul de la relation de concurrence (C)

Entrée : un réseau à choix libre, borné et vivant $\langle P, T, A, W, M_0 \rangle$.
Sortie : $\mathrm{C} \subseteq X \times X$, où $X = P \cup T$.
debut
C:= $\{(p,p') \mid M_0 \geq M_p + M_{p'}\} \cup \bigcup_{t \in T} t\bullet \times t\bullet$;
où M_p désignent un marquage où seule la place p est marquée.
$A := \{(p,p') \mid p' \in (p\bullet)\bullet\}$;
E :=C $\cap (X \times P)$;
tant que $E \neq \emptyset$ **faire**
choisir $(x,p) \in E$; $E := E - \{(x,p)\}$;
choisir $t \in p\bullet$;
si $\{x\} \times \bullet t \subseteq$ C **alors**
$E := E \cup (\{(x,p') \mid (p,p') \in A\} - \mathrm{C})$;
C:=C $\cup \{(x,p') \mid (p,p') \in A\}$;
fsi
fait
fin

La relation trouvée est ensuite restreinte aux couples de transitions en conflit (cf. Définition 14). La construction du graphe GC se fait alors en $O(X^3)$, avec $X = |P| + |T| + N_t$, où N_t désigne le nombre total des termes spécifiés dans $SPEC_s$.

2.3.3 Synthèse du flot de données

Pour qu'une transition de service puisse être exécutée dans son site d'allocation, il faudra que toutes les données qu'elle requiert en entrée soient disponibles dans ce site au moment où elle est sensibilisée. On appelle flot de données entre les entités de protocole, les messages de données qui circulent entre ces entités afin de permettre l'exécution des transitions de

service. Deux solutions sont possibles pour récupérer les données nécessaires à l'exécution d'une transition de service t :

- S1 : attendre jusqu'à ce que la transition t soit sensibilisée, et ensuite demander explicitement toutes les données nécessaires.

- S2 : quand une transition se termine, au lieu qu'elle transfère simplement le contrôle aux transitions successeurs, elle leur transmet en plus des données nécessaires à leur exécution. Ce transfert peut s'effectuer soit directement (si les données sont disponibles localement à la transition), soit (dans le cas contraire) indirectement, à travers un site intermédiaire.

Si la transition t ne possède pas de clause conditionnelle, la solution S1 est toujours correcte. Par contre la solution S2 ne marcherait pas. Un problème concernant la cohérence des données transférées se pose si la transition t possède plusieurs prédécesseurs concurrents. En effet, quand il y a plusieurs transitions prédécesseurs qui effectuent des modifications sur une ressource donnée de façon concurrente et que cette ressource est utilisée par la transition t, la transition t ne pourrait accéder à cette ressource que lorsque toutes ces transitions prédécesseurs seraient terminées. Donc, il est impossible de réaliser le transfert de la valeur de cette ressource au niveau des transitions prédécesseurs. De plus, si la transition t est conflictuelle, la demande d'entrée en section critique, ne peut être faite que lorsque toutes les transitions prédécesseurs soit terminées, donc, il est également impossible de l'effectuer au niveau des transitions prédécesseurs. Ce qui implique que S2 ne marcherait pas.

Si la transition t possède une clause conditionnelle, elle participe forcément dans une structure sélective. Dans ce cas, la solution S1 n'est pas correcte pour le transfert des données utilisées dans la clause conditionnelle. Car l'évaluation de la condition de franchissement d'une transition ne doit pas changer le marquage des places en entrée. En effet, la partie condition peut faire référence à des ressources distantes. Dans ce cas, les raffinements effectués par les règles de synthèse afin de collecter les valeurs de ces ressources peuvent introduire des transitions précédant l'évaluation de la condition. Ces transitions peuvent donc changer le marquage des places en entrée de t. Ce qui changerait forcément la sémantique du réseau de Petri.
Par contre la solution S2 serait correcte puisqu'au moment où la transition est sensibilisée, toutes les données nécessaires à l'évaluation de la condition sont disponibles. Mais, le problème engendré par des transitions prédécesseurs concurrentes se pose toujours. C'est pour cela que nous avons posé les restrictions R1 et R3 qui, conjointement, supposent que le réseau de Petri sous-jacent à une spécification de service est sauf et à choix libre. On peut facilement montrer que pour un tel réseau les transitions prédécesseurs de la transition t ne peuvent pas être concurrentes.

Pour réaliser la synthèse du flot de données nous modélisons les entrées d'une transition de service t par un arbre attribué, et nous proposons un algorithme qui, parcourant cet arbre, synthétise un ensemble de blocs de réseaux de Petri coopérant dont la tâche est de rendre disponibles à la transition t les données spécifiées dans cet arbre. Les attributs d'un arbre

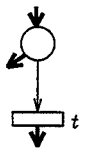

FIG. 2.4 – *Une structure contenant une transition t alternative*

servent, entre autres, à combiner efficacement les solutions S1 et S2.

Définition 18 (Arbre des entrées d'une transition de service) *L'arbre des entrées associé à une transition de service est un couple $\langle N, F\rangle$, où N est l'ensemble des noeuds et F une fonction qui associe à chaque noeud l'ensemble de ses fils. F^+ est la fermeture transitive de F. $F^+(n)$ donne tous les descendants du noeud n (n non compris). Un noeud n est un 6-uplet $\langle nat, id, SE, TS, TR, TE\rangle$ où $n.nat \in \{RAC, ACT, COND, OUT, TERM\}$, désigne la nature du noeud. La racine RAC est un noeud particulier qui possède trois fils de nature $COND$, ACT et OUT, désignant respectivement les parties "condition", "action" et "output" de la transition de service. Tous les autres noeuds sont des termes d'entrée de la transition ($n.nat = TERM$); $n.id$ désigne l'identité du terme. Soient R_k une ressource, f une fonction d'arité i liée à R_k et $m_1, ..., m_i$ des termes. Si un noeud est associé au terme $R_k.f(m_1, ..., m_i)$, les fils de ce noeud sont respectivement associés aux termes $m_1, ..., m_i$. Les autres champs du noeud n définissent ses attributs :*

- *$n.SE$: est le site d'évaluation du noeud n.*
- *$n.TS$: désigne l'ensemble des couples définis par la transition de service qui doit être synthétisée pour évaluer le noeud n et le site initiateur de l'évaluation.*
- *$n.TE$: est la transition d'émission de la valeur du noeud n.*
- *$n.TR$: est la transition de réception de la valeur du noeud n.* ■

Le calcul de l'attribut SE consiste à fixer pour chaque noeud de l'arbre son site d'évaluation. Le site d'évaluation d'un noeud peut être arbitraire, mais il serait judicieux de choisir un site de telle sorte à minimiser les communications. Néanmoins, les problèmes d'optimisation n'étant pas abordés dans ce chapitre, nous nous limitons à une stratégie simple de choix du site d'évaluation. Cette stratégie consiste à sélectionner le site qui peut évaluer localement le noeud courant.
Le calcul de l'attribut TS pour un noeud donné consiste à déterminer d'une part les transitions de service qui doivent être synthétisées pour l'évaluation de ce noeud, et d'autre part, pour chacune de ces transitions, le site initiateur de l'évaluation qui lui correspond. A la différence du site d'évaluation, le site initiateur de l'évaluation ne peut pas être arbitraire, il dépend de l'allocation des transitions de service. Le calcul de cet attribut se fait selon les étapes suivantes :

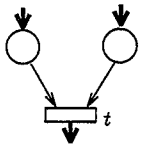

FIG. 2.5 – *Une structure contenant une transition t non alternative*

Calcul de l'attribut TS

Soit t une transition de service, deux cas sont possibles:

- *Cas 1 (la transition t possède une clause conditionnelle (cf. Figure 2.4))*
  - *Etape 1: Soit* $c \in N$ *tel que* $c.nat = COND$ *alors tout noeud* $n \in F^+(c)$ *est évalué selon la solution S2. Deux cas sont possibles:*
    - *Cas 1.1: si la transition t est conflictuelle sur le terme n.id alors* $n.TS = \bullet\bullet t \times \{Alloc(t)\}$
    - *Cas 1.2: si la transition t n'est pas conflictuelle sur le terme n.id alors* $n.TS = \{(t_k, Alloc(t_k)) \mid t_k \in \bullet\bullet t\}$
  - *Etape 2: Soit* $a, b \in N$ *tels que* $a.nat = ACT$ *et* $b.nat = OUT$ *alors tout noeud* $n \in F^+(a) \cup F^+(b)$ *est évalué selon la solution S1;* $n.TS = \{(t, Alloc(t))\}$
- *Cas 2 (la transition t ne possède pas de clause conditionnelle)*
  - *Cas 2.1: si la transition t est alternative (i.e.,* $|(\bullet t)\bullet| > 1$*) (cf. Figure 2.4) alors tout noeud* $n \in N$ *tel que* $n.nat = TERM$ *est évalué selon la solution S1;* $n.TS = \{(t, Alloc(t)\}$.
  - *Cas 2.2: si la transition t n'est pas alternative (cf. Figure 2.5) alors les noeuds* $n \in N$ *sont parcourus de bas en haut et de gauche à droite. Pour chaque itération, les cas suivants sont possibles:*
    - *Cas 2.2.1: s'il existe un conflit sur le terme n.id entre t et une autre transition, ou bien il existe un conflit de type E/E sur n.id entre deux transitions prédécesseur de t, ou bien t n'a pas de prédécesseurs alors le noeud n est évalué selon la solution S1;* $n.TS = \{(t, Alloc(t)\}$.
    - *Cas 2.2.2: sinon, si le noeud courant n'est pas un noeud feuille alors si tous les noeuds* $n' \in F(n)$ *ont le même attribut TS alors* $n.TS = n'.TS$ *sinon* $n.TS = \{(t, Alloc(t)\}$. *Par contre, si le noeud courant est un noeud feuille alors* $n.TS = \{(t_k, Alloc(t_k)) \mid t_k \in (\bullet t_l)\bullet \cap \bullet\bullet t\}$ *où* $t_l \in \bullet\bullet t$ *(i.e., on choisit une transition* t_l *parmi les prédécesseurs de t ainsi que toutes ses alternatives).*

■

L'algorithme de synthèse du flot de données (noté SFD) construit un ensemble de blocs de réseaux de Petri qui coopèrent afin d'évaluer un noeud terme n donné de l'arbre des entrées d'une transition de service t, sachant que le site initiateur de la coopération est s.

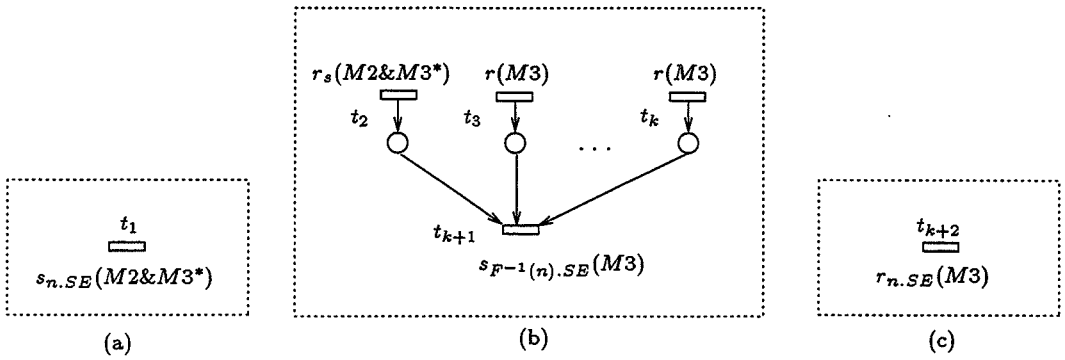

FIG. 2.6 – *Blocs construits à l'étape 5 de l'algorithme SFD (a) Bloc BP_1 (b) Bloc BP_2 (c) Bloc BP_3*

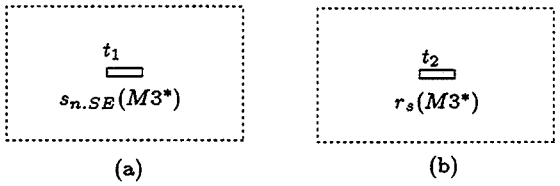

FIG. 2.7 – *Blocs construits à l'étape 6 de l'algorithme SFD (a) Bloc BP_1 (b) Bloc BP_2*

Algorithme SFD(n,s)*cet algorithme parcourt le sous-arbre dont la racine est n de bas en haut et de gauche à droite; pour chaque itération (dont le noeud courant est noté n) les étapes suivantes sont effectuées:*

- **Etape 1 :** *Si le noeud n est de nature COND, OUT ou RAC alors ignorer le noeud et passer à l'itération suivante sinon aller à l'étape 2.*
- **Etape 2 :** *Vérifier si la transition en cours de raffinement t_c est bien la transition qu'il faut synthétiser pour évaluer le noeud n à l'initiative du site s ; si $(t_c, s) \in n.TS$ alors aller à l'étape 3, sinon ignorer le noeud n et passer à l'itération suivante.*
- **Etape 3 :** *Vérifier si le noeud n peut être évalué localement au site initiateur s ; si c'est le cas ($n.SE = s$) alors le noeud n peut être évalué directement par le site s, passer à l'itération suivante, sinon aller à l'étape 4.*
- **Etape 4 :** *Si le noeud courant est un terme et son site d'évaluation est différent de celui de son père ($n.nat = TERM$ et $n.SE \neq F^{-1}(n).SE$) alors aller à l'étape 5 sinon aller à l'étape 6.*
- **Etape 5 :**
  - *Construire un message de type M2&M3* (cf. Table 2.1); l'émetteur est s, le récepteur est $n.SE$, le terme dont la valeur est requise est n.id et le site qui doit recevoir cette valeur est $F^{-1}(n).SE$. Le message M3* contient toutes les valeurs*

des termes qui peuvent être évaluées par s et dont le site $n.SE$ aura besoin pour évaluer le noeud n ; il s'agit des valeurs des noeuds $n_i \in F(n)$ tels que $n_i.SE = s$.

- *Soit BP_1 le bloc de réseaux de Petri à construire dans le site s.*
- *Construire une transition t_1 dans le bloc BP_1 pour l'émission du message M2&M3* (cf. Figure 2.6a).*
- *Soit BP_2 le bloc de réseaux de Petri à construire dans le site $n.SE$.*
- *Construire dans BP_2 le sous-réseau donné dans la Figure 2.6b. La transition t_2 spécifie la réception du message M2&M3*. Les transitions t_3 à t_k sont des transitions construites lors des itérations précédentes, elles sont récupérées par l'attribut TR ; pour tout noeud $n_i \in \{n_0, ..., n_{k-3}\}$ tel que $n_i \in F(n)$ et $n.SE \neq n_i.SE$, $t_{i+3} = n_i.TR$. La transition t_{k+1} est la transition d'émission de la valeur du terme $n.id$ au site $F^{-1}(n).SE$.*
- *Calculer l'attribut TE pour le noeud n ; $n.TE = t_{k+1}$.*
- *Soit BP_3 le bloc de réseaux de Petri à construire dans le site $F^{-1}(n).SE$.*
- *Construire dans BP_3 la transition de réception t_{k+2} correspondant à t_{k+1} (cf. Figure 2.6c).*
- *Calculer l'attribut TR pour le noeud n ; $n.TR = t_{k+2}$.*
- *passer à l'itération suivante.*

- **Etape 6 :**

  - *Construire un message de type M3* (cf. Table 2.1) dont l'émetteur est s et le récepteur est $n.SE$, ce message contient toutes les valeurs des termes qui peuvent être évaluées par s et dont le site $n.SE$ aura besoin pour évaluer le noeud n ; il s'agit des valeurs des noeuds $n_i \in F(n)$ tels que $n_i.SE = s$.*
  - *Soit BP_1 le bloc de réseaux de Petri à construire dans le site s.*
  - *Construire dans le bloc BP_1 une transition t_1 pour l'émission du message M3* (cf. Figure 2.7a).*
  - *Soit BP_2 le bloc de réseaux de Petri à construire dans le site $n.SE$.*
  - *Construire dans BP_2 une transition t_2 de réception correspondant à t_1 (cf. Figure 2.7b).*
  - *Pour tout noeud $n_i \in F(n)$ tel que $n.SE = s$, calculer les attributs TE et TR ; $n_i.TE = t_1$ et $n_i.TR = t_2$.*
  - *passer à l'itération suivante.* ∎

2.3.4 L'ensemble des règles de synthèse

Soit t^i une transition de $SPEC_s$ qui est allouée au site i (i.e., $Alloc(t^i) = i$), on note EP^i l'entité de protocole qu'il faut construire dans le site i.
Initialement, les sous-réseaux de Petri $SP^k(t^i)$, tels que $k \neq i$, sont vides ($SP^k(t^i) = \varepsilon, \forall k \neq i$), et le sous-réseau $SP^i(t^i)$ est réduit à la méta-transition tr_0 attribuée par la transition t^i (on note $SP^i(t^i) = tr_0(t^i)$). En utilisant le concept de règle de synthèse défini dans la

Type	*Sémantique du message*	*Paramètres*
M1	Message de synchronisation	identité de la transition de service en cours de raffinement (id_transition)
M2	Demande de la valeur d'un terme pour le compte d'une entité de protocole EP^k	id_transition, identité du terme demandé (id_terme) et l'identité de l'entité de protocole destinataire (EP^k)
M3	Message de données	id_transition, id_terme, valeur du terme
M4	Demande d'exécution d'une action distante	id_transition
M5	Indication du résultat de l'exécution d'une action distante (causée par M4)	id_transition
M6	Demande de mise à jour d'une ressource dupliquée (causée par M4)	id_transition
M7	Indication de fin de mise à jour d'une ressource dupliquée (causée par M6)	id_transition

M2&M3* : désigne un message de type M2 et zéro, un ou plusieurs messages de type M3.
M3&M3*, M2&M3*, M6&M3*, M5&M7, M5&M7&M3* : dénotent les messages de type composé.

TAB. 2.1 – *Types de messages échangés entre les entités de protocole*

section 2.3.1, nous proposons dans ce qui suit un ensemble complet de règles contextuelles permettant de construire, à partir de la transition t^i, n sous-réseaux de Petri définissant l'implantation de cette transition sur l'architecture cible.

Les différents types de message qui peuvent être échangés entre les entités de protocole sont donnés par le tableau 2.1. Des messages de types composés peuvent aussi être échangés ; par exemple un message de type M6&M3* signifie une demande de mise à jour d'une ressource avec les données nécessaires pour effectuer cette mise à jour. Ces différents types de message sont utilisés par les règles de synthèse contextuelles.

La règle RS1 traite l'entête d'une transition t^i, c'est à dire la condition de franchissement et l'événement en entrée. Les règles RS2 à RS4 traitent la partie action. La règle RS5 traite la partie "output". Les règles RS6 et RS7 traitent la synthèse du flot de données entre les entités de protocole. La règle RS8 traite la synthèse du flot de contrôle. Enfin, les règles RS9 et RS10 traitent, respectivement, la demande d'entrée en section critique et la libération d'une section critique.

Règle RS1 (entête)
Pour une transition t^i qui a été évaluée franchissable, trois cas sont possibles :

- Cas 1 : si la transition t^i n'appartient pas à une structure sélective alors l'entité EP^i commence par consommer l'événement en entrée ($\mathcal{X}(t^i).in$) et effectuer, éventuellement, une demande d'entrée en section critique (point de raffinement tr_E, voir Figure 2.8a).

- Cas 2 : si la transition t^i appartient à une structure sélective centralisée alors l'entité EP^i commence par consommer l'événement en entrée ($\mathcal{X}(t^i).in$) puis libère, éven-

tuellement, toutes les sections critiques demandées pour les transitions alternatives[5] (points de raffinement tr_S, voir Figure 2.8a).

- Cas 3 : si la transition t^i appartient à une structure sélective distribuée alors l'entité EP^i commence par libérer, éventuellement, toutes les sections critiques demandées pour les transitions alternatives locales[6] (points de raffinement tr_S, voir Figure 2.8a).

Les transformations de RS1 sont (cf. Figure 2.8a) :
$SP^i(t^i) = [tr_0/BP3]SP^i(t^i)$ (Cas 1);
$SP^i(t^i) = [tr_0/BP1]SP^i(t^i)$ (Cas 2);
$SP^i(t^i) = [tr_0/BP2]SP^i(t^i)$ (Cas 3);
Le prédicat associé à RS1 est la constante "VRAI".

Règle RS2 (action locale)
Si la partie "action" de la transition t^i est une opération locale alors :

1. l'entité EP^i demande éventuellement certaines ressources nécessaires pour effectuer cette opération (messages de type M2). Les blocs de réseaux de Petri réalisant le transfert de données sont construits en utilisant l'algorithme SFD appliqué au noeud de nature ACT de l'arbre des entrées de t^i, sachant que le site initiateur du transfert est i.

2. quand EP^i reçoit toutes les données nécessaires (messages de type M3), deux cas se présentent :

   - cas 1 : si $\mathcal{X}(t^i).act$ est une opération de lecture op sur une ressource locale R_k ou bien une opération locale op non liée à une ressource alors l'entité EP^i effectue localement l'opération et entame la partie "output".
   - cas 2 : si $\mathcal{X}(t^i).act$ est une opération d'écriture sur une ressource locale alors l'entité EP^i effectue localement l'opération et diffuse un message contenant la demande de mise à jour de R_k ainsi que les données nécessaires (message de type M6&M3*) à toutes les entités détenant une copie de R_k. Lorsqu'elle reçoit tous les messages accusant la fin de la mise à jour (messages de type M7), elle entame la partie "output".

Les transformations de RS2 (dans le cas 2) sont (cf. Figure 2.8b) :
$SP^i(t^i) = [tr_1/BP1]SP^i(t^i)$;
$SP^j(t^i) = SP^j(t^i) + BP2$ (pour toute entité EP^j qui reçoit M2&M3*);
$SP^k(t^i) = SP^k(t^i) + BP3$ (pour toute entité EP^k qui reçoit un message de type M6&M3*);
Le prédicat associé à RS2 est : $(\mathcal{X}(t^i).act = R_k.op(...)$ et $i \in Alloc(R_k))$ ou $\mathcal{X}(t^i) = op(...)$.

Règle RS3 (action distante)
Si la partie "action" de la transition t^i possède une opération op sur une ressource R_k distante ($i \notin Alloc(R_k)$) alors :

5. Les transitions ayant une place en entrée commune avec t^i.
6. Les transitions du site i ayant une place en entrée commune avec t^i.

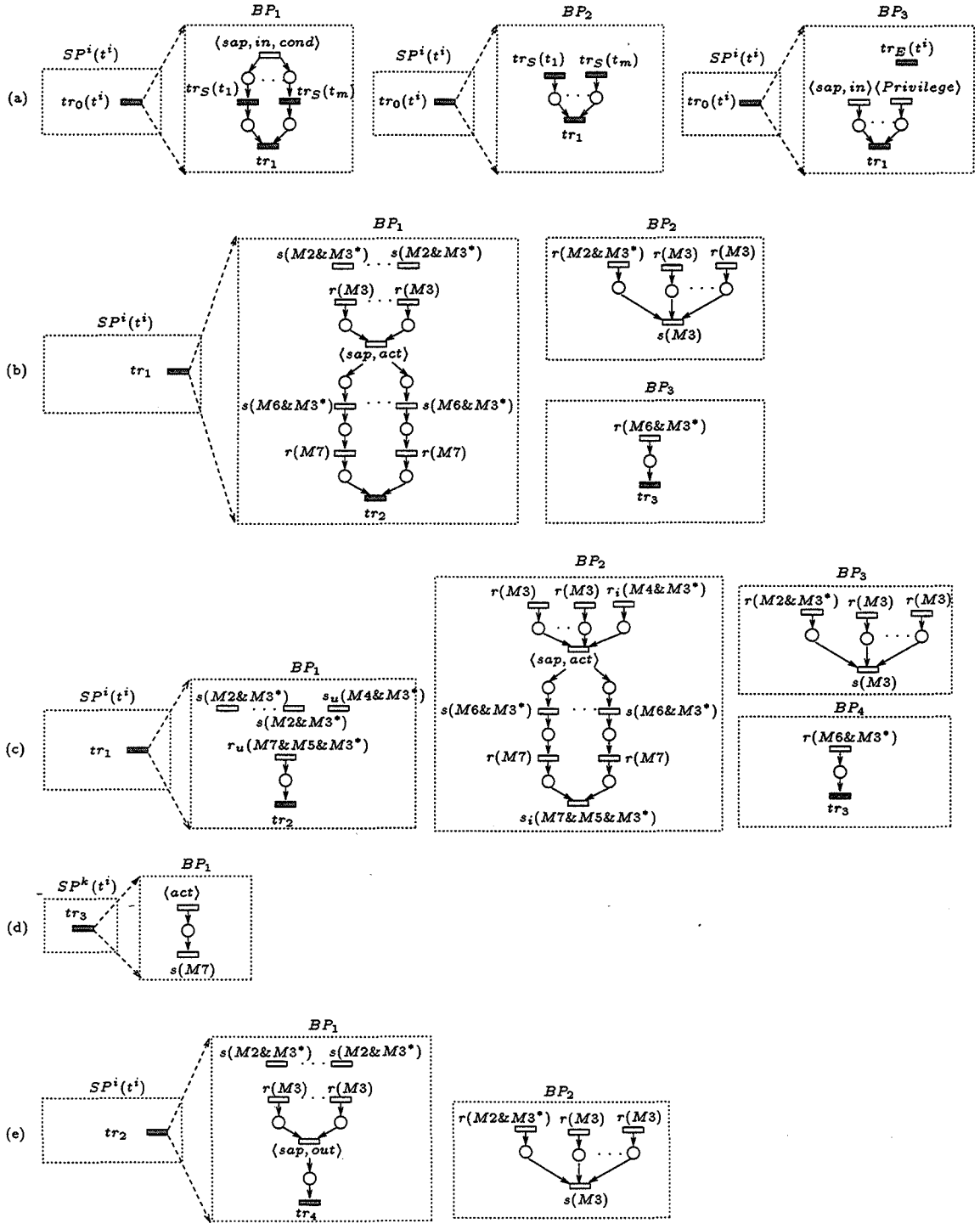

FIG. 2.8 – *Règles de synthèse*

1. une entité EP^u, telle que $u \in Alloc(R_k)$, est choisie afin d'effectuer cette opération,

2. l'entité EP^i envoie à EP^u une requête d'exécution de l'opération *op* (message de type M4) avec les données qui lui sont nécessaires (messages de type M2 et M3) puis se met en attente des résultats. Les blocs de réseaux de Petri réalisant le transfert de données sont construits en utilisant l'algorithme SFD appliqué au noeud de nature ACT de l'arbre des entrées de t^i, sachant que le site initiateur du transfert est i.

3. Quand l'entité EP^u reçoit la requête ainsi que toutes les données nécessaires (messages de type M4 et M3) pour effectuer l'opération *op*, deux cas se présentent :

   - cas 1 : si $R_k.op.nat = Lecture$ alors l'entité EP^u effectue l'opération et renvoie les résultats à EP^i (message de type M5&M3*),
   - cas 2 : si $R_k.op.nat = Ecriture$ alors l'entité EP^u effectue l'opération localement, et diffuse un message, contenant la demande de mise à jour de R_k et les données nécessaires (message de type M6&M3*), à toutes les entités qui détiennent une copie de la ressource R_k. Ainsi, quand elle reçoit tous les messages accusant la fin de la mise à jour (message de type M7), elle envoie un accusé de fin de mise à jour ainsi que les résultats à l'entité EP^i (message de type M7&M5&M3*).

Les transformations de RS3 (dans le cas 2) sont (cf. Figure 2.8c) :
$SP^i(t^i) = [tr_1/BP1]SP^i(t^i)$;
$SP^u(t^i) = SP^u(t^i) + BP2$;
$SP^j(t^i) = SP^j(t^i) + BP3$ (pour toute entité EP^j qui reçoit M2&M3*);
$SP^k(t^i) = SP^k(t^i) + BP4$ (pour toute entité EP^k qui reçoit M6&M3*)
Le prédicat associé à RS3 est : $\mathcal{X}(t^i).act = R_k.op(...)s$ et $i \notin Alloc(R_k)$.

Règle RS4 (mise à jour)
Quand une entité EP^k reçoit une demande de mise à jour de sa ressource locale R_k (message de type M6&M3*) de la part d'une entité EP^i, elle effectue localement cette opération et envoie un accusé de fin de mise à jour (message de type M7) à l'entité EP^i. La transformation de RS4 est (cf. Figure 2.8d) :
$SP^k(t^i) = [tr_3/BP1]SP^k(t^i)$;
Le prédicat associé à RS4 est la constante "VRAI".

Règle RS5 (clause "output")
Si la transition t^i a besoin de ressources en entrée pour effectuer sa partie "output", elle demande ces ressources auprès des entités qui les détiennent (messages de type M2&M3*). Les blocs de réseaux de Petri réalisant ce transfert de données sont construits à l'aide de l'algorithme SFD appliqué au noeud de nature OUT de l'arbre des entrées de t^i, sachant que le site initiateur du transfert est i. Les transformations de RS5 sont (cf. Figure 2.8e) :
$SP^i(t^i) = [tr_2/BP1]SP^i(t^i)$;
$SP^j(t^i) = SP^j(t^i) + BP2$ (pour toute entité EP^j qui reçoit M2&M3*);
Le prédicat associé à RS5 est la constante "VRAI".

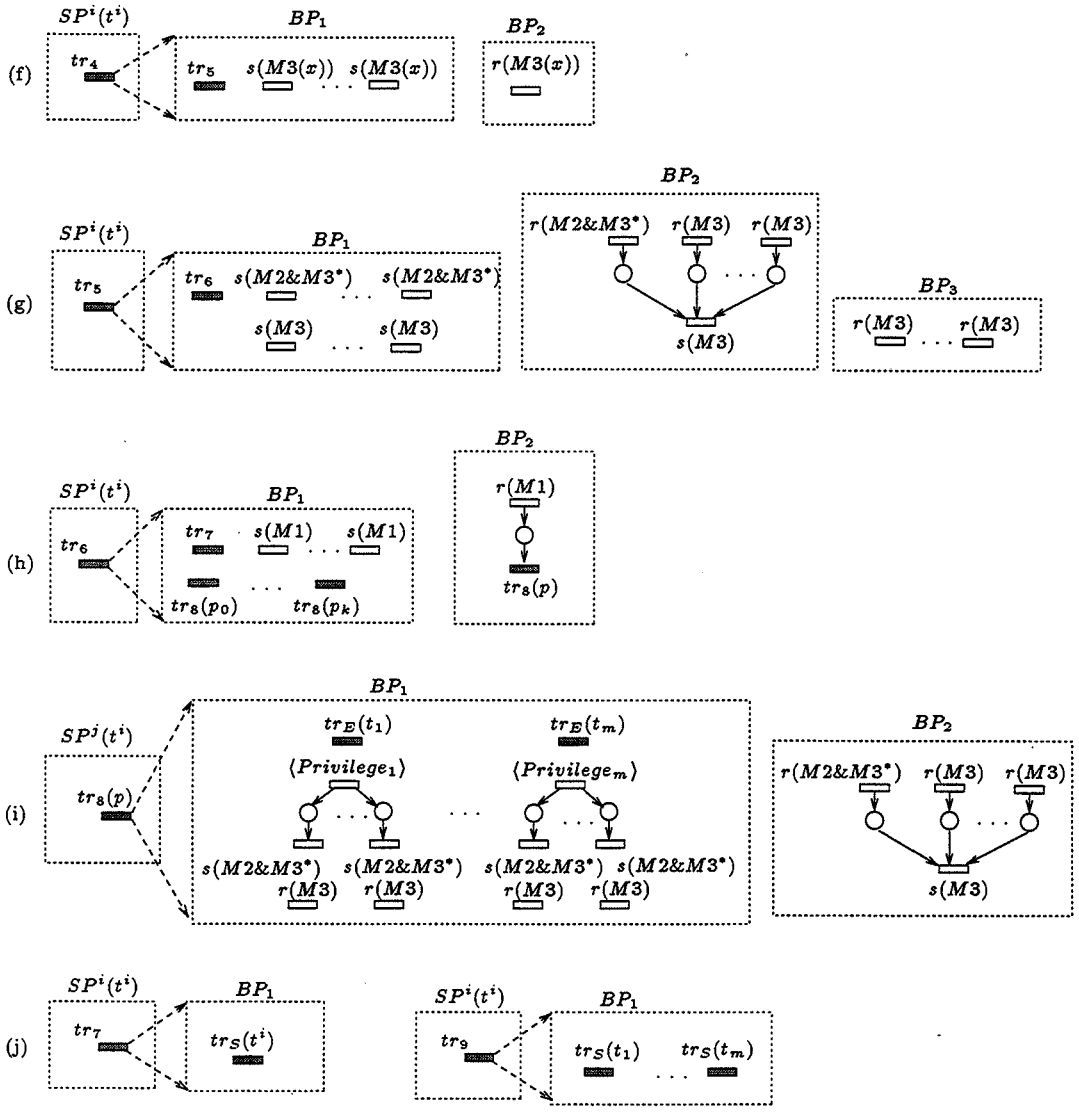

FIG. 2.9 – *Règles de synthèse (suite)*

Règle RS6 (premier cas de transfert de données)
Si la transition t^i possède une variable x en sortie et s'il existe une transition t^j ($j \neq i$ et $t^j \in t^{i+}$) telle que :

- x est en entrée de t^j,
- et x n'apparaît pas en sortie avant t^j ; $\forall t' \in ((t^{i+}) \cap ({}^+t^j))$, x n'apparaît pas en sortie dans t',

alors l'entité EP^i doit envoyer la valeur de x (message de type M3) à EP^j. Les transformations de RS6 sont (cf. Figure 2.9f) :
$SP^i(t^i) = [tr_4/BP1]SP^i(t^i)$;

$SP^j(t^i) = SP^j(t^i) + BP2$ (pour toute entité EP^j qui reçoit M3(x));
Le prédicat associé à RS6 est la constante "VRAI".

Règle RS7 (deuxième cas de transfert de données)
Quand une transition t^i se termine, des ressources peuvent être transférées afin de permettre l'exécution des transitions successeurs. Pour ce faire, pour toute transition $t^j \in t^i \bullet\bullet$, on applique l'algorithme SFD à l'arbre des entrées de t^j, sachant que le site initiateur du transfert est i. Les transformations de RS7 sont (cf. Figure 2.9g) :
$SP^i(t^i) = [tr_5/BP1]SP^i(t^i)$;
$SP^k(t^i) = SP^k(t^i) + BP2$ (pour toute entité EP^k qui reçoit M2&M3*);
$SP^j(t^i) = SP^j(t^i) + BP3$ (pour toute entité EP^j telle que $t^j \in t^i \bullet\bullet$);
Le prédicat associé à RS7 est la constante "VRAI".

Règle RS8 (transfert de contrôle)
Quand la transition t^i se termine, l'entité EP^i transfert le contrôle en envoyant des messages de synchronisation à certaines transitions successeurs t^j telles que $j \neq i$. Le transfert de contrôle s'effectue de la manière suivante : Soit $p \in t^i\bullet$, si toutes les transitions de $p\bullet$ sont allouées à un même site j, alors l'entité EP^i envoie un message de synchronisation à EP^j (message de type M1) sinon il s'agit d'une place particulière dont le traitement est donné en section 2.3.5. Les transformations de RS8 sont (cf. Figure 2.9h) :
$SP^i(t^i) = [tr_6/BP1]SP^i(t^i)$;
$SP^j(t^i) = SP^j(t^i) + BP2$ (pour toute entité EP^j qui reçoit M1);
L'attribut p_m $(1 \leq m \leq k)$ d'une méta-transition tr_8 de BP_1 définit une place $p_m \in t^i\bullet$ telle que p_m est une place à choix centralisé local au site i. L'attribut p de la méta-transition tr_8 de BP_2 définit une place $p \in t^i\bullet$ telle que p est une place à choix centralisé local à un site $j \neq i$. La méta-transition tr_8 est utilisée pour la demande d'entrée en section critique et aussi pour la demande des ressources nécessaires à l'exécution de la partie "condition" d'une transition de service. Le prédicat associé à RS8 est la constante "VRAI".

Règle RS9 (entrée en section critique)
Quand une entité EP^j reçoit le contrôle de la part d'une transition de service t^i (ou termine l'exécution d'une transition de service locale) pour exécuter une transition successeur appartenant à une structure sélective $p\bullet$[7], elle commence par effectuer une demande d'entrée en section critique pour toutes les transitions t^j de $p\bullet$ qui sont locales à EP^j et conflictuelles (point de raffinement tr_E). Lorsque EP^j aura l'autorisation d'entrée pour une transition t^j $(Privilege_j = VRAI)$, elle initie le transfert des ressources nécessaires à l'évaluation de la partie "condition" $\mathcal{X}(t^j).cond$. Ce transfert est synthétisé en appliquant l'algorithme SFD au noeud de nature $COND$ associé à l'arbre des entrées de la transition t^j, sachant que le site initiateur du transfert est j. Les transformations de RS9 sont (cf. Figure 2.9i) :
$SP^j(t^i) = [tr_8/BP1]SP^j(t^i)$;
$SP^k(t^i) = SP^k(t^i) + BP2$ (pour toute entité EP^k qui reçoit M2&M3*);
Le prédicat associé à RS9 est la constante "VRAI".

7. Cette situation arrive au moment où la méta-transition $tr_8(p)$ est sensibilisée (cf. Figure 2.9i).

Règle RS10 (sortie d'une section critique)
Une entité de protocole EP^i libère ses sections critiques dans les deux cas suivants :

- Cas 1 : Quand EP^i termine l'exécution d'une transition de service t^i conflictuelle, elle libère sa section critique (point de raffinement tr_S).
- Cas 2 : Quand EP^i découvre que toutes les transitions de service locales appartenant à une structure sélective distribuée sont non franchissables, elle libère toutes les sections critiques demandées pour ces transitions (points de raffinement tr_S).

Les transformations de RS10 sont (cf. Figure 2.9j) :
$SP^i(t^i) = [tr_7/BP1]SP^i(t^i)$ (cas 1);
$SP^i(t^i) = [tr_9/BP1]SP^i(t^i)$ (cas 2);
Le prédicat associé à RS10 est la constante "VRAI".

2.3.5 Traitement des structures sélectives distribuées

Le problème des structures sélectives distribuées se pose lorsqu'il existe dans $SPEC_s$ une place p qui possède plusieurs transitions en sortie parmi lesquelles au moins deux transitions t^a et t^b sont allouées, respectivement, à deux sites différents a et b. Cette place est dite *à choix distribué*. Une fois que les transitions t^a et t^b deviennent franchissables, quelle est l'entité de protocole qui va décider de la transition à franchir? Pour résoudre ce problème, nous introduisons de nouvelles règles contextuelles pour les places à choix distribué. Ces règles consistent à synthétiser un mécanisme d'élection distribuée sur un anneau logique entre les entités de protocole qui participent dans le choix distribué.

Définition 19 (Place à choix distribué) *Une place p de $SPEC_s$ est dite à choix distribué si et seulement si :*

$$\exists t_1, t_2 \in p\bullet \mid \mathcal{X}(t_1).sap \neq \mathcal{X}(t_2).sap$$

■

Définition 20 (Place à choix déterministe) *Une place p de $SPEC_s$ est dite à choix déterministe si et seulement si : Pour tout état accessible de $SPEC_s$, si p est marquée alors il existe une et une seule transition $t \in p\bullet$ qui soit franchissable.* ■

Une place qui est à la fois à choix distribué et à choix déterministe est dite place de type CDD, alors qu'une place qui est à choix distribué non déterministe est appelée place de type CDN. Une place à choix distribué est a priori de type CDN. Nous supposons que le type d'une place à choix distribué est défini par le spécifieur.

2.3.5.1 Synthèse d'une place de type CDN

Considérons dans $SPEC_s$ la structure sélective de la Figure 2.10. Puisque le réseau sous-jacent à $SPEC_s$ est sauf, il ne peut y avoir qu'une seule transition franchissable parmi celles de $\bullet p$. Soit t^x une telle transition, le réseau sous-jacent à $SPEC_s$ étant à choix libre alors il ne peut pas y avoir d'autre places en entrée de t^a et t^b. On suppose que les sites d'allocation des transitions de $p\bullet$ constituent un anneau logique.
Nous donnons dans ce qui suit deux règles correspondant respectivement aux cas où le site

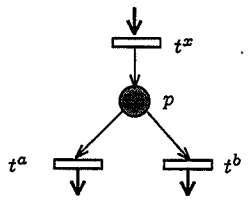

FIG. 2.10 – *Structure sélective distribuée*

de la transition t^x fait partie ou non de la structure sélective distribuée.
Initialement, pour tout site k le sous-réseau associé à la place p est réduit à la méta-place pr_0 attribuée par la place p (i.e., $\forall k \in \{1...n\}, SP_0^k(p) = pr_0(p)$).

Règle RS11 :
Quand la transition t^x se termine, l'entité EP^x envoie de manière non déterministe à l'une des entités EP^a ou EP^b, un message de synchronisation (de type $Jeton(p)$ paramétré par la place p). Si EP^a reçoit le jeton, elle effectue localement un choix non déterministe entre : d'une part, faire suivre le jeton à l'entité successeur EP^b et d'autre part, sélectionner la transition t^a. Si t^a est sélectionnée alors EP^a diffuse un message de synchronisation (de type $Elu(p, t^a)$ paramétré par la place p et la transition sélectionnée) à toutes les entités de protocole. Chaque entité qui reçoit le message $Elu(p, t^a)$ sélectionne localement le sous-réseau de Petri associé à t^a. Les transformations associées à cette règle de raffinements sont (cf. Figure 2.11a) :
$SP^x(p) = [pr_0/BP_1]SP^x(p)$;
$SP^a(p) = [pr_0/BP_2]SP^a(p)$;
$SP^b(p) = [pr_0/BP_3]SP^b(p)$;
$SP^y(p) = [pr_0/BP_4]SP^y(p)$ (pour toute autre entité EP^y);
Le prédicat de cette règle est : "p est de type CDN et $\forall t \in p\bullet, x \neq Alloc(t)$".

L'idée de cette règle est de dériver, à partir d'un choix distribué, un choix local qui est offert, dans un ordre circulaire pré-défini, à chaque site participant au choix distribué. Il a été montré dans [LAN 90] que l'expression d'un choix distribué est équivalente selon la relation de congruence observationnelle à l'expression qui consiste à l'offrir de manière circulaire.

Règle RS12 :
S'il existe $t \in p\bullet$ telle que $x = Alloc(t)$, l'entité EP^x effectue localement un choix entre d'une part envoyer le jeton (message de type Jeton(p)) à son successeur et d'autre part sélectionner la transition t^x. Chaque entité qui reçoit le jeton effectue localement le même choix. Si la transition t^x a été sélectionnée par l'entité EP^x alors EP^a diffuse à toutes les entités de protocole un message de synchronisation (de type $Elu(p, t^x)$ paramétré par la place p et la transition sélectionnée). Chaque entité qui reçoit le message $Elu(p, t^x)$ sélectionne localement le sous-réseau de Petri associé à t^x. Les transformations associées à cette règle sont (cf. Figure 2.11b) :
$SP^x(p) = [pr_0/BP_1]SP^x(p)$;
$SP^b(p) = [pr_0/BP_2]SP^b(p)$;

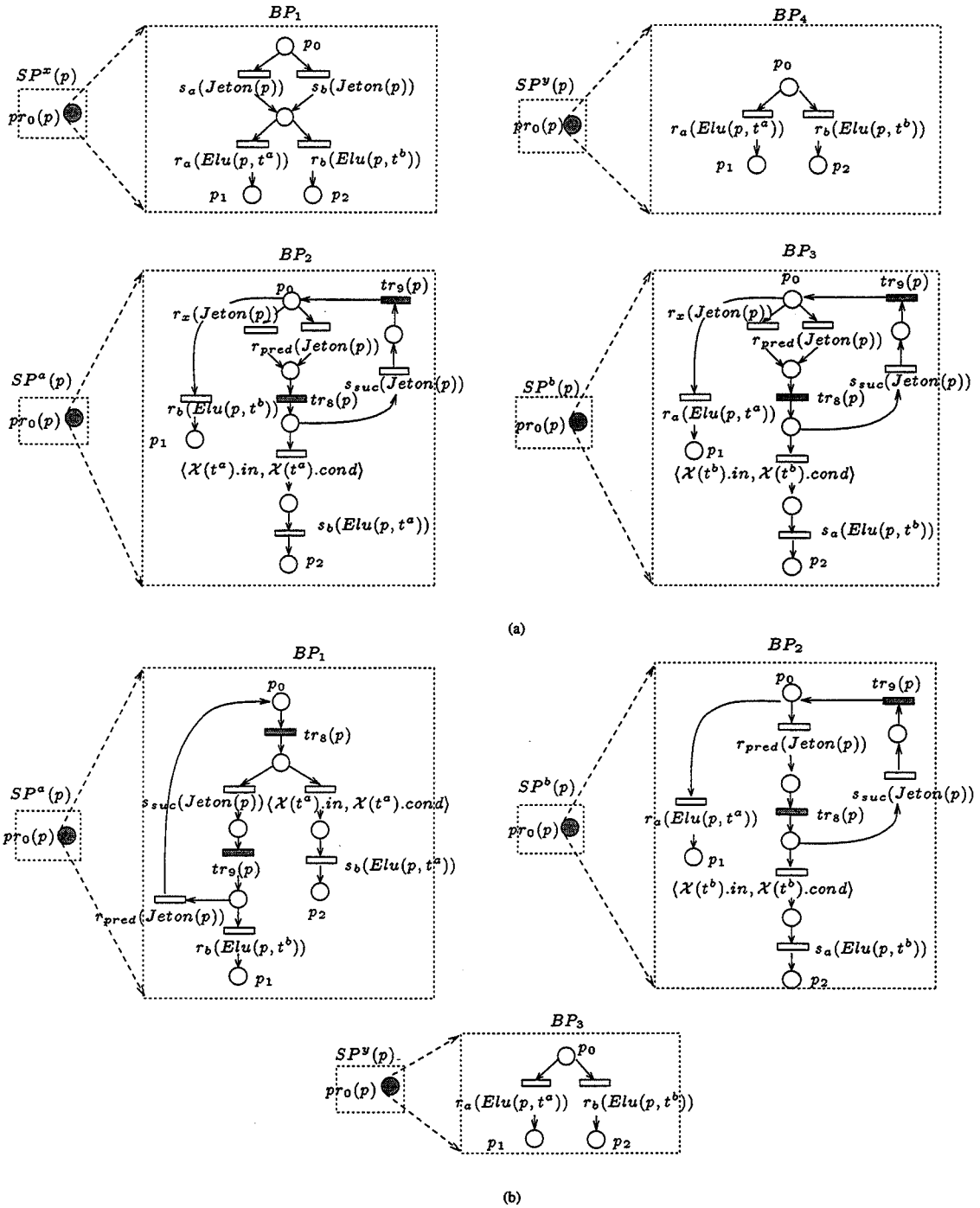

FIG. 2.11 – *Règles de synthèse d'une place de type CDN*

$SP^y(p) = [pr_0/BP_3]SP^y(p)$ (pour toute autre entité EP^y);
Le prédicat de cette règle est: "p est de type CDN et $\exists t \in p\bullet, x = Alloc(t)$".

Pour éviter que le jeton boucle indéfiniment entre les entités qui participent à un choix distribué, les transitions étiquetées par $\langle \mathcal{X}(t^a).in, \mathcal{X}(t^a).cond \rangle$ et $\langle \mathcal{X}(t^b).in, \mathcal{X}(t^b).cond \rangle$ devraient être plus prioritaires par rapport à $ts_9(p)$. En d'autres termes, si une transition est franchissable, on ne transmet pas le jeton pour tester le franchissement d'une autre transition, mais on sélectionne celle qui est déjà franchissable.

La synthèse d'une place de type CDN nécessite n messages où n est le nombre de sites de l'architecture cible; un message de type $Jeton$ et $n-1$ messages de type Elu. Ce nombre peut-être optimisé dans les cas suivants:

- Soit t^a la transition élue par EP^a. Si, pour une entité EP^y telle que $y \neq a$, $SP^y(t^a)$ est non vide, il n'est pas nécessaire que EP^a envoie un message de type Elu à EP^y, puisque EP^x participe à l'exécution de t^a.
- Si, pour une entité EP^y telle que $y \neq a$, $SP^y(t^a)$ est vide et aucune des transitions successeurs de t^a n'est allouée au site y, il n'est pas nécessaire que EP^a envoie aussi un message de type Elu à EP^y.

Donc, un message de type Elu n'est envoyé à une entité de protocole que lorsque celle-ci ne participe pas à l'exécution de la transition élue, et elle peut être l'initiatrice du franchissement d'une transition successeur de celle qui est élue (i.e., $SP^x(t^a) = \epsilon$ et $\exists t \in t^a \bullet\bullet \mid Alloc(t) = x$).

2.3.5.2 Synthèse d'une place de type CDD

Une place de type CDD définit un choix déterministe. Cela signifie, d'après la Définition 20 qu'il ne peut pas y avoir plus d'une transition franchissable simultanément une fois que la place de type CDD est marquée. Dans ce cas, il n'est pas nécessaire de faire circuler un jeton entre les sites qui participent au choix.
Nous donnons dans ce qui suit deux règles correspondant respectivement aux cas où le site de la transition t^x participe ou non dans le choix distribué.

Règle RS13:
Quand la transition t^x se termine, l'entité EP^x diffuse un message de synchronisation (de type $Jeton(p)$) à toutes les entités de protocole EP^y telles que $\exists t \in p\bullet, Alloc(t) = y$ et $y \neq x$. L'entité EP^x effectue ensuite un choix entre, d'une part, sélectionner t^x (si elle est franchissable) et d'autre part, se mettre en attente d'un message de type Elu. Chaque entité qui reçoit le jeton effectue le même choix. L'entité qui arrive à sélectionner localement sa transition, diffuse à toutes les autres entités un message de type Elu paramétré par la transition sélectionnée et la place qui est synthétisée. Les transformations associées à cette règle sont (cf. Figure 2.12b):
$SP^x(p) = [pr_0/BP_1]SP^x(p)$ (où x=a);
$SP^b(p) = [pr_0/BP_3]SP^b(p)$;
$SP^y(p) = [pr_0/BP_4]SP^y(p)$ (pour toute autre entité EP^y);
Le prédicat de cette règle est: "p est de type CDD et $\exists t \in p\bullet, x = Alloc(t)$".

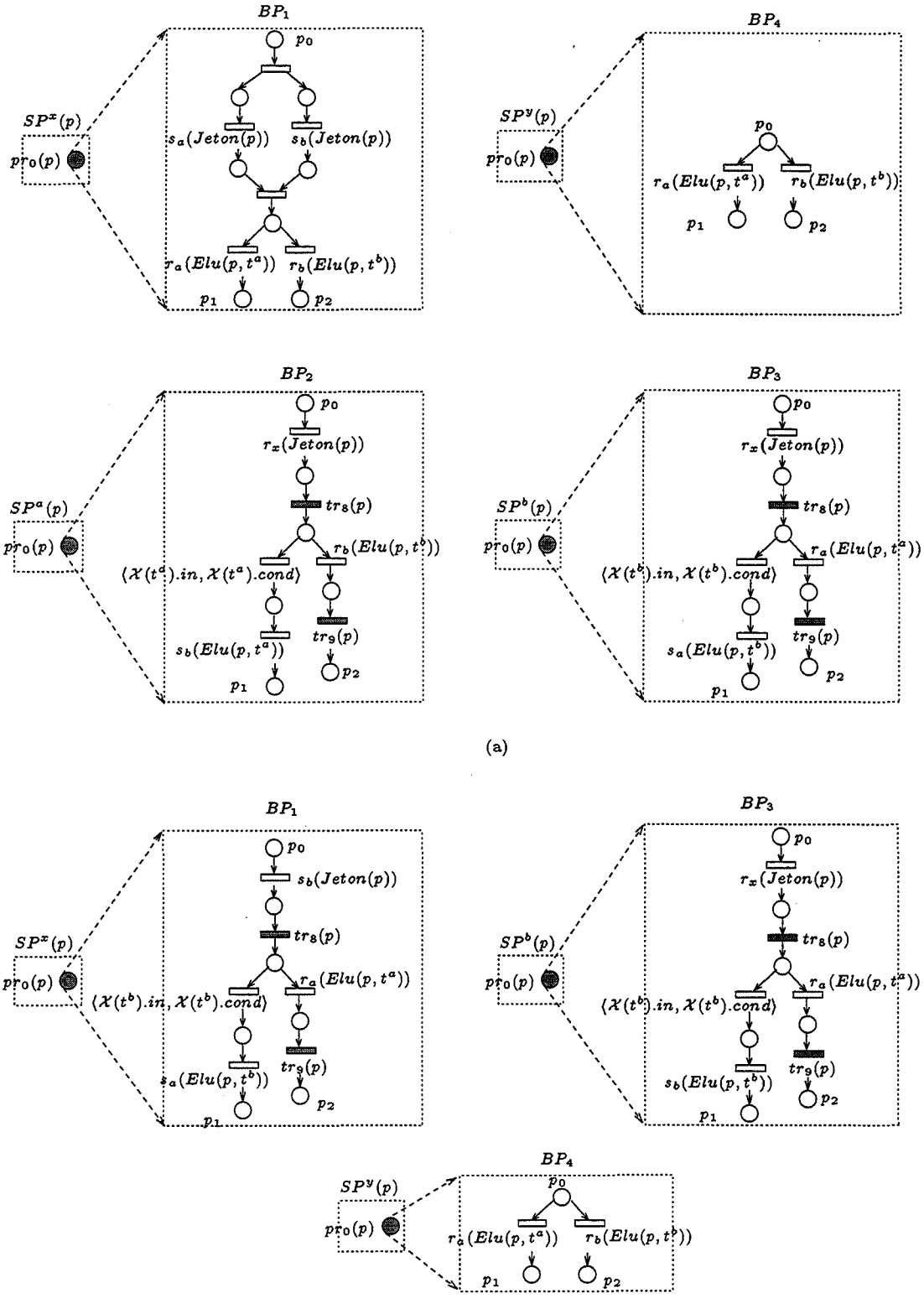

FIG. 2.12 – *Règles de synthèse d'une place de type CDD*

Règle RS14 :
Quand la transition t^x se termine, l'entité EP^x diffuse un message de synchronisation (de type $Jeton(p)$) à toutes les entités de protocole EP^y telles que $\exists t \in p\bullet, Alloc(t) = y$. L'entité EP^a, qui reçoit le jeton et qui arrive à sélectionner une transition franchissable t^a, diffuse un message $Elu(p, t^a)$ à toutes les entités de protocole.
Chaque entité qui reçoit le message $Elu(p, t^a)$, sélectionne localement le sous-réseau de Petri associé à t^a. Les transformations associées à cette règle sont (cf. Figure 2.12a):
$SP^x(p) = [pr_0/BP_1]SP^x(p)$;
$SP^a(p) = [pr_0/BP_2]SP^a(p)$;
$SP^b(p) = [pr_0/BP_3]SP^b(p)$;
$SP^y(p) = [pr_0/BP_4]SP^y(p)$ (pour toute autre entité EP^y);
Le prédicat de cette règle est: "p est de type CDD et $\forall t \in p\bullet, x \neq Alloc(t)$".

L'émission du message Elu dans les règles RS13 et RS14 peut-être guidée par la même optimisation que celle présentée pour les places de type CDN.
Pour que le choix déterministe soit correct, les transitions étiquetées par $\langle \mathcal{X}(t^a).in, \mathcal{X}(t^a).cond \rangle$ et $\langle \mathcal{X}(t^b).in, \mathcal{X}(t^b).cond \rangle$ devraient être plus prioritaires par rapport à $ts_9(p)$.

2.3.6 Règles de réduction d'une entité de protocole

Dans cette section nous présentons un ensemble de règles de réduction d'une entité de protocole construite par les règles de synthèse. Ces règles visent à éliminer les synchronisations non nécessaires, à regrouper certains messages et à éliminer les ϵ-transitions.

2.3.6.1 Elimination des synchronisations non nécessaires (règle RR1)

Soit t une transition de réception d'un message de synchronisation de type $M1$ ou ELU (i.e, $\mathcal{X}(t) = r_i(M1)$ ou $\mathcal{X}(t) = r_i(ELU)$) dans une entité de protocole EP^k. Si le message de synchronisation reçu concerne une transition de service ts, alors toutes les transitions successeurs de t dans EP^k appartiennent aux sous-réseaux de Petri implémentant les successeurs de ts. Le message de synchronisation assure qu'aucun de ces sous-réseaux n'entrera dans son état initial avant que ceux qui implantent ts n'aient atteint leur état final. Or, si toutes les transitions successeurs de t sont des réceptions de messages alors ces transitions ne peuvent s'exécuter qu'après la fin de tous les sous-réseaux implémentant ts. Donc, le message de synchronisation spécifié par t devient non nécessaire. L'élimination de cette synchronisation consiste à transformer respectivement la transition t et la transition d'émission correspondante dans EP^i en ϵ-transitions.

2.3.6.2 Regroupement de messages (règle RR2)

Certains messages émis par une entité de protocole à un même destinataire peuvent être regroupés en un seul message. Par exemple, l'émission successive (ou en parallèle) de deux messages de données (de type M3) à une même destination, peut-être remplacée par une émission d'un seul message de type M3&M3, obtenu par concaténation des deux messages de type M3. Pour réaliser ces regroupements, nous utilisons deux règles d'agglomération de transitions présentées dans [BER 86]. Ces règles consistent à réaliser de façon indivisible

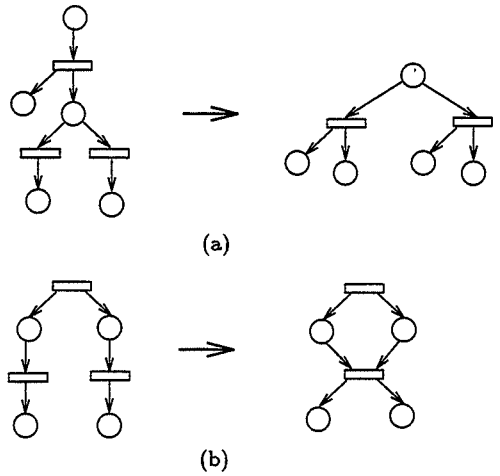

FIG. 2.13 – *Agglomérations de transitions (a) Post-agglomération (b) Agglomération latérale*

certaines séquences de franchissements représentant une suite d'actions élémentaires. Il a été montré dans [BER 86] que ces règles maintiennent les propriétés "borné" et "vivant". La Figure 2.13 illustre les deux types d'agglomération ; à savoir, *la Post-agglomération* et *l'agglomération latérale.* La règle de regroupement s'applique alors selon les étapes suivantes :

- Cas 1 (Post-agglomération) : s'il existe une place p dans EP^k et une place p' dans EP^i telles que les conditions suivantes sont vérifiées :

  1. $\bullet p$ est post-agglomérable à $p\bullet$ (cf. Définition d'une Post-agglomération en annexe A),
  2. toute transition de $\bullet p$ est une émission à l'entité EP^i d'un message de type $M_x\&M3^y$ ou $M3^z$, tel que $y \geq 0$, $x \in \{2,3,4,5,6\}$ et $z \geq 1$ et toute transition de $p\bullet$ est une émission à EP^i d'un message de type $M3^z$ tel que $z \geq 1$ (ou inversement),
  3. les transitions de $\bullet p'$ sont les réceptions correspondant aux transitions de $\bullet p$ et les transitions de $p'\bullet$ sont les réceptions qui correspondent aux transitions de $p\bullet$,
  4. $\bullet p'$ est post-agglomérable à $p'\bullet$,

  alors chaque transition de $\bullet p$ est agglomérée successivement à chaque transition de $p\bullet$ et chaque transition de $\bullet p'$ est agglomérée à chaque transition de $p'\bullet$. La Post-agglomération d'une transition $t_1 = s_i(M_x\&M3^y)$ à une transition $t_2 = s_i(M3^z)$ consiste à remplacer les deux transitions par une seule transition $t_{12} = s_i(M_x\&M3^{y+z})$. Cette transition a pour entrées les entrées de t_1 et pour sorties les sorties de t_2.

- Cas 2 (agglomération latérale) : s'il existe un couple de transitions (t_1, t_2) dans EP^k et un couple (t_3, t_4) dans EP^i telles que les conditions suivantes soient vérifiées :

  1. t_1 et t_2 sont latéralement agglomérables (cf. Définition d'une agglomération latérale en annexe A),

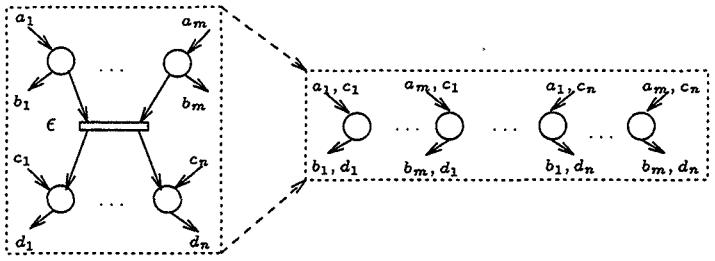

FIG. 2.14 – *Principe de suppression d'une ϵ-transition*

2. t_1 est une émission à l'entité EP^i d'un message de type $M_x\&M3^y$ tel que $y \geq 0$ et $x \in \{2,3,4,5,6\}$, et t_2 est une émission à EP^i d'un message de type $M3^z$ tel que $z \geq 1$ (ou inversement),
3. t_3 et t_4 sont les réceptions dans EP^i correspondant respectivement à t_1 et t_2,
4. t_3 et t_4 sont latéralement agglomérables,

alors les deux couples de transitions (t_1, t_2) et (t_3, t_4) sont agglomérés latéralement. L'agglomération latérale du couple (t_1, t_2) consiste à fusionner les transitions t_1 et t_2 en une seule transition ayant pour entrées l'union des entrées de t_1 et t_2 et pour sorties l'union des sorties de t_1 et t_2. La concaténation des messages se fait de la même manière que dans une Post-agglomération.

2.3.6.3 Elimination des ϵ-transitions (Règle RR3)

Des ϵ-transitions sont introduites dans une entité de protocole dans l'une des 3 situations suivantes :

1. Une entité de protocole qui ne participe pas à l'exécution d'une transition t de $SPEC_s$ aura dans sa spécification une ϵ-transition à la place de t,
2. La substitution (lors de la synthèse) d'un bloc de réseaux de Petri contenant des ϵ-transitions,
3. et suite à l'application de la règle de réduction $RR1$.

De ce fait, une phase de suppression des ϵ-transitions vient compléter la construction des entités de protocole.
L'idée consiste à fusionner les places d'entrée avec les places de sortie d'une ϵ-transition. S'il s'agit d'une transition ayant une place en entrée et une place en sortie, la fusion donne une seule place, mais dans le cas général où on a m places d'entrée et n places de sortie, la fusion donne $m * n$ places ; chaque place de sortie est fusionnée avec toutes les places d'entrée. La fusion de deux places p_1 et p_2 engendre une place p, telle que : $\bullet p = \bullet p_1 \cup \bullet p_2$ et $p\bullet = p_1 \bullet \cup p_2\bullet$ (cf. Figure 2.14) et le marquage de p est la somme des marquages de p_1 et p_2. Afin de préserver les propriétés vivant et borné, nous posons des conditions dans l'application de cette règle. Ces condition ont été introduites dans [KOV 91], elles sont définies comme suit : (a) Le réseau de Petri doit être à choix libre étendu, (b) $\bullet t \neq \emptyset \neq (t\bullet)\bullet$,

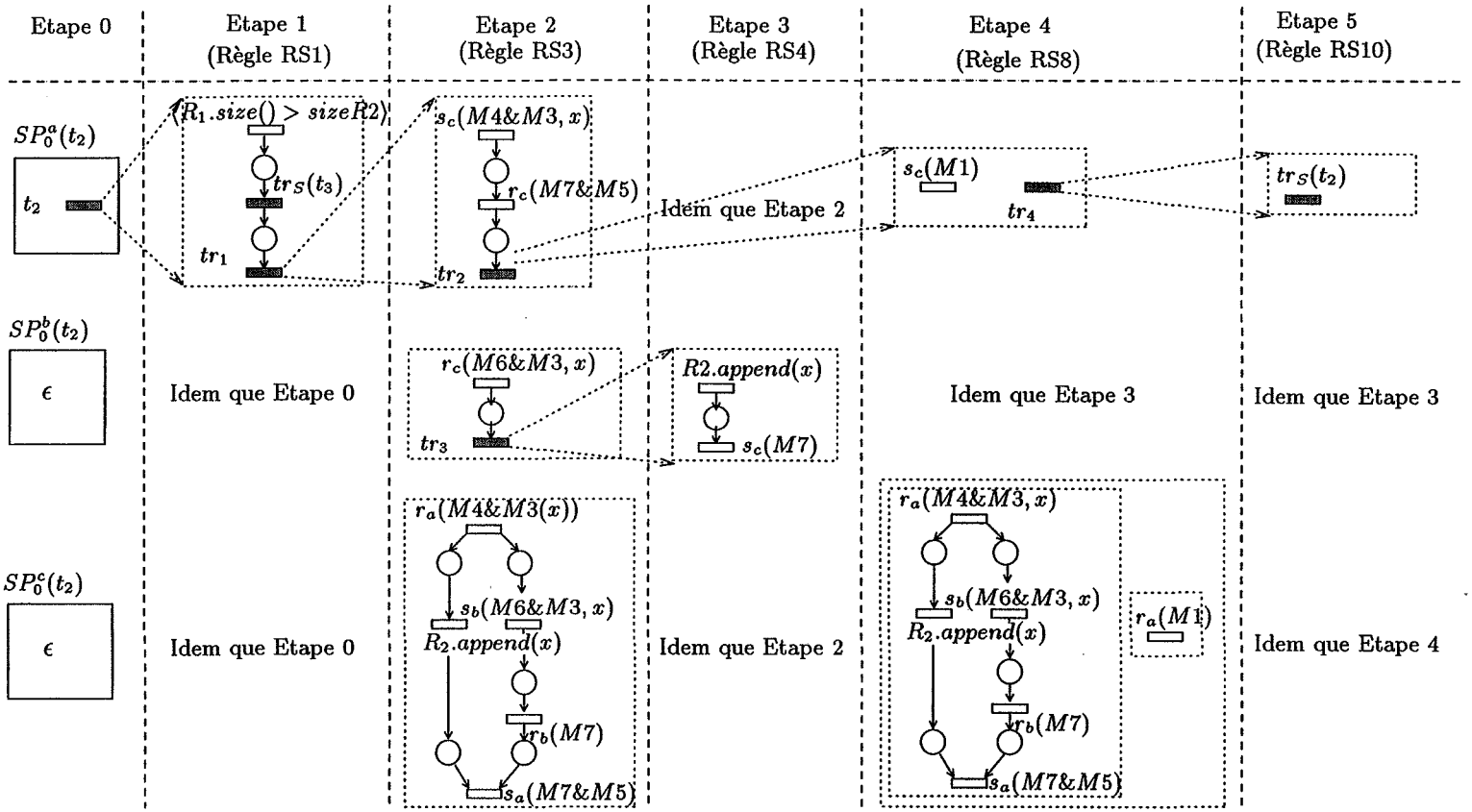

FIG. 2.15 – *Raffinement de la transition de service t_2*

(c) $\forall p \in \bullet t, \{t\} \subseteq p\bullet$, et (d) $(\bullet t)\bullet \cap (t\bullet)\bullet = \emptyset$. Il a été montré dans [KOV 91] que cette règle préserve le caractère borné et vivant. Dans notre cas, le réseau de Petri sous-jacent à la spécification de service ainsi que les blocs de réseaux utilisés par les règles de synthèse font tous partie de la classe des réseaux à choix libre étendus. Il est donc possible d'appliquer cette règle de réduction sans remettre en cause la vivacité et le caractère borné du réseau réduit.

2.3.7 L'algorithme de synthèse

Algorithme de Synthèse *L'algorithme de synthèse consiste en quatre phases:*

- **Phase 0 :**
  - *Construire le graphe des conflits associé à $SPEC_s$.*
  - *Pour chaque transition de service t :*
    - *Construire l'arbre des entrées de t.*
    - *Calculer les attributs SE et TS pour tout noeud de l'arbre des entrées de t.*
- **Phase 1 :** *pour chaque transition t^i et chaque place p de $SPEC_s$:*
  - *Initialement: $SP_0^i(t^i) = tr_0(t^i)$, $\forall k \neq i, SP_0^k(t^i) = \varepsilon$, $\forall k \in \{1...n\}$, $SP_0^k(p) = pr_0(p)$ (si p est une place à choix distribué) et $SP_0^k(p) = p$ (si p n'est pas une place à choix distribué).*
  - *Appliquer les règles contextuelles aux sous-réseaux de Petri $SP_0^k(t^i)$ et $SP_0^k(p)$ ($k = 1$ à n) jusqu'à une étape e où aucune règle ne peut s'appliquer. On obtient l'ensemble suivant:*

$$\{SP_e^1(t^i), ..., SP_e^n(t^i), SP_e^1(p), ..., SP_e^n(p)\}$$

- **Phase 2 :** *Construire les entités de protocole EP^k ($k = 1$ à n) par des T-substitutions et des P-substitutions:*

$$EP^k = \begin{cases} [t^i/SP_e^k(t^i)]SPEC_s & \forall t^i \in SPEC_s \\ [p/SP_e^k(p)]SPEC_s & \forall p \in SPEC_s \end{cases}$$

- **Phase 3 :** *Pour toute entité de protocole EP^k, réduire EP^k en utilisant les règles RR1, RR2 et RR3.* ■

2.3.8 Exemple illustratif

Reprenons l'exemple de la section 2.2.4. Il s'agit dans ce cas de dériver trois entités de protocole EP^1, EP^2 et EP^3 qui sont respectivement associées aux sites S_1, S_2 et S_3. Pour obtenir ces spécifications, on commence par synthétiser chaque transition de service donnée dans la Figure 2.2. Par exemple, la synthèse de la transition t_2 passe par une succession de raffinements comme indiqué par la Figure 2.15. Les spécifications des entités de protocole sont donc construites en remplaçant chaque transition de service par son sous-réseau raffiné comme le montre la Figure 2.16 (pour des raisons de clarté, nous avons omis la représentation des paramètres dans les messages générés).

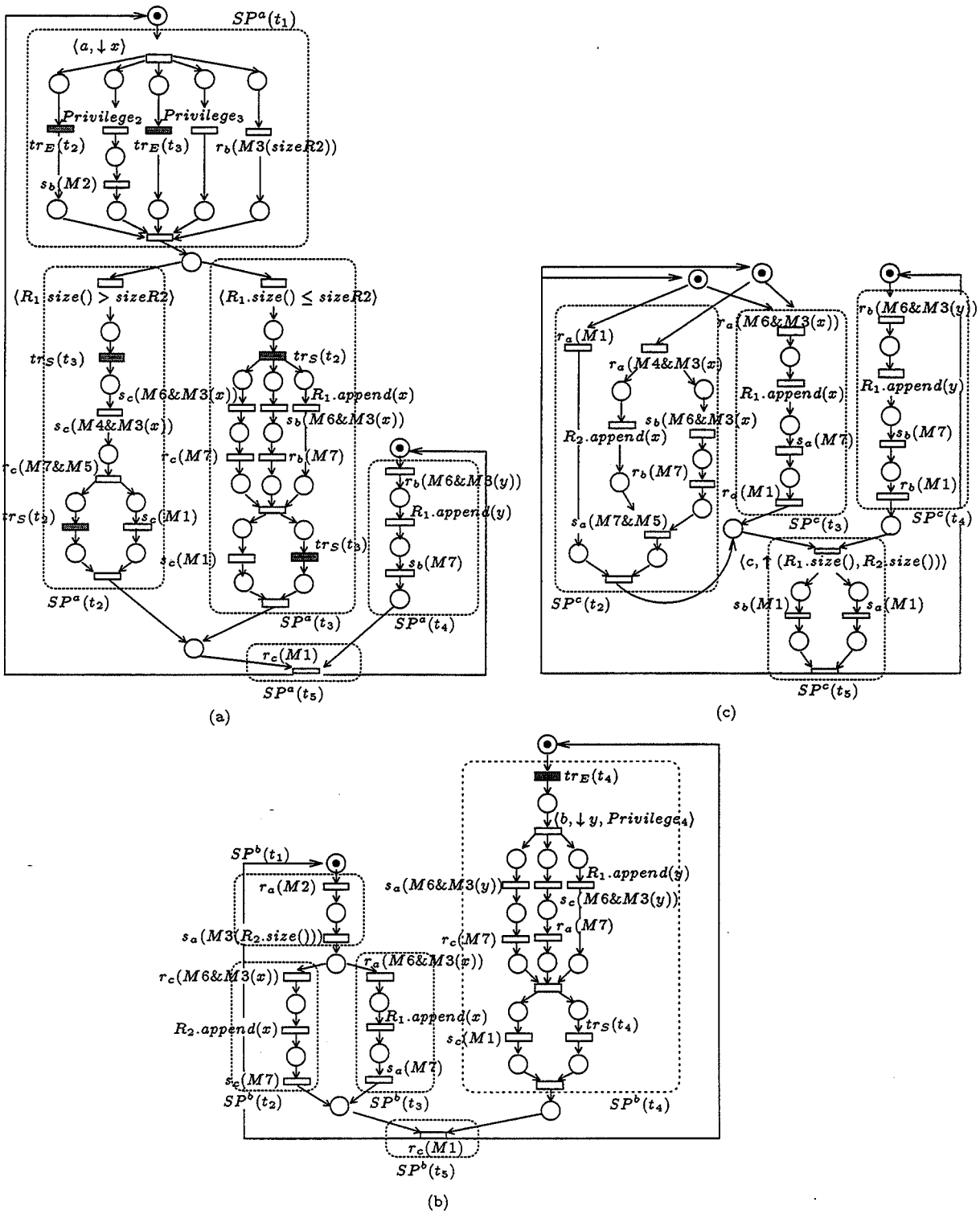

FIG. 2.16 – *Exemple de spécification de protocole (a) Entité de protocole EP^1 (b) Entité de protocole EP^2 (c) Entité de protocole EP^3*

2.4 Évaluation de la stratégie de synthèse

2.4.1 Correction de l'algorithme de synthèse

Pour montrer que l'algorithme de synthèse est correct, il suffit de montrer que les protocoles dérivés sont corrects logiquement et sémantiquement.

2.4.1.1 Correction logique

Dans la preuve de correction logique, nous ne considérons que la partie contrôle des spécifications, à savoir le réseau de Petri sous-jacent. Rappelons d'abord, les hypothèses sur l'environnement :

- Il existe un lien de communication bi-directionnel, FIFO et fiable entre toute paire d'entités de protocole.
- A chaque SAP est associée une file FIFO pour la réception des événements de la part d'un utilisateur
- A chaque utilisateur attaché à un SAP est associée une file FIFO pour la réception des événements de la part d'une entité de protocole.

Une émission de message n'est pas bloquante. De plus, il n'est pas possible d'émettre indéfiniment à cause de la propriété "borné". D'autre part, quand il y a une émission dans une entité, il est certain que le récepteur est dans l'état de recevoir, ceci est garanti par la stratégie de synthèse du flot de contrôle. Par ailleurs, l'hypothèse de fiabilité garantit que les messages émis sont nécessairement reçus, et l'hypothèse FIFO assure que l'ordre des réceptions est conforme à l'ordre des émissions et donc à l'ordre défini dans le flot de contrôle.
Par conséquent, il ne peut pas y avoir d'interblocage causé par l'environnement. C'est pour cela que la preuve de correction considère le réseau de Petri sans la partie environnement. L'hypothèse de liens FIFO et fiables est discutée encore à la fin du chapitre 4.

La correction logique consiste à montrer que les spécifications des entités de protocole construites par l'algorithme de synthèse sont vivantes et bornées. La spécification d'une entité de protocole est obtenue en substituant aux places et aux transitions de la spécification de service des sous-réseaux de Petri construits au moyen des règles contextuelles. La spécification obtenue est ensuite réduite par les règles RR1, RR2 et RR3. Pour montrer la correction logique, nous allons montrer que les transformations que nous faisons subir à une spécification de service pour construire une entité de protocole maintiennent les propriétés borné et vivant. Ce résultat est utilisé ensuite pour énoncer le théorème de la correction logique.

Proposition 1 (Substitution de transition) *Si une spécification de service N est vivante et bornée alors la spécification obtenue à partir de N par T-substitution d'un sous-réseau de Petri construit au moyen des règles RS1 à RS10 est aussi vivante et bornée.*

Preuve Nous utilisons un résultat dû à SUZUKI et MURATA [SUZ 83]. Soient N et N' deux réseaux de Petri différents, t_0 une transition de N, et t_{in}, t_{out} deux transitions de N'. Soit

$B(N')$ le réseau de Petri obtenu en ajoutant une place marquée p_0 à N', telle que p_0 soit en entrée de t_{in} et en sortie de t_{out}. Les transitions t_{in} et t_{out} doivent vérifier trois conditions : (C1) t_{in} doit être vivante dans $B(N')$, (C2) le nombre de tirs de t_{out} ne peut jamais dépasser celui de t_{in}, et (C3) dans le cas où C2 n'est pas vérifiée, il est toujours possible de trouver une séquence de franchissements qui égalise le nombre de tirs des deux transitions. Soit N'' le réseau de Petri obtenu à partir de N en remplaçant t_0 par N' ($\bullet t_{in} = \bullet t_0$ et $t_{out}\bullet = t_0\bullet$). Il a été montré dans [SUZ 83] que si N est m-borné et vivant et $B(N')$ est m'-borné et vivant alors N'' est m''-borné et vivant, où $m'' = Max(m, m')$.

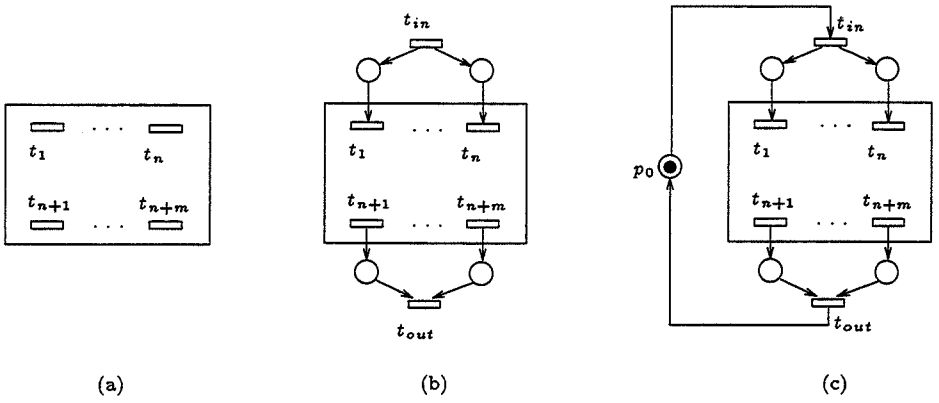

FIG. 2.17 – *Substitution de transition (a) $SP(t)$ (b) $SP(t)$ mis en forme (c) $B(SP(t))$*

Soit un sous-réseau de Petri $SP(t)$ construit par les règles de synthèse contextuelles RS1 à RS10 à partir d'une transition t. Le sous-réseau $SP(t)$ ainsi que tous les blocs de réseaux de Petri substitués par les règles RS1 à RS10 possèdent, par construction, un ensemble de transitions de début T_{in} (transitions sans places d'entrée) et un ensemble de transitions finale T_{fin} (transitions sans places de sortie). Avant d'être substitué, un bloc est mis en forme en lui ajoutant deux ϵ-transitions t_{in} et t_{out} (cf. Figure 2.17). Cette mise en forme est nécessaire seulement dans le cas où le bloc contient plus d'une transition initiale ou finale.

Montrons, en utilisant le résultat ci-dessus par induction sur l'application des règles contextuelles, que le réseau $B(SP(t))$ est borné et vivant. Initialement, $SP_0(t) = tr_0$ dans le site d'allocation de t et $SP_0(t) = \epsilon$ dans les autres sites. Dans les deux cas, $B(SP_0(t))$ est borné et vivant. On suppose qu'à une étape e de la synthèse $B(SP_e(t))$ est borné et vivant. Deux cas sont possibles pour construire $SP_{e+1}(t)$:
Cas 1 : $SP_{e+1}(t) = [tr/BP]SP_e(t)$ (où tr est une transition de $SP_e(t)$ et BP un bloc de réseaux de Petri). Il vient : $B(SP_{e+1}(t)) = [tr/BP]B(SP_e(t))$. Or $B(BP)$ est borné et vivant par construction, donc on déduit par utilisation du résultat ci-dessus que $B(SP_{e+1}(t))$ est borné et vivant. Pour ne pas changer la sémantique du réseau ayant subi la substitution, les ϵ-transitions t_{in} et t_{out} doivent être supprimées. Sachant, par construction, que $B(SP_{e+1}(t))$ fait partie de la classe des réseaux à choix libre étendus, nous pouvons déduire que la suppression de ces transitions par la règle RR3 préserve les propriétés borné et vivant.

Cas 2 : $SP_{e+1}(t) = SP_e(t) + BP$. L'Addition s'effectue ainsi : Nous ajoutons une transition t_i sans place d'entrée ni de sortie dans $SP_e(t)$. Nous obtenons $SP'_e(t)$. La transition t_i est à la fois initiale et finale dans $SP'_e(t)$. Puisque $B(SP_e(t))$ est borné et vivant, alors il est facile de montrer que $B(SP'_e(t))$ est borné et vivant. Pour obtenir $B(SP_{e+1}(t))$, nous substituons BP à t_i dans $B(SP'_e(t))$. Le bloc $B(BP)$ étant vivant et borné par construction, nous déduisons, par utilisation du résultat ci-dessus, que $B(SP_{e+1}(t))$ est vivant et borné. En conclusion du raisonnement par induction, il vient que $B(SP(t))$ est borné et vivant. De la même manière que précédemment, nous supprimons les ϵ-transitions ayant servi dans la substitution.

Nous déduisons (en utilisant toujours le résultat ci-dessus) que la spécification obtenue par substitution de $SP(t)$ à la transition t_0 dans N est bornée et vivante.

Après substitution de $SP(t)$, nous supprimons encore une fois, au moyen de la règle de réduction RR3, les ϵ-transitions ajoutées t_{in} et t_{out}. Le réseau sous-jacent à la spécification étant à choix libre, la suppression de ces transitions à l'aide de la règle RR3 préserve la vivacité et le caractère borné. D'où le résultat. ■

Proposition 2 (Substitution de place) *Si une spécification de service N est vivante et bornée alors la spécification obtenue à partir de N par P-substitution d'un sous-réseau de Petri construit par les règles RS11 à RS14 est aussi vivante et bornée.*

Preuve Soit un réseau de Petri N (cf. Figure 2.18a) et un réseau N' (cf. Figure 2.18b). Le réseau N' représente la forme générale d'un bloc construit par les règles RS11 à RS14. La place p_0 de N' est une place initiale (i.e., marquée à l'état initial), alors que les places p_1, ..., p_n sont des places finales (i.e., marquées à l'état final). Le réseau N' possède la propriété comportementale suivante : à partir de l'état initial où p_0 est marquée, il atteint inévitablement un état final où seulement une des places p_1 à p_n est marquée. Il est facile de voir que $B(N')$ est borné et vivant pour tout N' construit par les règles RS11 à RS14. Nous nous proposons de montrer dans ce qui suit, le résultat suivant : si N et $B(N')$ sont bornés et vivants alors $N'' = [p/N']N$ est borné et vivant.
Supposons que N'' est non borné et que N et $B(N')$ sont bornés, alors il existe une séquence répétitive croissante σ'' pour une place p_a dans N''. Puisque les transitions de la séquence σ'' qui sont dans N' ne font changer que le marquage des places de N', et que les transitions qui sont dans N ne font changer que le marquage des places de N et de $\{p_o, ..., p_n\}$, il est alors facile de voir que la projection de σ'' sur les transitions de N (on note $\Pi_N(\sigma'')$) est une séquence de N et que la projection $\Pi_{N'}(\sigma'')$ est une séquence de N'. On déduit alors que si $p_a \in N$ alors $\Pi_N(\sigma'')$ augmente le marquage de p_a dans N et que si $p_a \in N'$ alors $\Pi_{N'}(\sigma'')$ augmente le marquage de p_a dans N'. Or, on sait que N et $B(N')$ sont bornés d'où la contradiction. Par conséquent N'' est borné.

Supposons que N et $B(N')$ sont vivants et montrons que N'' est vivant. Soit $\sigma''_1 \in N''$ telle que $M''_0[\sigma''_1\rangle M''_1$ (i.e., à partir du marquage initial M''_0 de N'', le franchissement de σ''_1 conduit au marquage M''_1). Alors on déduit $M_0[\sigma_1\rangle M_1$ telle que : $\sigma_1 = \Pi_N(\sigma''_1)$ et M_0, M_1 sont des marquages de N. Soit une transition $t \in N''$:
Cas 1 ($t \in N$) : Puisque N est vivant alors il existe σ_2 telle que $M_1[\sigma_2 t\rangle$ (i.e., la séquence σ_2

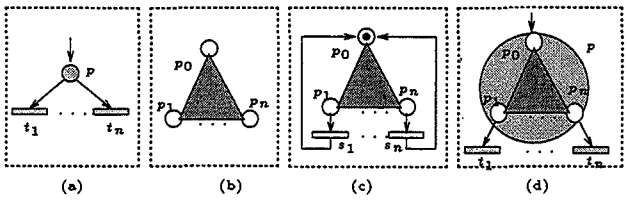

FIG. 2.18 – *Substitution de place (a) N (b) N' (c) $B(N')$ (d) $N'' = [p/N']N$*

est franchissable au marquage M_1). Or à partir d'une séquence σ_2 de N on peut toujours trouver une séquence σ_2'' de N''. En effet, si σ_2 ne possède aucune transition ayant p_0 comme place de sortie alors σ_2 est aussi une séquence de N'', sinon pour construire σ_2'', on insère une sous-séquence de N' à chaque occurrence d'une transition ayant p_0 comme place de sortie. On déduit, $M_1''[\sigma_2''t\rangle$.
Cas 2 ($t \in N'$) : Puisque N est vivant alors on peut trouver une séquence σ_2 et une transition t_1 ayant p_0 comme place de sortie de telle sorte que $M_1(\sigma_2 t_1\rangle$. Par ailleurs, on sait que $B(N')$ est vivant alors à partir du moment où p_0 est marquée on peut trouver une séquence σ_2' dans N' qui sensibilise la transition t. Donc on déduit : $M_1''(\sigma_2 t_1 \sigma_2' t\rangle$, d'où le résultat. ∎

Proposition 3 (Réduction) *Si une spécification IPN est vivante et bornée alors la spécification obtenue après application d'une règle de réduction est aussi vivante et bornée.*

Preuve La règle RR1 n'effectue aucun changement dans la structure du réseau de Petri, elle maintient donc les propriétés borné et vivant.
La règle RR2 consiste en une agglomération de transitions. D'après [BER 86], cette transformation préserve les propriétés vivant et borné. La règle RR3 préserve également le caractère borné et vivant, ce résultat est montré dans [KOV 91]. ∎

Théorème 1 (Correction logique) *Si une spécification de service est vivante et bornée alors les spécifications des entités de protocoles construites par l'algorithme de synthèse sont vivantes et bornées.*

Preuve La construction d'une entité de protocole démarre d'une spécification de service vivante et bornée. D'après les Propositions 1 et 2, les transformations effectuées sur la spécification de service dans la phase 2 de l'algorithme de synthèse préservent les propriétés vivant et borné. De plus, d'après la Proposition 3, ces propriétés sont maintenues aussi dans la phase 3 de l'algorithme. D'où le résultat. ∎

2.4.1.2 Correction sémantique

La correction sémantique consiste à prouver que la spécification d'un protocole construite par l'algorithme de synthèse est conforme à la spécification du service qu'il doit fournir. Il s'agit donc de comparer les deux spécifications. La comparaison des spécifications de deux systèmes est basée, en général, sur la notion d'équivalence. Pour comparer la spécification d'un service à celle du protocole qui lui correspond, nous considérons le service et le protocole comme étant deux boites noires observables de l'extérieur. Les activités observables sont les

primitives de service définies dans $\mathcal{P}$. Le couple (service, protocole) est ainsi assimilé à un couple de processus du style CSP [HOA 85]. La comparaison porte alors sur les traces de ces deux processus. Nous avons donc choisi *l'equivalence de trace* pour comparer un service au protocole qui lui correspond. Ce type d'équivalence nous semble suffisant. En effet, le résultat que nous voulons atteindre est que *toutes les séquences d'exécution valides de primitives de service soient des séquences exécutables par le protocole. Inversement, toutes les séquences d'exécution observables au niveau protocole doivent être des séquences valides.* Les séquences valides sont celles définies par la spécification de service.
D'après [MIL 89], le principal désavantage de l'équivalence de trace est qu'elle n'indique rien sur les interblocages. En d'autres termes, elle peut qualifier d'équivalents un système sans interblocage et un système conduisant à une situation d'interblocage. Néanmoins, ce problème ne se pose pas dans notre cas. En effet, d'une part, les spécifications de service sont supposées correctes et donc sans interblocage. D'autre part, la correction logique de notre algorithme de synthèse garantit que les protocoles dérivés sont logiquement corrects (donc sans interblocage).
Par ailleurs, l'équivalence de trace est une congruence, elle est préservée par tous les combinateurs de base. En d'autres termes, l'équivalence de trace est préservée dans tous les contextes où peuvent se trouver les systèmes comparés. De ce fait, les protocoles synthétisés seraient corrects indépendamment du contexte des processus qui utilisent le service fourni.

Nous considérons deux processus $SPEC_s$ et $SPEC_p$, une trace d'un processus est définie comme étant une séquence finie d'activités que le processus peut réaliser. Si α est une trace de $SPEC_s$ alors $\alpha \in \mathcal{P}^*$ (où $\mathcal{P}^*$ désigne tous les mots qu'on peut construire à l'aide des symboles de l'ensemble $\mathcal{P}$). De plus, α s'écrit $\mathcal{X}(t_1)\mathcal{X}(t_2)...\mathcal{X}(t_n)$, de telle sorte que $t_1t_2...t_n$ soit une séquence de transitions franchissable dans $SPEC_s$. Si α est une trace de $SPEC_p$ alors $\alpha \in (\mathcal{P} \cup IPS \cup \{\epsilon\})^*$. Dans ce cas, α s'écrit $\mathcal{X}_{p_1}(t_1)\mathcal{X}_{p_2}(t_2)...\mathcal{X}_{p_n}(t_n)$, où $\mathcal{X}_{p_i}$ est la fonction d'étiquetage associée à l'une des entités de protocole de $SPEC_p$ et t_1 à t_n sont des transitions de $SPEC_p$. De plus, la séquence $t_1t_2...t_n$ est franchissable dans $SPEC_p$. A partir d'une trace, on peut construire une nouvelle trace en appliquant un opérateur de restriction noté Π. Ainsi, l'expression $\Pi_A(\alpha)$ dénote la trace α réduite aux symboles de l'ensemble A (ou encore la projection de α sur les symboles de A).

Proposition 4 *Étant donnée une transition t de $SPEC_s$, le raffinement de cette transition par l'ensemble des règles de synthèse RS1 à RS10 produit un réseau de Petri R_t tel que :*

$$\Pi_{\mathcal{P}}(traces(R_t)) = \{\mathcal{X}(t)\}$$

Où "traces" est un opérateur qui renvoie l'ensemble de toutes les traces possibles d'un processus.

Preuve Soit une transition t de $SPEC_s$ telle que $\mathcal{X}(t) = \langle sap, in, cond, act, out\rangle$. On suppose que $Alloc(t) = S_1$ et que le nombre de sites de l'architecture cible est arbitraire. Nous allons commencer par écrire une grammaire $\mathcal{G}$ qui engendre l'ensemble des traces d'un réseau de Petri obtenu par utilisation des règles RS1 à RS10. La forme générale d'une règle de cette grammaire est :

$$tr_i \longrightarrow \Sigma_1, ..., \Sigma_n$$

tr_i est un symbole non terminal, et Σ_i est un ensemble de traces correspondant à un site donné. En d'autres termes, à partir de tr_i, on peut produire n ensembles de traces chacun associé à un site. Cette grammaire est construite à partir des règles contextuelles. En effet, chaque méta-transition tr_i devient un non terminal dans la grammaire $\mathcal{G}$ et chaque bloc de réseaux de Petri associé à un site donné est remplacé par l'ensemble des traces qu'il engendre. Les terminaux de cette grammaire appartiennent à l'ensemble des primitives de service $\mathcal{P} \cup IPS \cup \{\epsilon\}$. Si α_1 et α_2 sont deux traces, alors $\alpha_1.\alpha_2$ désigne une trace obtenue par concaténation des deux traces. Si X et Y sont deux ensembles de traces alors $X \parallel Y$ définit tous les entrelacements pouvant être engendrés par la mise en parallèle de deux processus ayant respectivement pour traces X et Y. De plus, un opérateur noté $\parallel^*$ est utilisé afin simplifier l'écriture des règles. L'expression $\parallel^* A$ signifie tous les entrelacements pouvant être engendrés par la mise en parallèle de zéro, un ou plusieurs processus ayant pour traces A. Les symboles "|" et "[]" sont utilisés dans la grammaire pour exprimer respectivement un choix et une expression optionnelle.

$$
\begin{array}{lll}
tr_0 & \longrightarrow & \langle sap, in, cond\rangle.(\parallel^* s(M9)).tr_1 \\
 & | & (\parallel^* s(M9)).tr_1 \\
 & | & ((\langle sap, in\rangle \parallel Privilege).tr_1) \parallel s(m8) \\
tr_1 & \longrightarrow & (\parallel^* s(M2\&M3^*)) \parallel ((\parallel^* r(M3)).\langle sap, act\rangle.(\parallel^* (s(M6\&M3^*).r(M7))).tr_2), \\
 & & [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], ..., [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], \\
 & & r(M6\&M3^*).tr_3, ..., r(M6\&M3^*).tr_3 \\
 & | & (\parallel^* s(M2\&M3^*)).s_u(M4\&M3^*).r_u(M7\&M5\&M3^*).tr_2, \\
 & & (\parallel^* r(M3)).r_i(M4\&M3^*).\langle sap, act\rangle.(\parallel^* (s(M6\&M3^*).r(M7))).s_i(M7\&M5\&M3^*), \\
 & & [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], ..., [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], \\
 & & r(M6\&M3^*).tr_3, ..., r(M6\&M3^*).tr_3 \\
tr_2 & \longrightarrow & (\parallel^* s(M2\&M3^*)) \parallel ((\parallel^* r(M3)).\langle sap, out\rangle.tr_4), \\
 & & [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], ..., [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)] \\
tr_3 & \longrightarrow & \langle act\rangle.s(M7) \\
tr_4 & \longrightarrow & (\parallel^* s(M3)) \parallel tr_5, r(M3), ..., r(M3) \\
tr_5 & \longrightarrow & (\parallel^* s(M2\&M3^*)) \parallel tr_6 \parallel (\parallel^* s(M3)), \\
 & & [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], ..., [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], \\
 & & \parallel^* r(M3), ..., \parallel^* r(M3) \\
tr_6 & \longrightarrow & tr_7 \parallel (\parallel^* s(M1)) \parallel (\parallel^* tr_8), r(M1).tr_8, ..., r(M1).tr_8 \\
tr_8 & \longrightarrow & \parallel^* (s(M8) \parallel (Privilege.(\parallel^* s(M2\&M3^*))) \parallel (\parallel^* r(M3))), \\
 & & [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)], ..., [(r(M2\&M3^*) \parallel (\parallel^* r(M3))).s(M3)] \\
tr_7 & \longrightarrow & s(M9)
\end{array}
$$

Pour calculer l'ensemble des traces (restreintes aux symboles de $\mathcal{P}^*$) engendré par la grammaire $\mathcal{G}$, nous remplaçons certains symboles d'émission et de réception de messages par ϵ. Nous obtenons la grammaire $\mathcal{G}'$ suivante:

$$
\begin{array}{lll}
tr_0 & \longrightarrow & \langle sap, in, cond\rangle.tr_1 \\
 & | & tr_1 \\
 & | & \langle sap, in\rangle.tr_1 \\
tr_1 & \longrightarrow & \langle sap, act\rangle.(\parallel^* (s(M6\&M3^*).r(M7))).tr_2, \\
 & & r(M6\&M3^*).tr_3, ..., r(M6\&M3^*).tr_3 \\
 & | & s_u(M4\&M3^*).r_u(M7\&M5\&M3^*).tr_2, \\
 & & r_i(M4\&M3^*).\langle sap, act\rangle.(\parallel^* (s(M6\&M3^*).r(M7))).s_i(M7\&M5\&M3^*), \\
 & & r(M6\&M3^*).tr_3, ..., r(M6\&M3^*).tr_3
\end{array}
$$

$$
\begin{array}{lcl}
tr_2 & \longrightarrow & \langle sap, out\rangle.tr_4, \\
tr_3 & \longrightarrow & \langle act\rangle.s(M7) \\
tr_4 & \longrightarrow & tr_5 \\
tr_5 & \longrightarrow & tr_6 \\
tr_6 & \longrightarrow & tr_7 \parallel (\|^* tr_8), tr_8, ..., tr_8 \\
tr_8 & \longrightarrow & \epsilon \\
tr_7 & \longrightarrow & \epsilon
\end{array}
$$

En remplaçant encore les émissions et les réceptions par ϵ, la grammaire $\mathcal{G}'$ devient :

$$
\begin{array}{lcl}
tr_0 & \longrightarrow & \langle sap, in, cond\rangle.\langle sap, act\rangle.\langle sap, out\rangle, \langle act\rangle, ..., \langle act\rangle \\
 & | & \langle sap, act\rangle.\langle sap, out\rangle, \langle act\rangle, ..., \langle act\rangle \\
 & | & \langle sap, in\rangle.\langle sap, act\rangle, \langle act\rangle, ..., \langle act\rangle
\end{array}
$$

La grammaire $\mathcal{G}'$ peut encore s'écrire :

$$
\begin{array}{lcl}
tr_0 & \longrightarrow & \langle sap, in, cond, act, out\rangle, \langle act\rangle, ..., \langle act\rangle \\
 & | & \langle sap, act, out\rangle, \langle act\rangle, ..., \langle act\rangle \\
 & | & \langle sap, in, act\rangle, \langle act\rangle, ..., \langle act\rangle
\end{array}
$$

Dans $SPEC_s$, les ressources étant supposées globales, la partie "action" ne s'exécute qu'une seule fois, d'où nous déduisons que la grammaire $\mathcal{G}'$ engendre bien la transition t, par conséquent il en est de même pour la grammaire $\mathcal{G}$, et donc pour les règles RS1 à RS10. ■

Proposition 5 *L'ordre d'exécution des sous-réseaux de Petri dans chaque entité de protocole est identique à celui des transitions correspondantes dans la spécification de service.*

Preuve Soient t une transition de $SPEC_s$ et $\{SP^1(t), ..., SP^n(t)\}$ l'ensemble des sous-réseaux de Petri obtenus après application des règles de synthèse. Soit EP^i une entité de protocole donnée, d'après l'algorithme de synthèse, EP^i est obtenue en remplaçant dans $SPEC_s$ chaque transition t par $SP^i(t)$. Si dans une trace de $SPEC_s$ la transition t précède une transition t', alors après substitution de $SP^i(t)$ à t et $SP(t')$ à t', dans toute séquence de EP^i les transitions de $SP^i(t)$ précèdent celles de $SP(t')$.
Inversement, si une transition t_i précède t'_i dans EP^i alors deux cas sont possibles : (1) t_i et t'_i appartiennent à un même sous-réseau de Petri, auquel cas ces transitions sont relatifs à une même transition de $SPEC_s$, (2) t_i et t'_i appartiennent à deux sous-réseaux différents ; respectivement $SP^i(t)$ et $SP(t')$. Dans ce cas, $SP^i(t)$ précède forcément $SP(t')$ et par conséquent t précède t' dans $SPEC_s$. ■

Proposition 6 *Soit p une place à choix distribué dans $SPEC_s$, les sous-réseaux de Petri construits à partir de p par les règles RS11 à RS14 garantissent qu'exactement une seule transition sortante de p est exécutée par les entités de protocole.*

Preuve
Soit p une place à choix distribué de $SPEC_s$ et $SP(p)$ l'ensemble des sous-réseaux de Petri construits par les règles RS11 à RS14, les sous-réseaux de $SP(p)$ possèdent la forme donnée par le réseau N' (cf. Figure 2.18b). La place p_0 de N' est une place initiale (i.e., marquée à l'état initial), alors que les places p_1, ..., p_n sont des places finales (i.e., marquées à l'état final). Chaque sous-réseau de $SP(p)$ possède la propriété comportementale suivante : à partir

de l'état initial où p_0 est marquée, il atteint inévitablement un état final où seulement une des places p_1 à p_n est marquée. De plus, les états finaux atteints par les sous-réseaux de $SP(p)$ correspondent aux états initiaux des sous-réseaux associés à une seule transition sortante de p. Cette propriété est garantie par le mécanisme d'élection synthétisé au moyen des règles RS11 à RS14. ∎

Corollaire 1 *Soient $SPEC_s$ une spécification de service vérifiant les restrictions R1 à R4 et $SPEC_p$ la spécification du protocole produite par les phases 1 et 2 de l'algorithme de synthèse à partir de $SPEC_s$, alors :*

$$traces(SPEC_s) = \Pi_{\mathcal{P}}(traces(SPEC_p))$$

Preuve Pour montrer ce corollaire, nous allons montrer les deux sens de l'inclusion des deux ensembles $traces(SPEC_s)$ et $\Pi_{\mathcal{P}}(traces(SPEC_p))$.
Soit $\alpha = \mathcal{X}(t_1)\mathcal{X}(t_2)...\mathcal{X}(t_n)$ une trace de $SPEC_s$, d'après la Proposition 4, pour chaque transition t_i de α nous avons : $\Pi_{\mathcal{P}}(traces(SP(t_i)) = \{\mathcal{X}(t_i)\}$. Donc, il existe une trace $\alpha_i \in traces(SP(t_i))$ telle que $\Pi_{\mathcal{P}}(\alpha_i) = \mathcal{X}(t_i)$. Et si l'on suppose que t_i appartient à une structure sélective distribuée alors d'après la Proposition 3, il existe une trace $\gamma_i \in traces(SP(p_i))$ telle que : $\Pi_{\mathcal{P}}(\gamma_i\alpha_i) = \mathcal{X}(t_i)$ et γ_i permet de sélectionner de façon unique la transition t_i. De plus, d'après la Proposition 2, si t_i précède t_j $(j \neq i)$ dans α, alors α_i précède forcément α_j dans $SPEC_p$. Par conséquent, nous pouvons construire à partir de α une trace α' de $SPEC_p$, $\alpha' = \alpha_1...\gamma_i\alpha_i...\alpha_n$. La projection de α' sur $\mathcal{P}$ donne évidemment la trace α.
Soit α' une trace de $SPEC_p$, on suppose que $SPEC_s$ possède une seule place p à choix distribué. D'après les Propositions 4, 5 et 6, il existe $t_1, ..., t_n$ des transitions de $SPEC_s$ (où seulement une transition $t_i \in {}^\bullet p$) et $\alpha_1, ..., \alpha_n, \gamma_i$ des traces de $SPEC_p$ telles que: $\alpha' = \alpha_1...\gamma_i\alpha_i...\alpha_n$ avec $\alpha_k \in traces(SP(t_k))$ (k allant de 1 à n) et $\gamma_i \in traces(SP(p))$. La projection de α' sur $\mathcal{P}$ donne $\alpha = \mathcal{X}(t_1)...\mathcal{X}(t_n)$ qui est une trace de $SPEC_s$. D'où le résultat. ∎

Proposition 7 *Les règles de réduction RR1 à RR3 maintiennent l'équivalence de trace.*

Preuve Nous allons considérer trois cas. Chacun concerne la preuve du maintien de l'équivalence par rapport à une règle donnée. Soient $SPEC_p$ une spécification de protocole construite par les phases 1 et 2 de l'algorithme de synthèse à partir de $SPEC_s$, et $SPEC'_p$ la spécification du protocole obtenue après application d'une règle de réduction, nous devons prouver que $SPEC'_p$ est trace-équivalente à $SPEC_s$. Dans chaque cas, nous montrons les deux sens de l'inclusion des deux ensemble $traces(SPEC_s)$ et $\Pi_{\mathcal{P}}(traces(SPEC'_p))$.
Cas 1 (Règle RR1) : Soit α une trace de $SPEC_p$ qui contient une ϵ-transition $\alpha = \beta\epsilon\gamma$. Après application de RR1, $\alpha' = \beta\gamma$ est une trace de $SPEC'_p$. Or d'après le Corollaire 1, nous savons qu'il existe une trace α'' dans $SPEC_s$ telle que $\Pi_{\mathcal{P}}(\alpha) = \alpha''$. Puisque $\Pi_{\mathcal{P}}(\alpha) = \Pi_{\mathcal{P}}(\alpha')$, il vient : $\Pi_{\mathcal{P}}(\alpha') = \alpha''$.
Soit α une trace de $SPEC_s$, alors d'après le Corollaire 1, il existe α' dans $SPEC_p$ telle que $\Pi_{\mathcal{P}}(\alpha') = \alpha$. Si α' ne contient pas des ϵ-transitions alors α' est aussi une trace de $SPEC'_p$. Par contre, si α' contient une ϵ-transition que l'on souhaite supprimer à l'aide de RR1, α' s'écrira $\beta\epsilon\gamma$. Après application de RR1, la trace $\alpha'' = \beta\gamma$ est une trace de $SPEC'_p$ et $\Pi_{\mathcal{P}}(\alpha'') = \alpha$.
Cas 2 (Règle RR2) : Soit α une trace de $SPEC_p$ qui contient la transition de réception que

l'on souhaite supprimer par la règle $RR2$. Donc α est de la forme $\beta\mathcal{X}_p(t_e)\gamma\mathcal{X}_p(t_r)\delta$ où t_r est la transition de réception, t_e est la transition d'émission qui lui correspond et β, γ, δ sont des traces de $SPEC_p$. D'après le Corollaire 1, nous savons que $\Pi_{\mathcal{P}}(\alpha)$ est une trace de $SPEC_s$. Après application de $RR2$, $\alpha' = \beta\gamma\delta$ est une trace de $SPEC'_p$. Puisque $\Pi_{\mathcal{P}}(\alpha') = \Pi_{\mathcal{P}}(\alpha)$ alors $\Pi_{\mathcal{P}}(\alpha')$ est une trace de $SPEC_s$.
Soit α une trace de $SPEC_s$. D'après le Corollaire 1, il existe une trace α' de $SPEC_p$ telle que $\Pi_{\mathcal{P}}(\alpha') = \alpha$. Deux cas sont possibles : soit la trace α' contient la transition de réception t_e que l'on souhaite supprimer par RR2, soit elle ne contient pas cette transition. Dans le premier cas, $\alpha' = \beta\mathcal{X}_p(t_e)\gamma\mathcal{X}_p(t_r)\delta$. Après application de RR2, la trace $\alpha'' = \beta\gamma\delta$ est une trace de $SPEC'_p$ et $\Pi_{\mathcal{P}}(\alpha'') = \alpha$ est une trace de $SPEC_s$. Dans le second cas α' est une trace de $SPEC'_p$.
Cas 3 (Règle RR3) : Soit α une trace de $SPEC_p$ qui contient deux transitions t_1 et t_2 post-agglomérables, $\alpha = \beta\mathcal{X}_p(t_1)\mathcal{X}_p(t_2)\gamma$. Après application de la post-agglomération de t_1 et t_2, $\alpha' = \beta\mathcal{X}_p(t_{12})\gamma$ est une trace de $SPEC'_p$ (où t_{12} est une transition obtenue par post-agglomération de t_1 et t_2). Or d'après le Corollaire 1, il existe une trace α'' dans $SPEC_s$ telle que $\Pi_{\mathcal{P}}(\alpha) = \alpha''$. Nous déduisons alors: $\Pi_{\mathcal{P}}(\alpha') = \Pi_{\mathcal{P}}(\alpha)$, et donc $\Pi_{\mathcal{P}}(\alpha') = \alpha''$.
Soit α une trace de $SPEC_s$. D'après le Corollaire 1, il existe α' dans $SPEC_p$ telle que $\Pi_{\mathcal{P}}(\alpha') = \alpha$. Si α' ne contient pas de transitions post-agglomérables alors α' est aussi une trace de $SPEC'_p$, sinon α' aurait la forme suivante: $\beta\mathcal{X}_p(t_1)\mathcal{X}_p(t_2)\gamma$. Après post-agglomération de t_1 et t_2, $\alpha'' = \beta\mathcal{X}_p(t_{12})\gamma$ est une trace de $SPEC'_p$ et $\Pi_{\mathcal{P}}(\alpha'') = \alpha$.
Le raisonnement est le même pour la règle RR3 utilisant une agglomération latérale. ■

Théorème 2 (Correction Sémantique) *Soient $SPEC_s$ une spécification de service vérifiant les restrictions R1 à R4 et $SPEC_p$ la spécification du protocole produite par l'algorithme de synthèse à partir de $SPEC_s$ et de la description de l'architecture cible, alors :*

$$traces(SPEC_s) = \Pi_{\mathcal{P}}(traces(SPEC_p))$$

Preuve Directe à partir du Corollaire 1 et la Proposition 4. ■

2.4.2 Complexité de l'algorithme de synthèse

Pour évaluer la complexité en temps de l'algorithme de synthèse, nous calculons dans ce qui suit son expression à chaque phase de l'algorithme.

2.4.2.1 Complexité de la Phase 0

Nous avons montré dans la section 2.3.2 que la construction du graphe des conflits se fait en $O(X^3)$ avec $X = |P| + |T| + N_t$, où P et T désignent respectivement l'ensemble des places et des transitions de $SPEC_s$, et N_t dénote le nombre total des termes spécifiés dans les primitives de service de $SPEC_s$.
Par ailleurs, la construction de l'arbre des entrées pour chaque transition de $SPEC_s$ ainsi que l'évaluation de ses attributs SE et TS se font en $O(3|T|N_t)$. Par conséquent, la complexité majorée de la phase 0 est en $O(X^3)$.

Règle	Complexité majorée
SFD	$N_t^2\|T\| + N_t\|T\|N_{exp} + N_t\|T\|N_v$
RS1	$\|T\|^3$
RS2	$\mathcal{C}(SFD) + N_t + N_s(N_t + N_{exp} + N_v)$
RS3	$\mathcal{C}(SFD) + (N_t + N_{exp} + N_v)(N_s + 1) + N_tN_v$
RS4	$N_tN_v + N_t + N_{exp}$
RS5	$\mathcal{C}(SFD) + N_t$
RS6	$N_t\|P\|\|T\|^2 + N_t\|T\|^2(N_t + \|T\|) + \|T\|(N_t + N_{exp} + N_v)$
RS7	$\|P\|\|T\| + \|T\|\mathcal{C}(SFD)$
RS8	$\|P\| + \|P\|\|T\|N_t^2 + \|P\|(\|T\| + N_{exp} + N_v)$
RS9	$\|T\|^2 + \|T\|\mathcal{C}(SFD)$
RSi	$\|T\|^2$ $\forall i \in \{10, .., 14\}$

P : L'ensemble des places de $SPEC_s$
T : L'ensemble des transitions de $SPEC_s$
N_t : Le nombre total des termes spécifiés dans $SPEC_S$
N_{exp} : Le nombre totale des expressions conditionnelles spécifiées dans $SPEC_s$
N_v : Le nombre de variables de $SPEC_s$
N_s : Le nombre de sites de l'architecture cible

TAB. 2.2 – *Complexité des règles de synthèse*

2.4.2.2 Complexité de la Phase 1

Lors de la synthèse d'une transition ou d'une place de $SPEC_s$, chaque règle RS_i est appliquée au plus une seule fois. La complexité de cette phase est donc majorée par :

$$(|T| + |P|) \sum_{i=1}^{N_{RS}} \mathcal{C}(RS_i) \quad (1)$$

$\mathcal{C}(RS_i)$ désigne la complexité d'une règle RS_i et N_{RS} est le nombre total de règles contextuelles. La complexité de chaque règle ainsi que celle de l'algorithme SFD sont données dans le Tableau 2.2. En majorant chaque terme de $\mathcal{C}(RS_i)$ dans (1) par $Y = |P| + |T| + N_t + N_{exp} + N_v + N_s$, nous déduisons que la phase 1 s'effectue en $O(Y^5)$.

2.4.2.3 Complexité de la Phase 2

La phase 2 de l'algorithme consiste à substituer à toute place ou à toute transition de $SPEC_s$ un sous-réseau construit dans la phase 1. Elle se fait en $O(Z^2)$ telle que $Z = |P| + T + N_s$.

2.4.2.4 Complexité de la Phase 3

La phase de réduction démarre de la spécification de protocole construite par les phases 0, 1 et 2. Soient P_{ep} et T_{ep} respectivement les plus grands ensembles de places et de transitions associés à une entité de protocole, la complexité de chaque règle de réduction est donnée dans le Tableau 2.3. Puisque l'application de RR1 peut introduire des ϵ-transitions, alors une réduction RR1 est toujours suivie par RR3. Le cycle (RR1, RR3) est réitéré au maximum

Règle	Complexité majorée
RR1	$N_s\|T_{ep}\|^2$
RR2 (cas 1)	$N_s\|P_{ep}\|\|T_{ep}\|$
RR2 (cas 2)	$N_s\|T_{ep}\|^2$
RR3	$N_s\|T_{ep}\|\|P_{ep}\|^2$

N_s : Le nombre de sites de l'architecture cible
T_{ep} : Le plus grand ensemble de transitions associé à une entité de protocole
P_{ep} : Le plus grand ensemble de places associé à une entité de protocole

TAB. 2.3 – *Complexité des règles de réduction*

$n = N_s|T_{ep}|$ fois. Par ailleurs, RR2 n'est appliquée qu'une seule fois. En majorant les termes de la complexité de chaque règle par $W = N_s + |T_{ep}| + |P_{ep}|$, nous déduisons que la phase 3 se fait en $O(W^6)$.

2.5 Conclusion

Dans ce chapitre nous avons présenté une nouvelle méthode de synthèse de spécifications de protocoles de communication à partir de spécifications des services dans un modèle de réseaux de Petri interprétés. Notre approche traite l'ensemble des aspects suivants : (a) synthèse de traitements répartis engendrés par une transition de service, (b) synthèse du flot de contrôle, (c) synthèse du flot de données, (d) synthèse de structures sélectives distribuées, et (e) synthèse du contrôle de la cohérence des données. Par conséquent, elle possède un contexte de synthèse plus large relativement aux méthodes existantes. Notre approche constitue de ce fait un "bon" compromis entre le pouvoir d'expression du modèle de spécification de service et le pouvoir de synthèse. Dans notre approche nous avons défini un ensemble de règles de synthèse de base et un ensemble de règles de synthèse contextuelles. La capacité d'étendre l'un ou l'autre de ces deux ensembles par de nouvelles règles offre à notre approche un contexte de synthèse extensible.

Les spécifications construites dans la phase 2 de notre algorithme de synthèse peuvent présenter des synchronisations non nécessaires ou des transitions vides, nous avons alors défini trois règles de réduction permettant la simplification des spécifications de protocole. Cependant, la question de la complétude des réductions n'a pas été abordée, elle mérite d'être étudiée afin de savoir si l'on élimine toutes les redondances.

Nous avons prouvé que les protocoles générés par notre algorithme de synthèse sont correctes syntaxiquement et sémantiquement. Concernant la correction sémantique, nous avons montré qu'il existe une équivalence de traces entre un service et le protocole dérivé. Du fait que les règles de synthèse n'introduisent pas du non déterminisme dû à des événements inobservables, il nous semble qu'il y aurait même une équivalence observationnelle [MIL 89] entre les deux systèmes. Mais ce résultat doit être confirmé par une preuve formelle qui peut faire l'objet d'un travail futur.

Les applications de notre méthode sont diverses et concernent fondamentalement la construction automatique de logiciels de communication corrects.
Bien que des restrictions ont été faites sur la spécification des services, la classe des services spécifiables reste très importante par rapport aux méthodes de synthèse existantes et aussi par rapport aux systèmes réels. Notons, en particulier, que la restriction R1 n'exclue pas les systèmes à terminaison propre. En effet, un tel système peut facilement être transformé en un système vivant. Il suffit de rajouter des places et transitions qui réinitialisent le système une fois dans son état final.
Les restrictions R1 et R3 nous ont permis de simplifier les trois problèmes suivants :

- La synthèse du flot de données (cf. 2.3.3).
- Le traitement des structures sélectives distribuées : ce traitement devient compliqué notamment lorsqu'une transition participe dans plusieurs choix distribués.
- La détection de la concurrence dans un réseau de Petri : les algorithmes polynômiaux qui existent ne concernent que la classe des réseaux de Petri à choix libre vivants et bornés (cf. 2.3.2).

Comme perspective, il serait intéressant de reprendre la stratégie de synthèse et de l'affiner en essayant de lever (ou de relaxer) les restrictions R1 et R3, et donc d'augmenter davantage la classe des services spécifiables.

Chapitre 3

Application au protocole de transport ISO Classe 0

Résumé

Ce chapitre concerne une application réelle de notre approche de synthèse de protocoles [KAH 97a]. Nous présentons les étapes de conception du protocole de transport ISO classe 0 [ISO 8073]. Pour ce faire, nous appliquons l'algorithme de synthèse de protocoles que nous avons présenté dans le chapitre 2. Le point de départ de la conception est la spécification du service de transport ISO [ISO 8072].

3.1 Introduction

L'objectif de ce chapitre est de montrer l'importance et l'utilité d'une approche de synthèse, en particulier celle que nous avons présentée dans le chapitre 2, dans la conception de protocoles réels. Nous développons à cet égard les étapes de synthèse d'un protocole de transport ISO [ISO 8073] en utilisant notre algorithme de synthèse. Le point de départ de la conception est la spécification formelle du service de transport de l'ISO [ISO 8072]. Cette spécification est écrite dans un premier temps dans le modèle IPN. Elle est ensuite raffinée en utilisant les règles de synthèse contextuelles afin de produire les entités communicantes de la couche transport. Ces entités sont correctes "par construction". Elles définissent le protocole de transport ISO classe 0 [ISO 8073], puisqu'elles ne traitent pas le recouvrement d'erreurs et le contrôle du flux de données.

Dans les sections 3.2 et 3.3 nous présentons, respectivement, la description informelle puis formelle du service de transport de l'ISO. La section 3.4 développe les étapes de raffinement du service conduisant à la construction des spécifications du protocole de transport.

3.2 Description informelle du service de transport

Le service de transport assure un transfert transparent de données entre les utilisateurs du service. Il libère ces utilisateurs de toute préoccupation concernant les détails d'utilisation du support de communication. Le service de transport ISO peut être fourni avec ou sans connexion. Nous considérons dans ce chapitre le service avec connexion qui s'effectue en

trois phases: (1) établissement de la connexion de transport, (2) transfert de données, et (3) fermeture de la connexion de transport. Le Tableau 1 décrit les primitives du service de transport associées à chacune des trois phases d'une connexion.

Phase	Primitive de service	Notation
Établissement de la connexion de transport	T-CONNECT.request(cg, cd, qos, d) T-CONNECT.indication(cg, cd, qos, d) T-CONNECT.response(rg, qos, d) T-CONNECT.confirmation(rg, qos, d)	TConReq TConInd TConResp TConConf
Transfert de données	T-DATA.request(d) T-DATA.indication(d) T-EXPEDITED-DATA.request(d) T-EXPEDITED-DATA.indication(d)	TDataReq TDataInd TXdataReq TXdataInd
Fermeture de la connexion de transport	T-DISCONNECT.request(d) T-DISCONNECT.indication(r, d)	TDiscReq TDiscInd

cg : l'adresse du TSAP de l'entité appelante
cd : l'adresse du TSAP de l'entité appelée
qos : la qualité de service négociée de la connexion
d : données utilisateur à transférer
rg : l'adresse du TSAP avec lequel la connexion de transport a été établie
r : la raison de la déconnexion

TAB. 3.1 – *Les primitives du service de transport ISO*

L'exécution d'une primitive de service à une extrémité de la connexion a, en général, des conséquences à l'autre extrémité de cette connexion. Les enchaînements valides des primitives du service de transport au niveau des deux extrémités de la connexion de transport sont résumés dans la Figure 3.1.

La Figure 3.1a illustre un établissement réussi d'une connexion de transport. Une extrémité qui désire ouvrir une connexion de transport invoque la primitive *TConReq* en indiquant en paramètres: l'adresse du point d'accès au service de transport (TSAP) de l'extrémité destinataire. Cette primitive a pour conséquence l'occurrence de la primitive *TConInd* au TSAP du destinataire. L'utilisateur attaché à cette extrémité peut soit accepter la connexion en invoquant la primitive *TConResp*, soit refuser la connexion en utilisant la primitive *TDiscReq* (cf. Figure 3.1b). Dans le cas d'une ouverture de connexion réussie, le demandeur de la connexion est mis au courant par la primitive *TConConf*, alors que dans le cas contraire, il est avisé par la primitive *TDiscInd*. Cette primitive indique en paramètre la raison de la déconnexion. Il arrive dans certains cas que la connexion soit refusée directement par le fournisseur du service (cf. Figure 3.1c). Une fois que la connexion est établie, une extrémité peut demander un transfert de données par la primitive *TDataReq* (ou *TXdataReq*, s'il

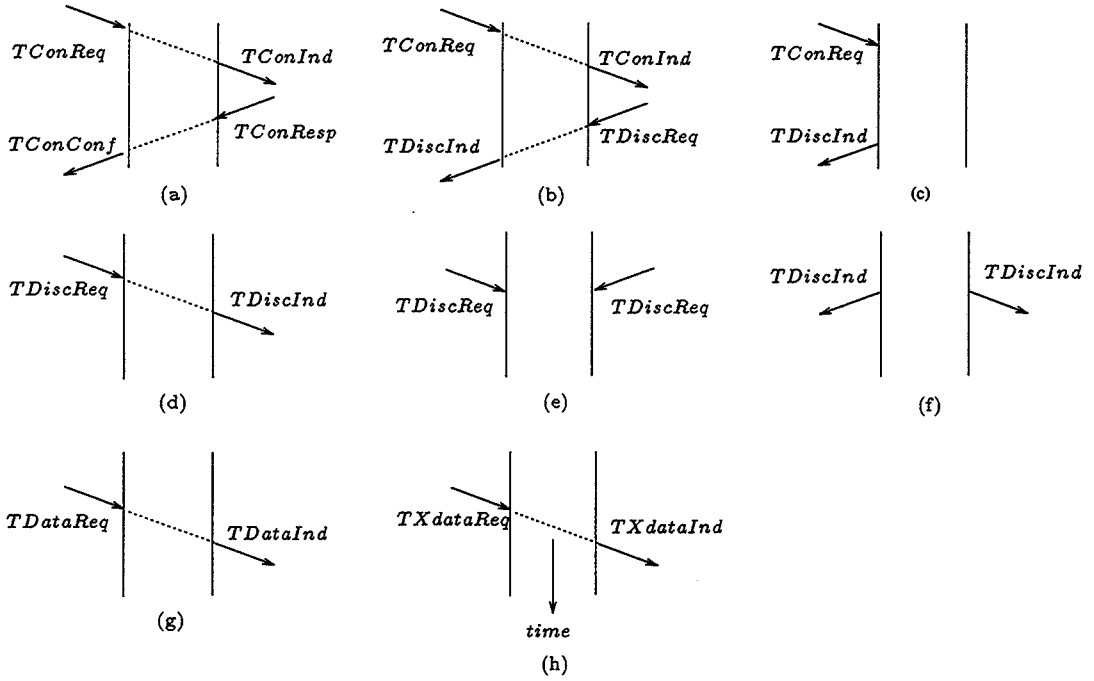

FIG. 3.1 – *Séquences valides des primitives du service de transport ISO*

s'agit de données prioritaires). L'extrémité correspondante reçoit alors les données au moyen de la primitive *TDataInd* (ou *TXdataInd*). Dans les Figures 3.1d-f, trois cas concernant la fermeture d'une connexion sont illustrés : fermeture à l'initiative d'une extrémité de la connexion (cf. Figure 3.1d), fermeture à l'initiative des deux extrémités simultanément (cf. Figure 3.1e), et fermeture à l'initiative du fournisseur du service (cf. Figure 3.1f).

L'une des difficultés principales dans le développement de protocoles ISO est de s'assurer d'une part, que le protocole fournit toutes les séquences valides des primitives de service et qu'il n'introduit pas d'autres séquences et d'autre part, que les valeurs des paramètres circulant entre les primitives de service sont correctes. D'où l'intérêt d'une méthode formelle qui libère le concepteur de cette difficulté.

3.3 Description formelle du service de transport

Il s'agit de spécifier, dans le modèle IPN (cf. chapitre 2), le service de transport ISO entre deux TSAPs a et b. Pour simplifier la spécification nous posons les deux hypothèses suivantes :

- La connexion est toujours demandée à partir du TSAP a.
- Le transfert de données se fait dans un seul sens ; du TSAP a vers le TSAP b.

Par ailleurs, nous considérons toutes les séquences valides définies dans la Figure 3.1.

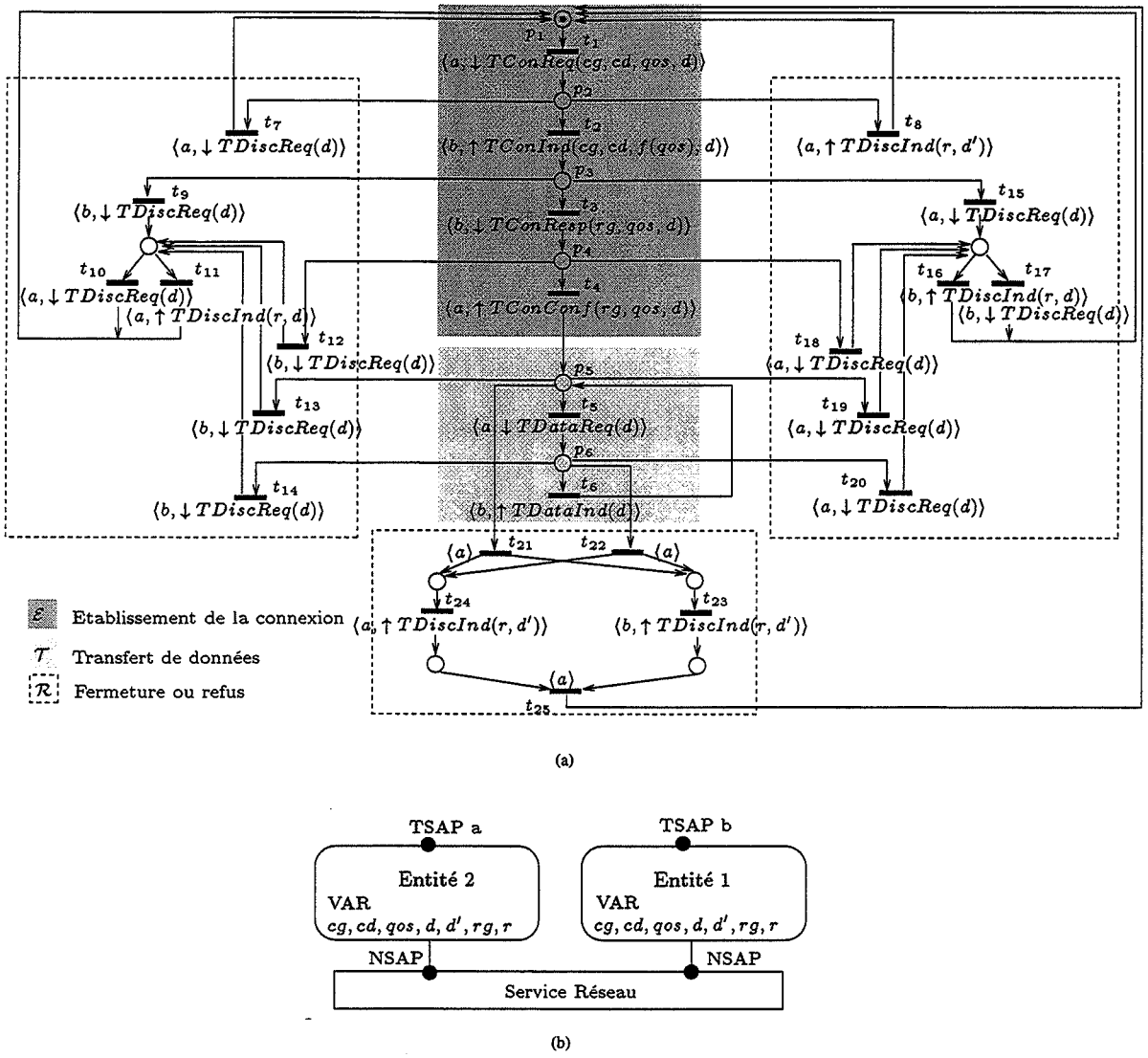

FIG. 3.2 – *Service de transport ISO (a) Spécification IPN du service de transport ISO (b) Architecture cible*

La spécification du service est donnée par la Figure 3.2a. Cette spécification décrit les trois phases d'une connexion de transport par les modules de réseaux de Petri suivants : $\mathcal{E}$, $\mathcal{T}$ et $\mathcal{R}$. Le module $\mathcal{E}$ décrit une séquence réussie de l'établissement d'une connexion de transport. Le module $\mathcal{T}$ décrit les séquences liées au transfert de données. Les modules $\mathcal{R}$ décrivent tous les cas possibles conduisant à une fermeture ou un refus de la connexion de transport. Les modules $\mathcal{E}$ et $\mathcal{T}$ sont des processus séquentiels qui peuvent être interrompus à tout moment par l'un des modules $\mathcal{R}$. L'interruption peut-être spécifiée dans notre modèle par une structure sélective à choix libre[1] avec priorités ; ainsi les transitions qui interrompent sont

1. Une structure sélective à choix libre est définie par une place p ayant plusieurs transitions en sortie telle que ces transitions n'ont pas d'autres places en entrée que p. Pour plus de détails se reporter au chapitre 2.

munies de priorités les plus hautes alors que celles à interrompre sont munies de priorités les plus basses. Les priorités n'étant pas encore acceptées par notre modèle, nous spécifions, pour le moment, les interruptions par des structures sélectives sans priorités. L'extension du modèle par un mécanisme de priorités peut se faire sans difficultés.
Notons que la fermeture de la connexion de la part du fournisseur de service peut intervenir pendant la phase d'ouverture de la connexion, c'est-à-dire quand l'une des places p_3 ou p_4 est marquée. Nous avons volontairement omis les possibilités de fermeture à partir de ces places pour des raisons de clarté de la figure.

La Figure 3.2b définit l'architecture cible sur laquelle sera implanté le service de transport. Cette architecture définit deux entités de transport EP_1 et EP_2 attachées respectivement aux TSAPs a et b. A chaque entité est associé un ensemble de variables locales ; à savoir les variables ayant fait l'objet d'au moins une référence à partir du TSAP auquel est attachée cette entité. Les primitives de service exécutées par les entités de transport EP_1 et EP_2 à travers les points d'accès de la couche réseau (NSAPs) sont en fait des émissions et des réceptions de messages produites automatiquement par l'algorithme de synthèse.
Il est facile de montrer, en utilisant les règles de réduction de réseau de Petri [BER 86] [DAV 92], que le réseau de la Figure 3.2 est vivant et 1-borné. Donc la spécification du service de transport est correcte et peut constituer une entrée à notre algorithme de synthèse de protocoles.

3.4 De la spécification du service à la spécification du protocole

Partant de la spécification du service de transport donnée par la Figure 3.2a, il s'agit de construire deux entités de transport comme indiqué par la Figure 3.2b. Pour ce faire, nous appliquons les étapes de notre algorithme de synthèse.

3.4.1 Phase 1 : Raffinement des places et transitions

Cette phase consiste à raffiner de manière systématique chaque place ou transition u de la spécification de service en deux sous-réseaux de Petri communicants. Chaque sous-réseau définit la contribution d'une entité de protocole dans l'implantation de la place ou transition u. Le raffinement est guidé par un ensemble complet de règles de synthèse de haut niveau, dites règles contextuelles[2]. Dans la spécification de service de la Figure 3.2a, nous distinguons 5 places à choix distribué, il s'agit des places p_2 à p_6. Nous présentons dans ce qui suit les étapes de raffinement pour la transition t_1 et la place p_2, les autres places et transitions sont raffinées selon le même principe.

3.4.1.1 Raffinement de la transition t_1

La Figure 3.3 montre les étapes de raffinement de la transition t_1. Initialement, le sous-réseau de Petri implantant la transition t_1 dans l'entité EP_1 contient seulement une copie de t_1, et le sous-réseau implantant t_1 dans EP_2 est vide.

2. L'ensemble des règles contextuelles est donné au chapitre 2

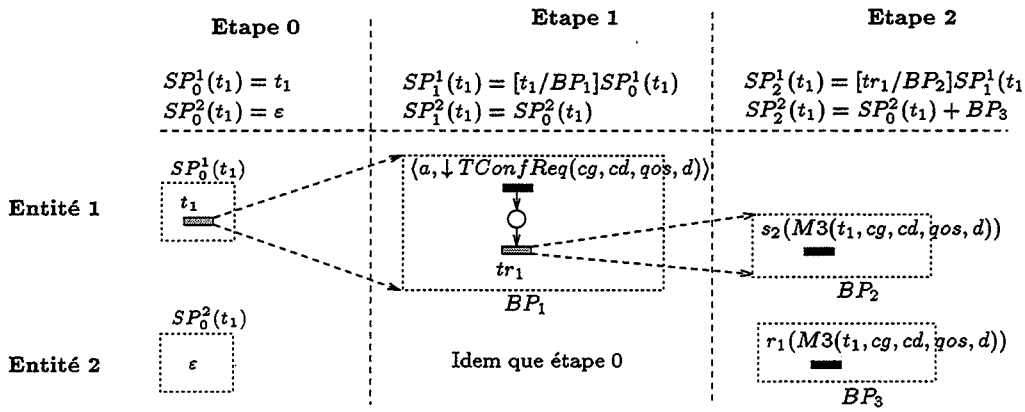

FIG. 3.3 – *Étapes de raffinement de la transition t_1*

A l'étape 1, la règle RS1 est appliquée pour raffiner l'entête de la transition t_1 (à savoir la clause "input"). La transition se décompose ainsi en deux transitions séquentielles : la première transition spécifie l'événement en entrée $\downarrow TConReq(cg, cd, qos, d)$, et la seconde dénote un point de raffinement servant pour les prochaines étapes. A cette étape, une seule transformation est appliquée ; notamment la substitution du bloc BP_1 à la transition t_1.
A l'étape 2, la règle RS6 est appliquée pour synthétiser le transfert de données entre l'entité de protocole détenant t_1 (i.e., EP_1) et les entités détenant les transitions successeurs de t_1 (i.e., EP_2). En effet, la primitive de service étiquetant t_1 (i.e., $\mathcal{X}(t_1)$) possède quatre paramètres en sortie : *cg*, *cd*, *qos* et *q*. La primitive $\mathcal{X}(t_2)$ possède les même paramètres en entrée. Un message de données contenant les valeurs des quatre paramètres est donc envoyé de EP_1 vers EP_2 (message de type M3). Ce message véhicule, en plus, l'identité de la transition de service raffinée (i.e., t_1). A cette étape, deux transformations sont appliquées : la substitution du bloc BP_2 qui spécifie l'émission du message M3 et l'addition du bloc BP_3 qui spécifie la réception de ce message.

3.4.1.2 Raffinement de la place p_2

La Figure 3.4 montre les étapes de raffinement de la place à choix distribué p_2. Une fois que la place p_2 devient marquée, les deux entités de transport s'engagent simultanément dans le choix de la transition à franchir. Ce type de place est raffiné à l'aide des règles RS11 à RS14 qui consistent à synthétiser un mécanisme d'élection distribuée entre les entités qui participent au choix. Rappelons que les blocs substitués par ces règles possèdent une forme particulière ; ils sont constitués d'une seule place marquée à l'état initial du bloc et une ou plusieurs places marquées à l'état final du bloc. Une propriété comportementale caractérise aussi ces blocs ; à savoir qu'à partir de leur état initial, ils atteignent inévitablement un état final où une et une seule place finale est marquée. Nous avons montré dans le chapitre 2 que la substitution de tels blocs maintient les propriétés "vivant" et "borné". Initialement, les sous-réseaux de Petri implantant la place p_2 contiennent chacun une copie de p_2. A l'étape 1, deux transformations sont appliquées afin de synthétiser le mécanisme d'élection. Le bloc BP_2 spécifie qu'une fois la place p_2 est marquée dans l'entité EP_2, cette

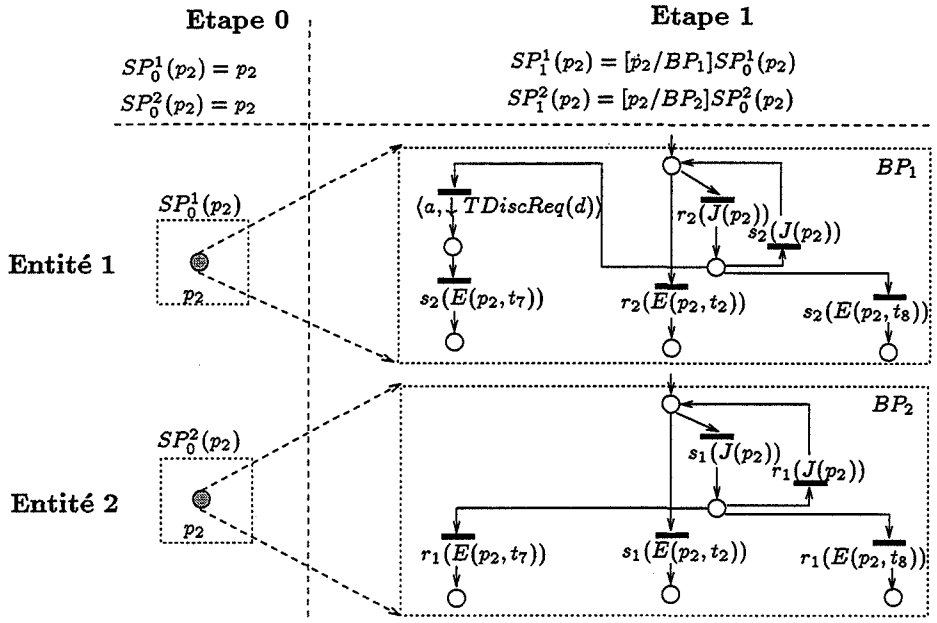

FIG. 3.4 – *Étapes de raffinements de la place p_2*

dernière aura le choix entre indiquer un événement de demande de connexion au TSAP b, $\uparrow TConInd(cg, cd, f(qos), d)$, ou envoyer un message de synchronisation de type $JETON$ (noté $J(p_2)$) à EP_1. La fonction $f(qos)$ indique la qualité de service calculée au TSAP b en fonction de celle fournie par le TSAP a. Le bloc BP_1 spécifie ce qui se passe dans EP_1. Ainsi, quand EP_1 reçoit le message $J(p_2)$, elle a la possibilité d'exécuter l'événement de demande de déconnexion $\downarrow TDiscReq(d)$, d'indiquer un événement de déconnexion de la part du fournisseur de service $\uparrow TDiscInd(d)$, ou d'envoyer le message $J(p_2)$ à EP_2. Une entité de protocole qui sélectionne une transition donnée, notifie cette sélection en diffusant à toutes les autres entités un message de type ELU (noté $E(p_2, t_7)$ si la transition t_7 est choisie). La place p_2 est, par conséquent, synthétisée en $SP_1^1(p_2)$ et $SP_1^2(p_2)$. Ces sous-réseaux respectent bien la propriété comportementale citée plus haut.

3.4.2 Phase 2 : Construction des entités de protocole

L'entité de transport EP_1 (resp. EP_2) est construite en remplaçant dans la spécification du service chaque place ou transition par le sous-réseau obtenu dans la phase précédente. Les spécifications obtenues sont données par les Figures 3.5 et 3.6. Ces spécifications contiennent des transitions non étiquetées notées par ϵ et quelques synchronisations non nécessaires qui doivent être supprimées dans la phase suivante.

3.4.3 Phase 3 : Réduction des entités de protocole

Cette phase consiste à éliminer les synchronisations non nécessaires, de regrouper certains messages et d'éliminer les transitions vides.

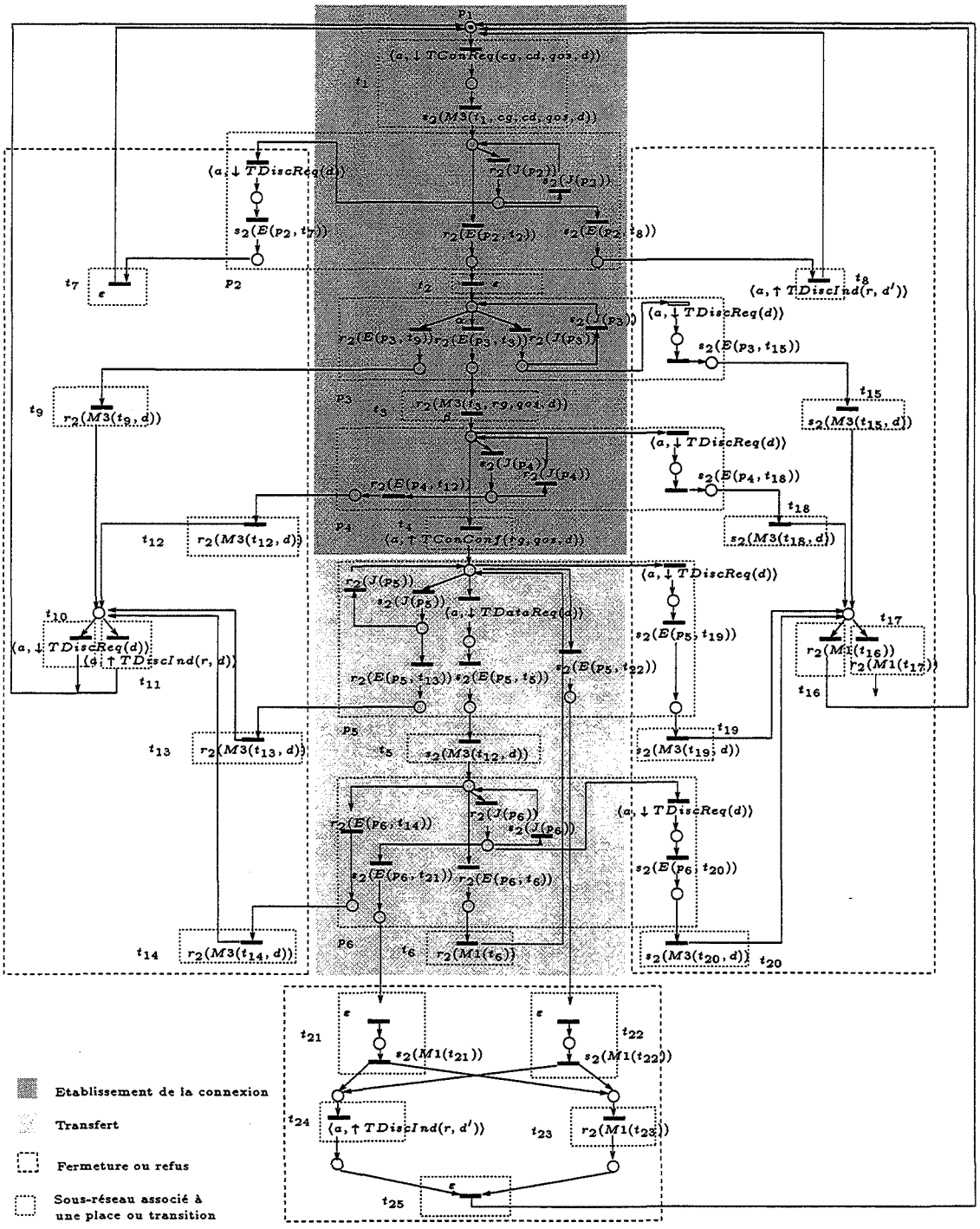

FIG. 3.5 – *Spécification non réduite de l'entité de transport EP_1*

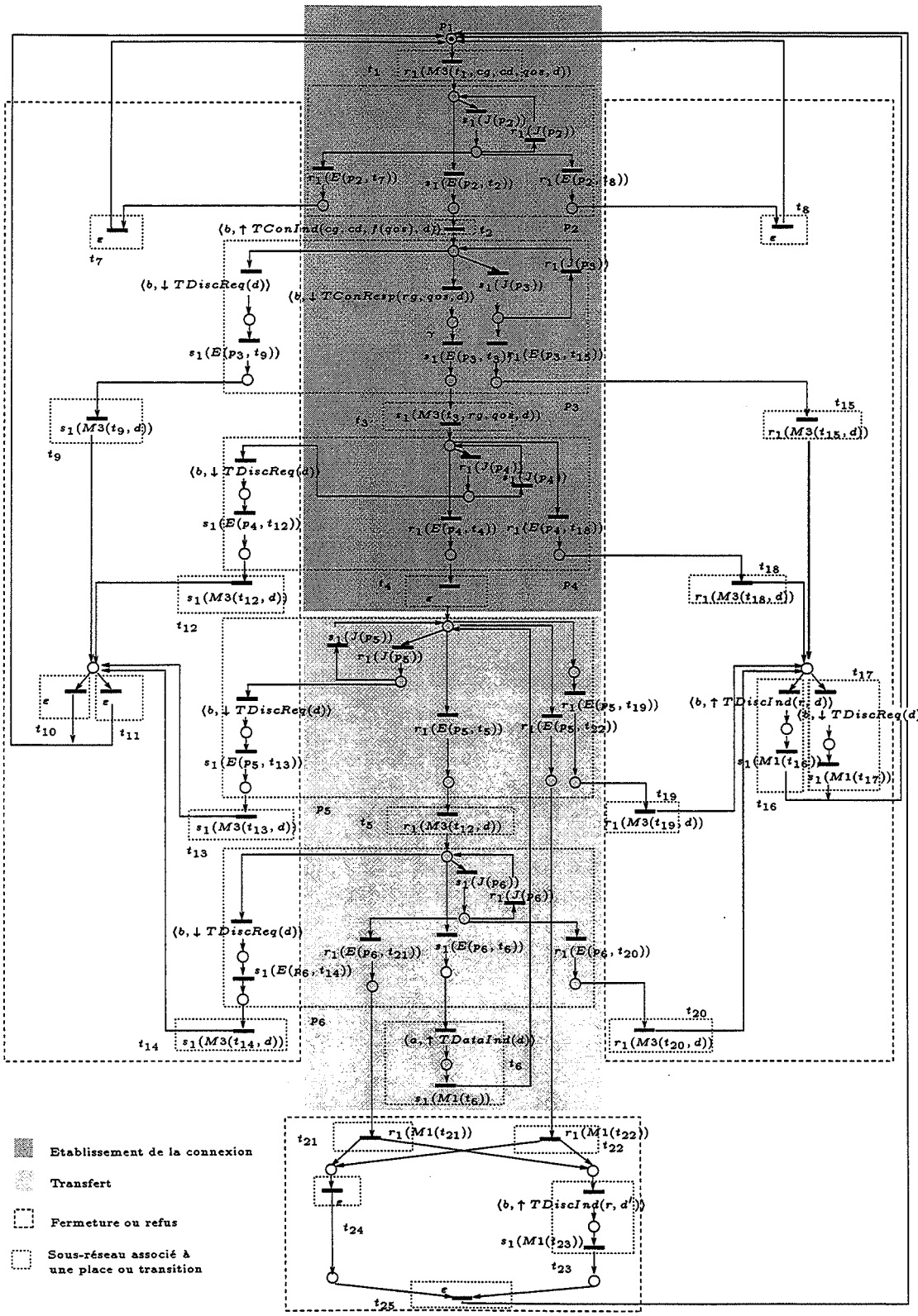

FIG. 3.6 – *Spécification non réduite de l'entité de transport EP_2*

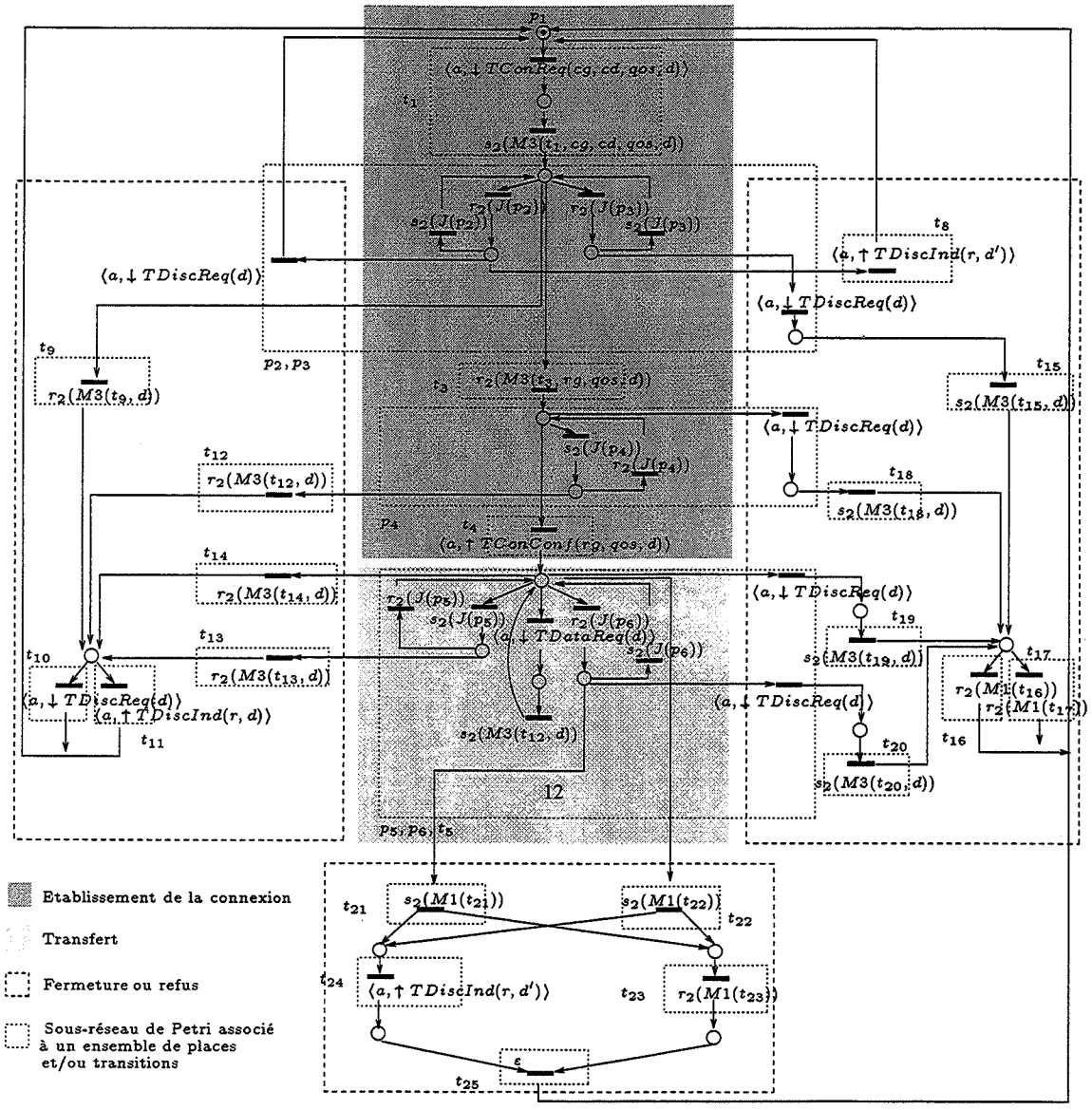

FIG. 3.7 – *Spécification finale de l'entité de transport EP_1*

Ceci est réalisé en utilisant, respectivement, trois règles de réduction RR1, RR2 et RR3 (cf. chapitre 2). Pour ce faire, on commence par éliminer les ϵ-transitions des deux entités EP_1 et EP_2 en utilisant la règle RR1. Par exemple, l'entité EP_1 ne participe pas dans l'implantation de la transition t_2. De ce fait, le sous-réseau de Petri implantant t_2 dans EP_1 est défini par une ϵ-transition. Cette transition peut être supprimée afin de simplifier la spécification. Par ailleurs, considérons la transition α dans EP_1. Cette transition est étiquetée par la réception d'un message de synchronisation $r_2(E(p_3, t_3))$. Cette transition possède un seul successeur ; la transition β.

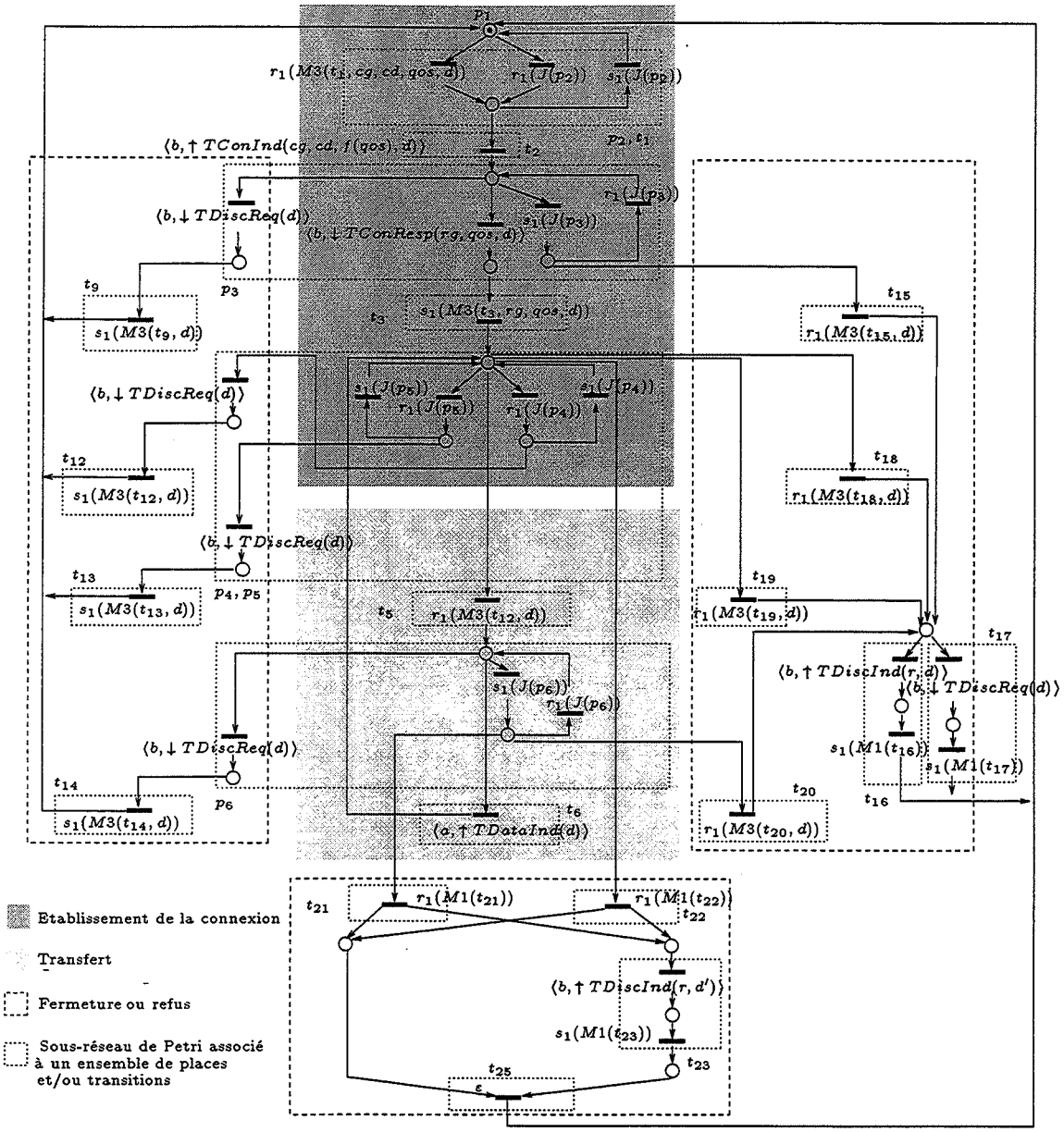

FIG. 3.8 – *Spécification finale de l'entité de transport EP_2*

Cette dernière est étiquetée par la réception d'un message de donnée $r_2(M3(t_3, rg, qos, d))$. La réception de ce message implique que les sous-réseaux associés à la transition t_3 ont atteint leur état initial, et que les sous-réseaux associés à la place p_3 ont déjà atteint leur état final. Par conséquent, la synchronisation définie par la réception $r_2(E(p_3, t_3))$ s'avère non nécessaire. La règle RR2 consiste, alors, à remplacer la transition α, ainsi que la transition d'émission qui lui correspond dans EP_2 (i.e., γ), par des ϵ-transitions. Les ϵ-transitions qui apparaissent après application de RR2 sont éliminées par la règle RR1. On applique à nouveau RR2 aux deux entités jusqu'à ce qu'il n'y ait plus de synchronisations non nécessaires.

Message	T-PDU équivalente	Sémantique
$J(p_i), 2 \leq i \leq 6$		Message de synchronisation servant à réaliser la structure sélective distribuée définie par la place p_i dans la spécification de service
$M3(t_1, cg, cd, qos, d)$	$CR(cg, cd, qos, d)$	Demande de connexion
$M3(t_3, rg, qos, d)$	$CC(rg, qos, d)$	Confirmation de la connexion
$M3(t_{12}, d)$	$DT(d)$	Message de données
$M3(t_9, d)$ ou $M3(t_{12}, d)$ ou $M3(t_{13}, d)$ ou $M3(t_{14}, d)$ ou $M3(t_{15}, d)$ ou $M3(t_{18}, d)$ ou $M3(t_{19}, d)$ ou $M3(t_{20}, d)$ ou $M1(t_{17})$	$DR(d)$ ou DR	Demande de fermeture de la connexion
$M1(t_{16})$	DC	Confirmation de la déconnexion
($M1(t_{21})$ et $M1(t_{23})$) ou ($M1(t_{22})$ et $M1(t_{23})$)		Messages indiquant une déconnexion aux 2 extrémités de la part du fournisseur de service

TAB. 3.2 – *Les messages échangés entre les entités de transport*

La règle RR3 consiste à regrouper certains messages ; par exemple deux messages de données envoyés séquentiellement (ou parallèlement) à une même destination peuvent être regroupés en un seul message. Cette règle est basée sur le principe d'agglomération présenté dans [BER 86]. L'agglomération consiste à réaliser de façon indivisible certaines séquences de franchissement en conservant les bonnes propriétés du réseau. Dans le cas des entités EP_1 et EP_2, Il n'y a pas de regroupements possibles à effectuer. Les entités de protocole obtenues après réduction sont données par les Figures 3.7 et 3.8. Ces entités correspondent au protocole de transport ISO classe 0 ; en effet, elles réalisent le service de transport sans prendre en compte le recouvrement d'erreurs ni le contrôle du flux de données[3].
Les correspondances entre les messages produits par l'algorithme de synthèse et les unités de données (T-PDUs) du protocole de transport sont données par le Tableau 3.2.

3.4.4 Scénario d'exécution

La Figure 3.9 montre un scénario d'exécution des entités de transport EP_1 et EP_2. Un utilisateur exécute une demande de connexion au TSAP a, l'entité EP_1 transmet alors un message de demande de connexion $M3(t_1, cg, cd, qos, d)$ à EP_2. L'entité EP_2 indique à l'utilisateur attaché au TSAP b la demande de connexion par la primitive $TConInd(cg, cd, f(qos), d)$. L'utilisateur du TSAP b peut soit refuser la connexion en exécutant $TDiscReq(d)$, soit l'accepter en exécutant $TConResp(rg, qos, d)$. Dans le cas où la connexion est acceptée, l'entité EP_2 envoie à EP_1 un message de confirmation $M3(t_3, rg, qos, d)$. L'entité EP_1 notifie alors la confirmation à son utilisateur par la primitive $TConConf(rg, qos, d)$. L'utilisateur du TSAP a commence alors à transférer des données en exécutant une ou plusieurs fois la primitive $TDataReq(d)$. L'entité EP_1 envoie un ou plusieurs messages de données $M3(t_{12}, d)$ à EP_2. L'entité EP_2 exécute la primitive $TDataInd(d)$ à chaque réception d'un message

3. La synthèse de ce type de fonctionnalités est discutée dans le chapitre 4.

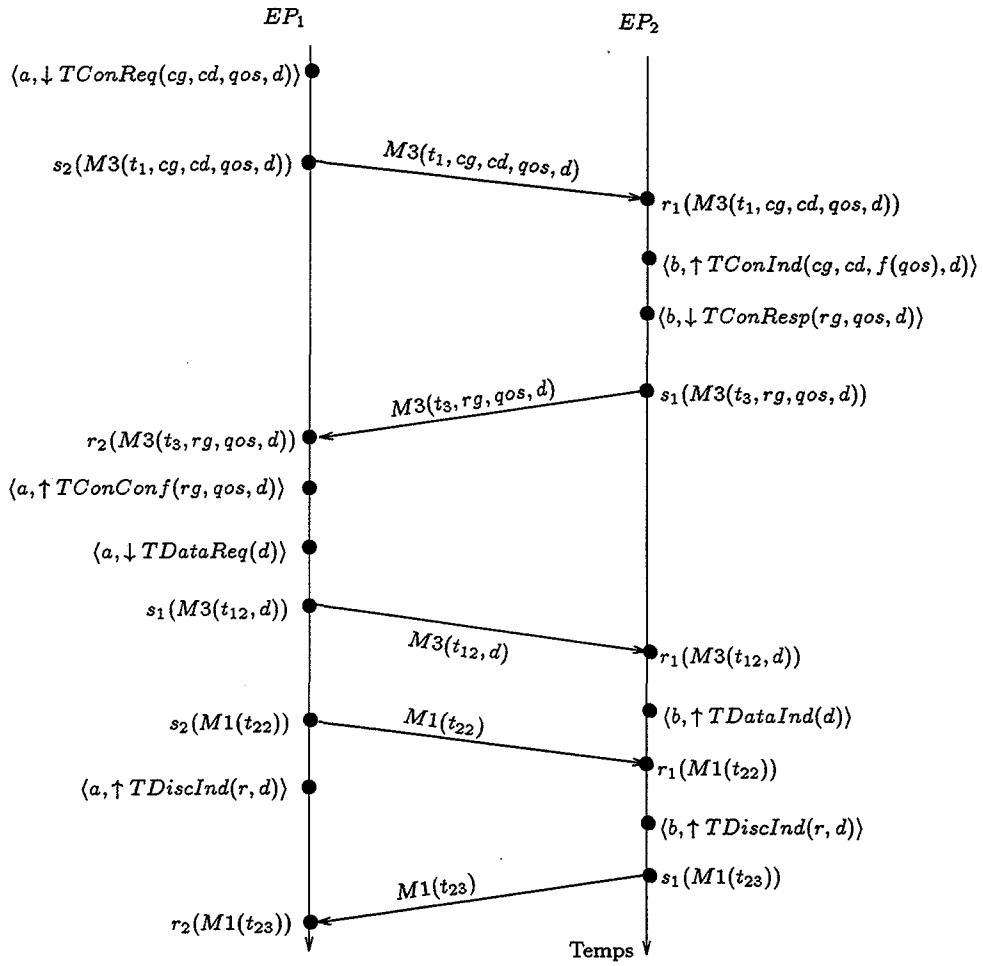

FIG. 3.9 – *Scénario d'exécution des entités de transport EP_1 et EP_2*

$M3(t_{12}, d)$. Dans ce scénario, la connexion est fermée par le fournisseur du service. L'entité EP_1 envoie un message de synchronisation $M1(t_{22})$ à EP_2 et indique à son utilisateur la fermeture de la connexion par la primitive $TDiscInd(r, d)$. Lorsque EP_2 reçoit le message de synchronisation, elle indique à son tour la déconnexion au TSAP b en exécutant la même primitive et envoie à EP_1 le message $M3(t_{23})$ qui confirme la déconnexion. La connexion est alors fermée et les deux entités reprennent leur état initial.
Les messages de synchronisation $J(p_i), 2 \leq i \leq 6$ assurent que les deux entités de protocole n'entrent jamais dans un état où plusieurs transitions de service alternatives en sortie de p_i sont en train de s'exécuter simultanément.

3.5 Conclusion

Nous avons présenté dans ce chapitre les étapes de conception du protocole de transport ISO classe 0 [ISO 8073] selon une approche de synthèse. Partant de la spécification du service de transport ISO [ISO 8072], nous avons dérivé les deux entités de la couche transport.

Pour ce faire nous avons utilisé une méthode de synthèse de protocole basée sur les raffinements successifs de réseaux de Petri [KAH 96]. Les entités de protocole construites sont correctes. Par conséquent, nous n'avons pas eu besoin d'une phase d'analyse pour s'assurer de la validation du protocole. Les méthodes de synthèse existantes telles que [GOT 90a], [YAM 95], [KAN 93a] et [KHO 95] n'accepteraient pas en entrée la spécification du service de transport considéré dans ce chapitre. En effet, les trois premières méthodes ne traitent pas le problème du choix distribué, et la dernière méthode ne considère pas des primitives de service paramétrées. Néanmoins, des améliorations restent à étudier pour notre approche de conception :

1. extension du modèle IPN par un mécanisme de priorités. Ceci permettra de faciliter la spécification des interruptions.

2. certaines fonctionnalités protocolaires, telles que le recouvrement d'erreurs, le contrôle de flux, ne peuvent pas être spécifiées au niveau service, et ne peuvent donc pas être synthétisées automatiquement par notre algorithme de synthèse. Ceci requiert, donc, l'extension de notre approche de conception afin de pouvoir prendre en compte des fonctionnalités intrinsèques au niveau protocole.

3. extension du modèle de spécification par des contraintes de temps.

Chapitre 4

STEPS, un outil logiciel pour la synthèse automatique de protocoles

Résumé

Afin de démontrer l'utilité et la faisabilité de l'approche de synthèse que nous avons présentée au chapitre 2, nous avons développé un ensemble d'outils logiciels la supportant appelé STEPS (Software Toolset for automatEd Protocol Synthesis) [KAH 97c]. Ce chapitre est consacré à la présentation et l'expérimentation des outils STEPS.

4.1 Introduction

Nous présentons dans ce chapitre un ensemble d'outils logiciels pour la synthèse automatique de protocoles que nous appelons STEPS (Software Toolset for automatEd Protocol Synthesis). Nous avons développé cet ensemble d'outils afin de démontrer l'utilité et la faisabilité de l'approche de synthèse que nous avons présentée au chapitre 2.
De nombreux outils logiciels ont été mis en oeuvre pour faciliter le développement des protocoles et ainsi diminuer leur coût de production. Une grande partie de ces outils est basée sur des techniques analytiques telles que l'analyse d'accessibilité et la simulation. Parmi ces outils, nous pouvons citer XEXAR [RIC 87], ESTIM [COU 92], EDT [BUD 92] et VESAR [ALG 93]. Une liste relativement complète est présentée dans [CHA 93].

Par ailleurs, très peu d'outils sont basés sur des techniques de synthèse. Nous pouvons citer : APS [RAM 85], SEPS [SHI 91] et [CHA 94b]. APS est l'un des premiers outils utilisant l'approche de synthèse, il est basé sur une méthode de synthèse de protocoles à deux entités dans un modèle d'automates, présentée dans 1.5.1. Il est limité aux protocoles à deux entités, ne considère pas le service fourni par le protocole et utilise la construction de graphe d'accessibilité. SEPS possède les mêmes limites que APS. L'outil présenté dans [CHA 94b] intègre la méthode utilisée par APS et la technique de raffinement interactif de réseaux de Petri décrite dans 1.5.2. Cet outil ne prend pas en compte les services fournis par un protocole. Il assure uniquement la correction logique des protocoles conçus. La correction sémantique requiert une phase d'analyse supplémentaire.

L'apport principal de STEPS par rapport à ces trois outils *est la prise en compte des spécifications de service dans le développement des protocoles de communication et des applications réparties.* L'intérêt de cet apport est double. D'une part, il permet de diminuer le coût de production des protocoles et des applications réparties. D'autre part, il garantit la correction logique et sémantique des spécifications produites. Ceci requiert bien entendu la correction logique des spécifications du service ayant servi comme point de départ dans le processus de développement.

Dans la section suivante nous développons les fonctionnalités et la structure de STEPS. La section 4.3 est consacrée à l'interface utilisateur définie par un langage de spécification des services et des protocoles. Dans la section 4.4 nous développons des expérimentations et les résultats obtenus. La section 4.5 présente les caractéristiques intrinsèques de STEPS. Nous terminons par des améliorations possibles de STEPS et une conclusion.

4.2 Fonctionnalités et structure de STEPS

Nous présentons ici les différentes fonctionnalités et la structuration des outils STEPS, ce qui implique la définition d'une méthodologie de développement des protocoles. Par ailleurs, nous montrons également les étapes suivies dans le développement de STEPS qui, à notre avis, serviront de référence pour son extension future.

4.2.1 Présentation générale

La première phase dans la conception d'un protocole de communication ou d'une application répartie est de fournir la spécification formelle du service que l'on souhaite rendre aux utilisateurs. Pour ce faire, nous avons défini un langage appelé IPNL (Interpreted Petri Net Language) pour la spécification des services et des protocoles. Ce langage est basé sur le modèle IPN que nous avons présenté au chapitre 2. Une description détaillée du langage est présentée dans la section 4.3.
La spécification IPNL du service est ensuite soumise au module ANALYSEUR afin d'en vérifier la syntaxe et la sémantique (cf. Figure 4.1). Dans le cas où la phase d'analyse se termine avec succès, *une structure de donnée interne* est construite. Elle représente une forme intermédiaire sur laquelle agissent les différents modules de STEPS. Une fois la structure interne construite, elle est exploitée par le module VERIFICATEUR dont le rôle est de vérifier la correction logique de la spécification du service. Dans notre cas, il s'agit de vérifier la *vivacité* et le *caractère borné* du réseau de Petri sous-jacent. Si l'une de ces propriétés n'est pas satisfaite, un diagnostic est délivré à l'utilisateur et la conception reprend du début.
Par contre, dans le cas où les deux propriétés sont satisfaites, la phase de synthèse est activée. Elle consiste à raffiner en une ou plusieurs étapes la structure interne du service afin de construire la structure interne du protocole. Le raffinement est guidé par un ensemble de règles de synthèse qui maintiennent la correction logique et sémantique des spécifications de façon incrémentale. Quand la phase de synthèse s'achève, la structure interne du protocole est soumise à un module de génération de code. Plusieurs modules de ce type peuvent être intégrés dans STEPS, chacun ayant la capacité de produire du code dans un langage spécifique partant de la forme interne du protocole.

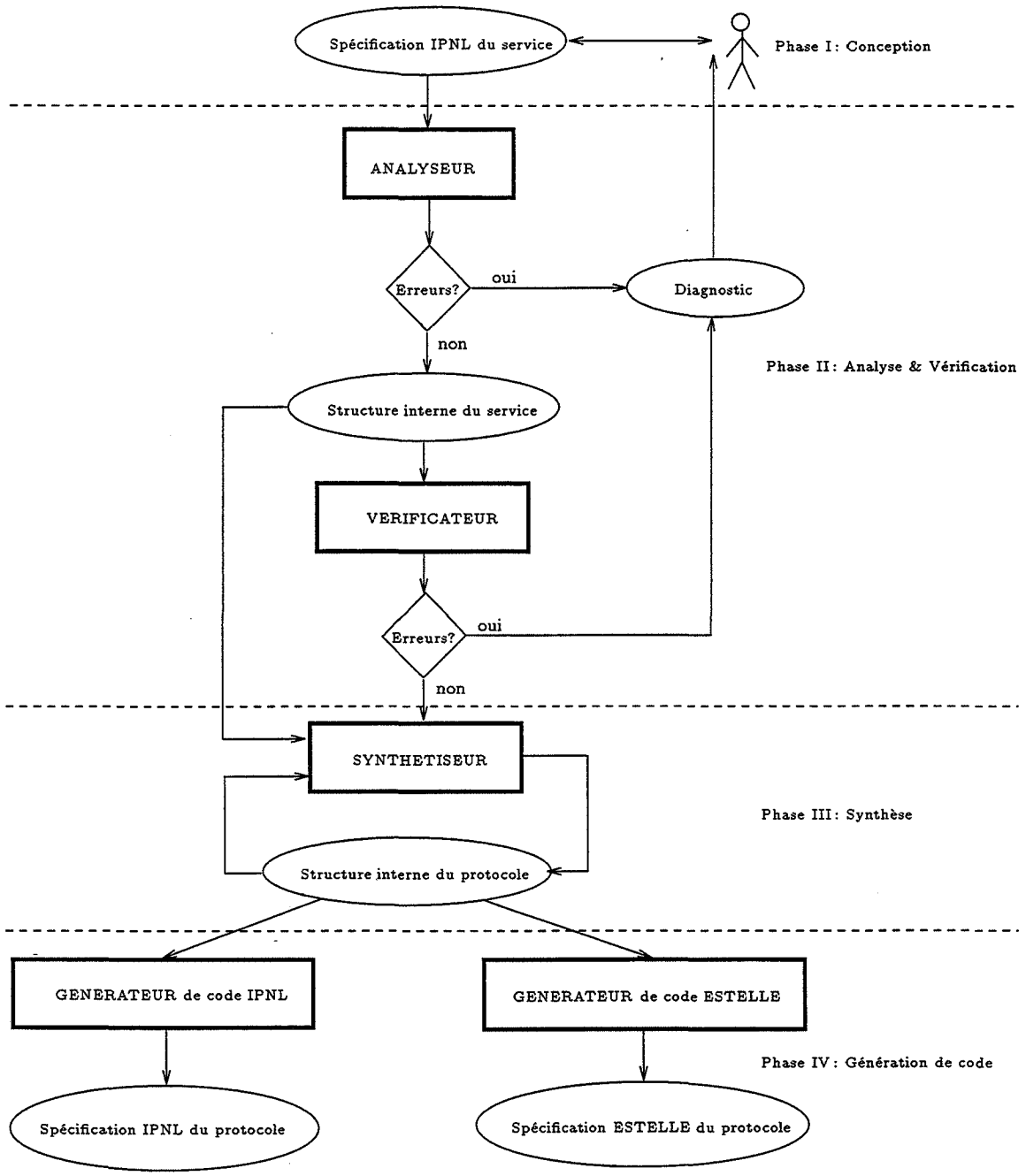

FIG. 4.1 – *Schéma fonctionnel de STEPS*

Nous avons prévu deux modules à cet égard : un module pour la génération de code dans le langage IPNL et un module pour la production de code dans le langage standard Estelle. STEPS génère par défaut des spécifications dans le langage IPNL. La génération de code Estelle a pour avantage de mettre les spécifications du protocole à la portée d'un large public. Ainsi, ces spécifications peuvent être directement exploitées par les outils existants autour

d'Estelle pour produire du code exécutable ou effectuer une évaluation de performances. D'autre part, un dispositif spécialisé pour l'exécution des spécifications IPNL peut également être efficace et réaliste notamment pour certains domaines d'application comme les systèmes embarqués.

4.2.2 Module de synthèse

Le module de synthèse représente le maillon principal de STEPS. Il est composé d'un ensemble de règles de base, un ensemble de règles contextuelles et un moteur de synthèse (cf. Figure 4.2). Nous décrivons dans ce qui suit chacun de ces composants. Les règles de synthèse de base sont définies à partir du modèle de réseaux de Petri et sont indépendantes de l'interprétation attribuée au réseau. Rappelons que nous avons défini trois règles de base au chapitre 2 : la T-Substitution, la P-Substitution et l'Addition. La T-Substitution (resp. la P-Substitution) est spécifiée par une fonction qui, recevant en entrée une transition (resp. une place) et un bloc à substituer, change la nature de la transition (resp. la place) qui devient ainsi une *macro-transition* (resp. une *macro-place*) [1] et établit un lien entre cette transition (resp. cette place) et le bloc qui la substitue. L'Addition est également spécifiée par une fonction qui, partant de deux blocs de réseaux de Petri ajoute le second dans le premier. L'Addition se fait en créant une macro-transition dans le premier bloc et en établissant un lien entre cette macro-transition et le deuxième bloc.

Les règles de synthèse contextuelles dépendent de l'interprétation attribuée au réseau de Petri, elles sont construites en poursuivant les 3 étapes suivantes (cf. Figure 4.2) :

- spécification informelle du contexte de la synthèse : le contexte de la synthèse représente les fonctionnalités qu'on se propose de synthétiser au niveau protocole (cf. Figure 4.2).

- spécification des différents types de messages qui peuvent être échangés entre les entités de protocole. Ces messages sont construits à partir du contexte de la synthèse.

- spécification des règles contextuelles : une règle contextuelle est spécifiée par une fonction recevant en entrée la liste des sous-réseaux de Petri implantant un objet u (place ou transition) à une étape donnée de la synthèse et délivrant en sortie le résultat de l'application de la règle. Le résultat vaut 1 si la règle a été appliquée avec succès et 0 dans le cas contraire. Cette fonction consiste dans un premier temps à localiser le point de raffinement qui est propre à la règle en question. Un point de raffinement est représenté dans la structure interne d'une spécification IPNL par une transition particulière (resp. une place particulière) dite *méta-transition* (resp. *méta-place*) [2]. Si le point de raffinement recherché est localisé et le prédicat associé à la règle est satisfait alors une séquence de transformations de base (règles de base) est construite et exécutée par la fonction. La séquence de transformations permet, en fait, de synthétiser une fonctionnalité bien précise du protocole. Cette séquence de transformations

1. Une *macro-transition* (ou *macro-place*) est une transition (ou place) particulière qui définit un bloc de réseaux de Petri.

2. Le concept de *méta-transition* (resp. *méta-place*) est défini au chapitre 2.

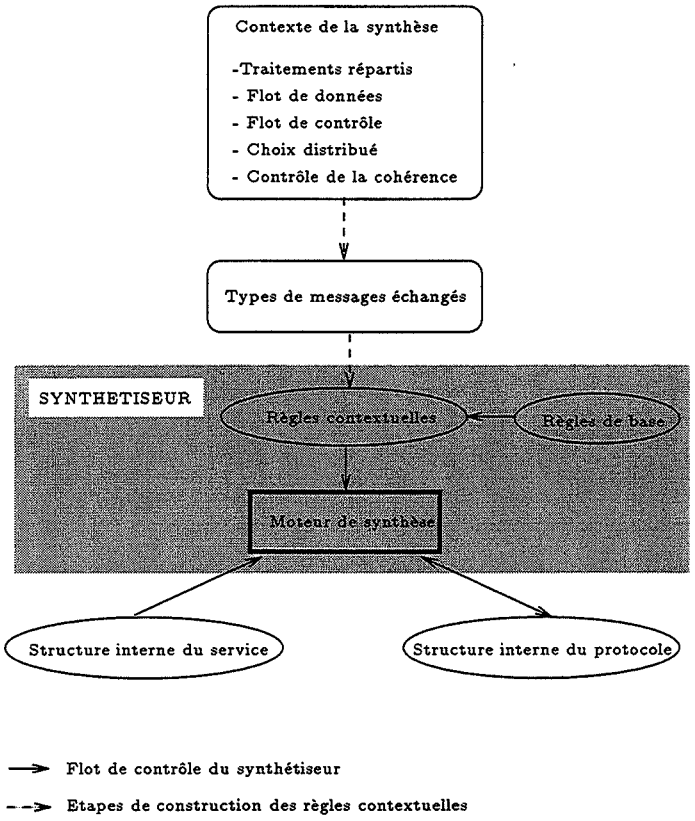

FIG. 4.2 – *Structure du module de synthèse*

peut introduire de nouveaux points de raffinements permettant, ainsi, de synthétiser d'autres fonctionnalités du protocole.

Le moteur de synthèse utilise les règles de synthèse contextuelles afin de construire la structure interne du protocole. Il implante, de ce fait, l'algorithme de synthèse que nous avons présenté au chapitre 2.

4.3 Langage de spécification des services et des protocoles

Afin de permettre la description des entrées et sorties de STEPS, nous avons développé un langage de spécification des services et des protocoles basé sur le modèle IPN. Le langage est appelé IPNL (Interpreted Petri Net Language). Nous présentons dans cette section les éléments de base du langage. Pour la définition des éléments syntaxiques, nous utilisons un style proche de BNF (Backus-Naur Form). Une syntaxe complète du langage IPNL est fournie en annexe B. Les méta-symboles suivants sont utilisés pour l'écriture d'une règle syntaxique :

Symbole non terminal	:	<..>
Symbole terminal	:	"."
Choix	:	..|.. ou {..|..}
Élément optionnel	:	[..]
Une ou plusieurs itérations	:	{..}+
Zéro, une ou plusieurs itérations	:	{..}*

4.3.1 Spécification d'un service

La spécification IPNL d'un service comporte trois parties : une *description d'un environnement*, une *description comportementale* et une *description de l'architecture cible*. La description de l'environnement définit les *types*, les *constantes*, les *variables*, les *ressources*, les *points d'accès au service* et les *interactions*. Les variables et les ressources représentent des données internes au service, alors que les points d'accès et les interactions définissent l'interface du service par rapport à ses utilisateurs. La description comportementale définit un comportement faisant abstraction du caractère réparti de l'architecture cible. Le comportement du service est spécifié à l'aide d'un réseau de Petri interprété. Cette spécification doit respecter les restrictions décrites dans le chapitre 2. La description de l'architecture cible définit un ensemble de sites sur lesquels doit être dérivé le protocole, ainsi qu'une fonction d'allocation des ressources et des points d'accès à ces sites. La structure générale d'une spécification de service est décrite selon la syntaxe suivante :

```
<Spécification de service> ::= "SERVICE" <Identificateur>
                                   <Déclaration des constantes>
                                   <Déclaration des types>
                                   <Déclaration des points d'accès>
                                   <Déclaration des interactions>
                                   <Déclaration des variables>
                                   <Déclaration des ressources>
                                   <Déclaration des transitions>
                                   <Architecture cible>
                               "END."
```

4.3.1.1 Déclaration des constantes, des types et des variables

IPNL permet la déclaration de constantes, de types et de variables. La technique de synthèse proposée dans le chapitre 2 n'impose aucune restriction sur le typage des données. Il n'y a donc pas de limite quant aux types permis par IPNL. La sémantique liée à l'utilisation d'une variable est définie au chapitre 2. La syntaxe associée à la déclaration des constantes, des types et des variables est donnée en annexe B.

4.3.1.2 Déclaration des ressources

Rappelons qu'une ressource définit une encapsulation d'un ensemble d'opérateurs agissant éventuellement sur un état interne de la ressource. La description d'un opérateur consiste à fournir uniquement son entête selon une syntaxe proche du langage Pascal. Un entête peut définir ainsi soit une fonction, soit une procédure. La description de l'entête doit indiquer

aussi s'il s'agit d'un opérateur modifiant ou non l'état interne de la ressource. La syntaxe d'une déclaration de ressource est donnée comme suit :

```
<ressource> ::= "RESOURCE" <Identificateur>
                   {<Opérateur> ";"}+
                "END"
<opérateur> ::= <Fonction> | <Procédure>
<Fonction> ::= ["READ"] "FUNCTION" <Identificateur> "("{<Paramètres>}*")"":"<type>
<Paramètres> ::= {<Identificateur> ","}* <Identificateur> ":" <type>
<Procédure> ::= [<Nature opérateur>] "PROCEDURE" <Identificateur>
                "("<Liste paramètres en entrée>["|"<Liste paramètres en sortie>]")"
<Liste paramètres en entrée> ::= {<Paramètres>}*
<Liste paramètres en sortie> ::= {<Paramètres>}*
<Nature opérateur> ::= "READ" | "WRITE"
```

La description d'une ressource fait abstraction de l'implantation de ses opérateurs. En effet, l'implantation d'un opérateur consiste à définir un traitement local qui peut être isolé de la description d'un service ou d'un protocole. Une fois que le protocole est produit dans un langage d'implantation, le développeur doit fournir le code des opérateurs et effectuer, ainsi, une compilation et/ou une édition des liens avec la description du protocole.
Nous pouvons distinguer une ressource avec état d'une ressource sans état par les attributs associés aux opérateurs. En effet, si une ressource possède au moins un opérateur auquel est associé le mot clé "WRITE", cette ressource dispose forcément d'un état interne variable (que nous qualifions de ressource avec état). Par contre, si une ressource possède uniquement des opérateurs caractérisés par le mot clé "READ", il s'agit dans ce cas d'une ressource à état constant (que nous qualifions de ressource sans état). La description de l'état interne d'une ressource est un détail intrinsèque à la ressource, il est donc omis dans la spécification d'un service ou d'un protocole.
Une ressource peut servir pour modéliser un serveur, une base de donnée, ou une application quelconque. L'exemple suivant spécifie une ressource avec état `R1` qui représente une base de données sur laquelle on peut invoquer deux opérateurs: une procédure `append` qui consiste à ajouter une valeur `x` dans la base de données, et une fonction `size` qui délivre la taille de la base de données.

Exemple :

```
RESOURCE R1
   WRITE PROCEDURE append(x:INTEGER);
   FUNCTION size():INTEGER;
END
```

4.3.1.3 Déclaration des points d'accès au service et des interactions

Les points d'accès et les interactions d'un service définissent son interface par rapport à ses utilisateurs. A chaque point d'accès sont associées les interactions qui peuvent être initiées par l'utilisateur du service ou par le fournisseur du service. Une interaction est définie par un nom et un ensemble de paramètres typés.

```
<Déclaration points d'accès> ::= "SAP" {<Identificateur>","}* <Identificateur>";"
```

```
<Déclaration des interactions> ::= "INTERACTION"
                                     {BY <Liste saps> ":" {<Interaction> ";"}+}+
                                   "END"
<Liste saps>   ::= {<Identificateur> ","}* <Identificateur>
<Interaction> ::= <Identificateur> ["("{<Paramètres> ";"}* <Paramètres>")"]
<Paramètres>   ::= {<Identificateur> ","}* <Identificateur> ":" <type>
```

Exemple : Cet exemple décrit trois points d'accès a, b et c, et deux interactions `GetValue` et `SizeIndication`. La première interaction peut être initiée à partir des points d'accès a et b, alors que la seconde ne peut être invoquée qu'à travers le point c.

```
SAP a, b, c;
INTERACTION
   BY a,b:
      GetValue(x:INTEGER);
   BY c:
      SizeIndication(x,y:INTEGER);
END
```

4.3.1.4 Description comportementale

La description comportementale spécifie un comportement effectif du service au moyen d'un réseau de Petri interprété. Chaque transition du réseau interprète une primitive de service[3]. Le réseau global définit donc toutes les séquences valides des primitives de service. La description d'une transition de service est définie par sept clauses comme indiqué dans la syntaxe ci-dessous.

```
<Transition de service> ::= "TRANS" <Identificateur>
                                [<Clause from>]
                                [<Clause to>]
                                 <Clause sap>
                                [<Clause when>]
                                [<Clause provided>]
                                [<Clause do>]
                                [<Clause output>]
```

Les clauses "from" et "to" définissent respectivement les places en entrée et celles en sortie de la transition décrite. Le réseau de Petri étant ordinaire[4] la valuation des arcs est prise par défaut à 1. La clause "sap" définit le point d'accès de la transition de service. Nous allons voir un peu plus loin que le point d'accès d'une transition permet de déterminer le site responsable de l'exécution de cette transition. La clause "sap" est de ce fait obligatoire. Ceci a pour conséquence que le placement des transitions aux différents sites de l'architecture cible est à l'initiative du spécifieur. Ce choix permet d'éviter les problèmes de placement qui ne sont pas l'objet de notre étude.

```
<Clause from> ::= "FROM" {<place> ","}* <place>
<Clause to> ::= "TO" {<place> ","}* <place>
<place> ::= <Identificateur>
<Clause sap> ::= "SAP" <Identificateur>
```

3. Dans le langage IPNL le concept de primitive de service est confondu avec le concept de transition.
4. Le caractère ordinaire découle des restrictions R1 à R4 posées dans le chapitre 2.

La clause "when" et la clause "provided" expriment les contraintes à satisfaire pour le franchissement de la transition. Alors que les clauses "do" et "output" décrivent l'effet du franchissement de la transition sur l'environnement interne (i.e., les variables et les ressources) et externe (i.e., les utilisateurs du service). La clause "when" traduit une attente d'une interaction à travers le point d'accès de la transition de service. Cette clause est satisfaite si l'interaction attendue est disponible dans la file de réception correspondant au point d'accès de la transition. Cela suppose qu'à chaque point d'accès au service est associée une file de réception FIFO (First in First Out), et qu'à chaque utilisateur attaché à un point d'accès est associée également une file de réception des interactions provenant de la part du fournisseur du service.

```
<Clause when> ::= "WHEN" <nom d'interaction> "(" {<Argument>","}* <Argument> ")"
<Argument> ::= <Référence variable> | <Constante>
```

L'interaction spécifiée dans la clause "when" doit correspondre à l'une des interactions déclarées par le mot clé "INTERACTION". Si un argument de l'interaction est une constante, l'élément correspondant dans l'interaction attendue doit être égal à cette constante. Par contre, si un argument est une référence à une variable, cette variable est utilisée pour conserver la valeur de l'élément correspondant dans l'interaction attendue.

La clause "provided" exprime un prédicat sur le franchissement de la transition. Elle est satisfaite si l'évaluation du prédicat fournit la valeur "vrai".

```
<Clause provided> ::= "PROVIDED" <Expression logique>
<Expression logique> ::= <Expression logique> <Opérateur logique>
                         <Expression logique>
                         | <Expression de comparaison>
                         | "(" <Expression logique> ")"
                         | "NOT" <Expression logique>
                         | <Terme>
<Expression de comparaison> ::= <Terme> <Opérateur de comparaison> <Terme>
<Opérateur de comparaison> ::= "<" | ">" | "<=" | "<=" | "!=" | "="
<Opérateur logique> ::= "AND" | "OR"
```

Les *termes* constituent les éléments de base utilisés dans une transition pour accéder à une donnée. Un terme peut donc définir soit une référence à une variable, soit une constante, soit une fonction. Une fonction représente un terme composé, ses arguments sont aussi des termes.

```
<Terme> ::= <Référence variable> | <Constante> | <Fonction d'une ressource>
<Référence variable> ::= <Identificateur>
<Fonction d'une ressource> ::= <Identificateur de ressource> "."
                               <Identificateur de fonction>"("[<Arguments>]")"
<Identificateur de ressource> ::= <Identificateur>
<Identificateur de fonction> ::= <Identificateur>
<Arguments> ::= {<Terme>","}* <Terme>
```

Exemple : La transition `t1` est associée au point d'accès `a`. Elle est franchissable si la place `p1` est marquée (i.e., le système spécifié se trouve dans l'état `p1`), l'interaction `Requete` est

disponible dans la file de réception associée au point d'accès a et le prédicat défini par la clause "provided" est satisfait.

```
TRANS t1
  FROM p1 TO p2
  SAP a
  WHEN Requete
  PROVIDED NbRequetes<MaxRequetes
  ...
```

La clause "do" spécifie un appel de procédure lié à l'une des ressources déclarées dans le service. Si la ressource est locale à la transition de service, c'est-à-dire que la ressource et le point d'accès de la transition sont alloués à un même site, il s'agit d'un appel local, sinon il s'agit d'un appel distant. Au niveau service, il n'y a pas de distinctions entre ces deux types d'appels.

```
<Clause do> ::= "DO" <Identificateur de ressource>"."<Identificateur de procédure>
                  "(" [{<Argument d'entrée>","}* <Argument d'entrée>]
                      ["|" {<Argument de sortie>","}* <Argument de sortie>] ")"
<Argument d'entrée> ::= <Terme>
<Argument de sortie> ::= <Référence variable>
<Identificateur de ressource> ::= <Identificateur>
<Identificateur de procédure> ::= <Identificateur>
```

La clause "output" spécifie une émission d'une interaction à travers le point d'accès de la transition. L'interaction émise est stockée dans la file de réception associée à l'utilisateur attaché au point d'accès de la transition.

```
<Clause output> ::= "OUTPUT" <Identificateur> ["("{<Terme>","}* <Terme>")"]
```

Exemple : La transition `t2` est tirée au point d'accès `c`. Elle spécifie que si les places `p2` et `p3` sont marquées (on dit que le système décrit est à l'état `p2,p3`), l'appel de procédure `append(x)` est initié à partir du point d'accès `c` sur la ressource `R1` et l'interaction `SizeIndication` est délivrée à l'utilisateur attaché au point `c`. Cette interaction utilise deux arguments dont les valeurs doivent être récupérées à partir des ressources `R1` et `R2`.

```
TRANS t2
   FROM p2, p3 TO p0, p5
   SAP c
   DO R1.append(x)
   OUTPUT SizeIndication(R1.size(), R2.size())
```

4.3.1.5 Description de l'architecture cible

La description de l'architecture cible permet de définir un ensemble de sites sur lesquels doit être implanté le service. Pour chaque site sont spécifiés les points d'accès et les ressources qui lui sont alloués.

```
<Architecture cible> ::= "TARGET"
                            {<Site>";"}*
                         "END"
```

```
<Site> ::= "SITE" <Identificateur>
              "SAP" {<Identificateur de sap>","}* <Identificateur de sap> ";"
              "RESOURCE" {<Identificateur de ressource>","}*
                          <Identificateur de ressource> ";"
           "END"
<Identificateur de sap> ::= <Identificateur>
<Identificateur de ressource> ::= <Identificateur>
```

Exemple : Cet exemple décrit une architecture définie par deux sites S1 et S2. Les points d'accès a et b sont respectivement associés aux sites S1 et S2. La ressource R1 est allouée au site S1 et la ressource R2 est dupliquée sur les deux sites.

```
TARGET
   SITE s1
     SAP a;
     RESOURCE R1, R2;
   END
   SITE s2
     SAP b;
     RESOURCE R2;
   END
END
```

4.3.2 Exemple de service

Reprenons l'exemple du service de saisie présenté au chapitre 2. La description IPNL de ce service est donnée comme suit :

```
SERVICE Data_acquisition_service
 SAP a,b,c;
 INTERACTION
    BY a,b : GetValue(x:INTEGER);
    BY c : SizeIndication(x:INTEGER; y:INTEGER);
 END;
 VAR x,y : INTEGER;
 RESOURCE R1
    FUNCTION size():INTEGER;
    WRITE PROCEDURE append(x:INTEGER);
 END;
 RESOURCE R2
   FUNCTION size():INTEGER;
   WRITE PROCEDURE append(x:INTEGER)
 END;
 TRANS t1
   FROM etat0 TO etat1 SAP a
   WHEN GetValue(x)
 TRANS t2
   FROM etat1 TO etat2 SAP a
   PROVIDED R1.size()>R2.size()
   DO  R2.append(x)
 TRANS t3
```

```
    FROM etat1 TO etat2 SAP a
    PROVIDED R1.size()<=R2.size()
    DO R1.append(x)
  TRANS t4
    FROM etat2,etat3 TO etat0,etat5 SAP c
    OUTPUT SizeIndication(R1.size(), R2.size())
  TRANS t5
    FROM etat4 TO etat3 SAP b
    WHEN GetValue(y)
    DO R1.append(y)
  TRANS t6
    FROM etat5 TO etat4 SAP b
  INIT etat0,etat5;
  TARGET
    SITE S1
       SAP a;
       RESOURCE R1;
    END;
    SITE S2
       SAP b;
       RESOURCE R1, R2;
    END;
    SITE S3
       SAP c;
       RESOURCE R1, R2;
   END;
  END
END.
```

4.3.3 Spécification d'un protocole

Une description IPNL d'un protocole est constituée de l'ensemble des descriptions des entités de protocole qui le composent. La description d'une entité de protocole est basée sur le même modèle qu'une description de service, à savoir le modèle IPN. Néanmoins, quelques différences existent entre les deux. Elles sont développées dans cette section. Une description d'une entité de protocole comporte deux parties : une description d'un environnement et une description d'un comportement.

4.3.3.1 Description de l'environnement

La description de l'environnement inclut en plus de la description des constantes, des types, des variables, des ressources, des points d'accès et des interactions, la description des *accointances*. Les accointances d'une entité de protocole définissent les entités avec lesquelles elle peut communiquer.

```
<Description des accointances> ::= "AQUAINTANCE"
                                    {<Identificateur>","}* <Identificateur> ";"
```

Une entité de protocole utilise, en plus des interactions définies dans la description de service, d'autres type d'interactions, à savoir les *messages* qu'elle peut échanger avec ses accointances.

4.3.3.2 Description comportementale

La description comportementale est définie par un réseau de Petri interprété. Ce dernier est dérivé, en une ou plusieurs étapes de raffinement, du réseau sous-jacent à la description du service. Il est, de ce fait, constitué de deux types de transitions. Les *transitions simples* et les *macro-transitions*. Une transition simple interprète soit une primitive de service avec événement externe, soit une primitive de service avec événement interne, soit une primitive de service avec les deux types d'événements. Le dernier cas permet de spécifier, entre autres, une transition qui, à la réception d'un message initie une interaction avec un utilisateur. Par ailleurs, une macro-transition correspond à un raffinement d'une transition simple. Elle est décrite au moyen d'un bloc de réseaux de Petri. Ce bloc comporte une description de constantes, de variables et de transitions. Afin d'assurer la correction des protocoles construits, une forme pré-définie est requise pour un bloc substituable à une transition (cf. chapitre 2).

```
<Transition de protocole> ::= <Transition simple> | <Macro-transition>
<Transition simple> ::= "TRANS" <Identificateur>
                            [<Clause from>]
                            [<Clause to>]
                            [<Clause when> {"ON" <sap>|"FROM" <Accointance>}]
                            [<Clause provided>]
                            [<Clause do>]
                            [<Clause output> {"ON" <sap>|"TO" <Accointance>}]
<sap> ::= <Identificateur>
<Accointance> ::= <Identificateur>
<Macro-transition>  ::= "MACROTRANS" <Identificateur>
                            [<Clause from>]
                            [<Clause to>]
                            <Bloc>
<Bloc> ::= "BLOCK" <Identificateur> "FOR" <Identificateur de macro-transition>
                            [<Déclaration des constantes>]
                            [<Déclaration des variables>]
                            {<Transition de protocole>}*
           "END"
```

Le mot clé "ON" permet de spécifier le point d'accès sur lequel porte la clause "when" ou la clause "output". Les mots clés "FROM" et "TO" permettent de spécifier respectivement l'entité de protocole émettrice et réceptrice d'un message. Des exemples de description de protocoles sont données en annexe C. Ils correspondent aux résultats des expérimentations que nous présentons dans la section suivante.

4.4 Expérimentations

Nous présentons dans cette section des expérimentations mettant en évidence les fonctionnalités principales de STEPS. Nous terminons par un exemple d'application récapitulatif.

4.4.1 Synthèse de traitements répartis

Rappelons qu'une transition de service peut spécifier dans sa clause "do" une action liée à une ressource donnée. Cette action peut engendrer au niveau protocole un traitement

réparti. En effet, un appel d'une opération liée à une ressource distante engendre un protocole d'appel de procédure distante (RPC). Une ressource est considérée comme distante par rapport à une transition de service dans le cas où elle n'est pas allouée au site qui contient le point d'accès de cette transition. D'autre part, un appel d'une opération liée à une ressource qui est à la fois distante et dupliquée engendre un RPC avec un mécanisme de mise à jour des copies locales. Par conséquent, la dérivation des traitements répartis est très liée à la configuration définie par l'architecture cible. La synthèse des traitements répartis est réalisée par les règles RS2, RS3 et RS4. Nous présentons dans ce qui suit un exemple simple illustrant un appel d'une opération liée à une ressource distante et dupliquée.

```
SERVICE RPC_avec_MaJ
 SAP a;
 VAR x,y,z,t : INTEGER;
 RESOURCE Serveur
   WRITE PROCEDURE P(x:INTEGER; y:INTEGER | z:INTEGER)
 END;
 TRANS t
   FROM etat TO etat
   SAP a
   DO Serveur.P(x,y|z)
 INIT etat;
 TARGET
   SITE S1
      SAP a;
   END;
   SITE S2
      RESOURCE Serveur;
   END;
   SITE S3
      RESOURCE Serveur;
   END
 END
END.
```

Le service RPC_avec_MaJ spécifie un serveur avec état sur lequel on peut invoquer la procédure P. Le serveur est dupliqué sur deux sites S2 et S3. L'invocation du serveur s'effectue à partir du site S1. Le protocole dérivé de cette spécification est donné en annexe C.1.

4.4.2 Synthèse du flot de contrôle

Le flot de contrôle synthétisé au niveau protocole permet d'assurer l'ordonnancement temporel défini entre les transitions de service. Dans sa version actuelle, STEPS considère un flot de contrôle sans structures sélectives distribuées. L'extension de STEPS pour la synthèse des structures sélectives distribuées consisterait simplement à étendre le module des règles contextuelles par l'écriture des fonctions implantant les règles RS11 à RS14. L'objectif de la première version de STEPS n'est pas d'implanter toutes les fonctionnalités définies au chapitre 2 mais de montrer, à l'aide d'une version minimale, la mise en pratique de notre démarche.
Nous montrons dans ce qui suit à l'aide d'un exemple simple la synthèse du flot de contrôle.

Nous considérons un exemple d'ordonnancement de 5 tâches. Chaque tâche étant initiée à partir d'un site bien précis.

```
SERVICE Ordonnancement_de_taches
  SAP a,b,c;
  TRANS tache1
   FROM etat1 TO etat2, etat3
   SAP a
  TRANS tache2
   FROM  etat2 TO etat4
   SAP b
  TRANS tache3
   FROM etat2 TO etat4
   SAP b
  TRANS tache4
   FROM etat3 TO etat5
   SAP c
  TRANS tache5
   FROM etat4, etat5 TO etat1
   SAP c
  INIT etat1;
  TARGET
    SITE S1
      SAP a;
   END;
   SITE S2
      SAP b;
   END;
   SITE S3
      SAP c;
   END
 END
END.
```

Le protocole dérivé du service `Ordonnancement_de_taches` est décrit en annexe C.2. Ce protocole décrit un ordonnancement de tâches réparti.

4.4.3 Synthèse du flot de données

Rappelons que le flot de données est défini par les messages de données qui circulent entre les entités de protocole afin de permettre l'exécution des transitions de service. Il existe deux manières de spécifier le flot de données au niveau service : soit à l'aide de variables soit à l'aide de ressources.

4.4.3.1 Spécification du flot de données à l'aide de variables

Une variable dont la valeur est produite par une transition de service donnée peut être utilisée par une autre transition de service qui lui succède. Ceci définit un flot de données entre les deux transitions. Ce flot de données est synthétisé au niveau protocole au moyen de la règle RS6. Considérons un exemple spécifiant une demande d'une connexion de transport

émanant d'un point d'accès a, suivie par une indication de la demande au point d'accès b.

```
SERVICE Connexion_transport
 SAP a,b;
 INTERACTION
   BY a : TConnectRequest(qos:INTEGER);
   BY b : TConnectIndication(qos:INTEGER);
 END;
 VAR qos: INTEGER;
 TRANS requete
   FROM etat1 TO etat2 SAP a
   WHEN TConnectRequest(qos)
 TRANS indication
   FROM etat2 to etat1 SAP b
   OUTPUT TConnectIndication(qos)
 INIT etat1;
 TARGET
  SITE S1
   SAP a;
  END;
  SITE S2
   SAP b;
  END;
 END
END.
```

Le protocole produit par STEPS à partir de cet exemple est décrit en annexe C.3. Il utilise deux messages : un message de données transférant les paramètres d'une demande de connexion et un message de synchronisation transférant le contrôle au demandeur de la connexion.

4.4.3.2 Spécification du flot de données à l'aide de ressources

Une transition de service peut spécifier des termes en entrée liés à des ressources distantes. Les valeurs de ces termes doivent être collectées afin que la transition puisse s'exécuter. Un algorithme de synthèse du flot de données (algorithme SFD) a été présenté, à cet égard, au chapitre 2. Cet algorithme construit pour une transition de service donnée un ensemble de blocs de réseaux de Petri qui coopèrent afin de collecter toutes les valeurs des termes attendues en entrée de cette transition. Les termes traités par cet algorithme sont liés à des ressources. L'algorithme SFD est utilisé par toute règle contextuelle qui a besoin de collecter des données liées à des ressources distantes, c'est-à-dire, les règles RS5, RS7 et RS9. Nous présentons ci-dessous un exemple simple de service spécifiant une transition qui requiert des données en entrée liées à des ressources distantes.

```
SERVICE Flot_de_donnees
 SAP a;
 VAR y: INTEGER;
 RESOURCE R1
    FUNCTION f():INTEGER;
    FUNCTION g(x:INTEGER):INTEGER;
```

```
 END;
 RESOURCE R2
    FUNCTION h():INTEGER;
 END;
 RESOURCE R3
    READ PROCEDURE p(|y:INTEGER);
 END;
 TRANS t1
   FROM etat1 TO etat2 SAP a
 TRANS t2
   FROM etat2 TO etat1 SAP a
   PROVIDED R1.f()>=R1.g(R2.h())
   DO R3.p(|y)
 INIT etat1;
 TARGET
   SITE S1
      SAP a;
      RESOURCE R3;
   END;
   SITE S2
      RESOURCE R1;
   END;
   SITE S3
      RESOURCE R2;
   END;
 END
END.
```

La transition `t1` requiert en entrée les valeurs des termes : `R1.f()`, `R2.h()` et `R1.g(R2.h())`. Ces valeurs sont utilisées pour évaluer le prédicat de la transition. Les ressources `R1` et `R2` sont distantes par rapport à `t1`. Le protocole dérivé du service `flot_de_donnees` (cf. annexe C.4) met en oeuvre des échanges de messages de données entre les trois entités qui le composent afin de permettre l'exécution de `t2` dans le site `S1`. Il utilise pour ce faire 6 messages.

4.4.4 Synthèse du contrôle de la cohérence

Une description d'un service pour une architecture répartie peut introduire des accès concurrents à des ressources partagées et éventuellement dupliquées. Un mécanisme de contrôle de la cohérence des ressources est, par conséquent, synthétisé au niveau protocole afin de garantir les contraintes de cohérence C1 et C2 (cf. chapitre 2). La synthèse automatique du contrôle de la cohérence permet au concepteur d'un service de se concentrer uniquement sur le problème qu'il veut spécifier et ainsi faire abstraction des problèmes de cohérence engendrés par la nature de l'architecture cible. L'exemple ci-dessous spécifie deux transitions accédant en concurrence à une base de donnée `R1` dupliquée sur deux sites `S1` et `S2`.

```
SERVICE Acces_concurrents
 SAP a,b;
 INTERACTION
   BY a: GetValue(x:INTEGER);
```

```
   BY b: SizeIndication(x:INTEGER);
         GetSize;
 END;
 VAR x: INTEGER;
 RESOURCE R1
   FUNCTION Size():INTEGER;
   WRITE PROCEDURE Append(x:INTEGER);
 END;
 TRANS t1
  FROM etat1 TO etat2 SAP a
  WHEN GetValue(x)
 TRANS t2
  FROM etat2 TO etat1 SAP a
  DO R1.Append(x)
  OUTPUT SizeIndication(R1.Size())
 TRANS t3
  FROM etat3 TO etat4 SAP b
  WHEN GetSize
 TRANS t4
  FROM etat4 TO etat3 SAP b
  OUTPUT SizeIndication(R1.Size())
 INIT etat1, etat2;
 TARGET
  SITE S1
    SAP a;
    RESOURCE R1;
  END;
  SITE S2
    SAP b;
    RESOURCE R1;
  END;
 END
END.
```

La transition `t1` lit une valeur `x` à travers le point d'accès `a`, elle est suivie par `t2` qui conserve systématiquement cette valeur dans `R1` et délivre la taille de `R1` à travers le point `a`. Parallèlement, la transition `t3` reçoit une requête `GetSize` à travers le point d'accès `b`. Elle est suivie par `t4` qui délivre la taille de `R1` à travers le point `b`. Le protocole produit par STEPS à partir de ces spécifications (cf. annexe C.5) permet de réaliser ce service tout en garantissant qu'à tout instant les tailles délivrées à travers les points d'accès `a` et `b` sont identiques.

Dans les spécifications du protocole, seules trois formes de transitions relatives au contrôle de la cohérence sont construites : les transitions de demande d'entrée en section critique, les transitions d'attente de l'autorisation d'entrée et les transitions de sortie de la section critique. Ces transitions doivent être raffinées encore afin de préciser le mécanisme utilisé dans le contrôle de la cohérence. Le mécanisme adopté est basé sur l'algorithme de LAMPORT [5].

5. Ce mécanisme n'a pas été inclus dans les spécifications dérivées afin de ne pas encombrer les spécifications du protocole. Pour plus de détails concernant ce mécanisme, se reporter au chapitre 2.

4.4.5 Application : Saisie de données dans un environnement réparti

Reprenons l'exemple spécifié dans la section 4.3.2. Le protocole synthétisé par STEPS est donné en annexe D.

4.5 Caractéristiques de STEPS

STEPS se distingue des autres outils de développement de protocoles par l'ensemble des caractéristiques suivantes :

1. Intégration de la synthèse et la vérification.
2. Un "bon" compromis entre le pouvoir de synthèse et le pouvoir d'expression.
3. Capacité de produire des protocoles à plusieurs entités.
4. Extensibilité du pouvoir de synthèse
5. Capacité de produire des spécifications de protocoles dans une norme standard.

Nous développons dans ce qui suit chacune de ces caractéristiques.

4.5.1 Intégration de la synthèse et la vérification

La phase de vérification permet de garantir la correction logique de la spécification du service. Par ailleurs, lors du processus de synthèse, la correction logique et sémantique des spécifications produites est assurée de manière incrémentale. Par conséquent, l'intégration de la vérification et la synthèse dans STEPS a pour but de garantir la correction logique et sémantique des spécifications finales à savoir les spécifications du protocole construit.

4.5.2 Un "bon" compromis entre le pouvoir d'expression et le pouvoir de synthèse

L'une des difficultés qui touchent les techniques de synthèse de protocoles réside dans l'opposition du pouvoir d'expression des services et le pouvoir de synthèse. En effet, les modèles d'expression devraient être assez riches et expressifs afin de prendre en compte toutes les fonctionnalités essentielles d'un service. Alors que les techniques de synthèse, de manière analogue aux techniques de preuve, exigent l'utilisation d'un modèle mathématique relativement simple (automates à états finis, réseaux de Petri non interprétés, etc.) afin de conduire à des techniques de synthèse automatisables et efficaces. En somme, plus le modèle est riche, plus il est difficile de synthétiser. C'est ainsi que la plupart des techniques existantes font des restrictions limitatives sur le modèle d'expression des services. La technique de synthèse que nous avons présentée au chapitre 2 constitue un "bon" compromis entre le pouvoir de synthèse et le pouvoir d'expression relativement aux techniques existantes. En effet, STEPS permet de synthétiser automatiquement au niveau protocole l'ensemble des fonctionnalités suivantes :

- traitements répartis,
- flot de contrôle,

- flot de données,
- contrôle de cohérence des données,
- choix distribué.

4.5.3 Extensibilité de la capacité de synthèse

L'ensemble des outils STEPS est facilement extensible. En effet, notre technique de synthèse étant basée sur un ensemble de règles de raffinement de réseaux de Petri, l'extension du modèle d'expression par de nouvelles fonctionnalités se traduit par une extension de l'ensemble des règles de raffinements sans remettre en cause celles qui existent déjà. Une telle extension a été démontrée au chapitre 2 sur le traitement des structures sélectives distribuées.

4.5.4 Capacité de produire des protocoles à plusieurs entités

Contrairement à certains outils existants tels que SEPS, KSPS [SHI 91] et APS [RAM 85], STEPS a la capacité de synthétiser des protocoles à plus de deux entités. Ceci est rendu possible en intégrant dans le problème de synthèse la spécification de l'architecture cible sur laquelle on souhaite implanter le protocole.

4.5.5 Capacité de produire des spécifications dans une norme standard

STEPS permet non seulement de produire du code dans son langage interne IPNL mais aussi dans un langage standard de description des protocoles. Nous avons choisi pour cela le langage Estelle car il nous semble le plus proche de IPNL. Nous montrons au chapitre 5 les techniques utilisées pour générer des spécifications Estelle.

4.5.6 Autres caractéristiques

STEPS a été implanté sur une station de travail SUN SPARC-5 utilisant le langage C sous le système SunOS 5.5. La taille du logiciel est d'environ 8700 lignes. Les modules C de STEPS sont structurés selon le schéma fonctionnel donné par la Figure 4.1. Cette structuration modulaire lui assure une maintenance facile.

4.6 Améliorations possibles de STEPS

Nous décrivons dans cette section les extensions possibles que nous pouvons apporter à notre approche de synthèse et ainsi aux outils STEPS.

4.6.1 Synthèse de fonctionnalités intrinsèques au niveau protocole

Certaines fonctionnalités protocolaires telles que : le recouvrement d'erreurs de transmission, le contrôle du flux de données et le séquencement des messages, ne peuvent pas être spécifiées explicitement au niveau service. Ils sont intrinsèques au niveau protocole. Par conséquent, ils ne peuvent pas être synthétisées automatiquement par STEPS. Ceci requiert donc l'extension de STEPS afin de prendre en compte ce type de fonctionnalités.

L'idée que nous proposons consiste à définir *deux niveaux de conception.* Le premier niveau fait abstraction des fonctionnalités protocolaires qui ne peuvent pas être spécifiées par un service. Ainsi, pour s'abstraire du recouvrement d'erreurs, il suffit de faire l'hypothèse d'un médium fiable. Le premier niveau correspond, en fait, à notre stratégie de synthèse. Le deuxième niveau de conception reprend les spécifications produites par le premier et consiste à les raffiner encore afin de réaliser les fonctionnalités requises. Deux approches sont possibles pour la mise en oeuvre du deuxième niveau.

Approche interactive Cette approche consiste à donner la liberté au concepteur pour raffiner de manière interactive les spécifications produites par le premier niveau. Nous proposons pour ce faire l'utilisation des règles de *tricotage* de réseaux de Petri (*knitting rules*) [CHA 94a]. De cette manière, le concepteur peut intégrer de nouvelles fonctionnalités en ajoutant des chemins de type PP ou TT [6] sans remettre en cause la correction du protocole.

Approche par boîte à outils Cette approche va dans le même sens que notre stratégie de synthèse. En effet, elle consiste à construire une boîte à outils définissant pour chaque fonctionnalité protocolaire un ensemble de règles contextuelles. Par conséquent, partant d'une spécification de protocole ne supportant pas une fonctionnalité donnée, il serait possible de synthétiser automatiquement un protocole réalisant cette fonctionnalité en appliquant les règles contextuelles qui correspondent à cette nouvelle fonctionnalité.

L'avantage de la première approche est qu'elle donne plus de liberté au concepteur sans compromettre la correction du protocole, alors que la deuxième approche reste toujours limitée à un ensemble donné de fonctionnalités. Ces deux approches ne sont pas exclusives et peuvent donc être combinées.

4.6.2 Interface graphique

STEPS utilise une interface utilisateur du style ligne de commande. Pour bénéficier de l'aspect graphique du modèle IPN, il est intéressant d'intégrer à STEPS une interface utilisateur graphique. Nous pouvons définir une interface graphique propre à notre formalisme en utilisant, par exemple, l'interface multi-formalisme MACAO ; cet interface fait partie d'un atelier de modélisation interactif appelé AMI [BER 89]. Nous avons déjà fait usage de cette interface en créant un formalisme graphique pour la spécification des systèmes distribués symétriques et hiérarchiques [KAH 95].

4.6.3 Contraintes temporelles

STEPS ne considère pas les contraintes temporelles dans les spécifications de services et de protocoles. Il peut, néanmoins, être étendu à la synthèse de protocoles et applications réparties temps réel. Pour ce faire, il faut étendre le modèle sous-jacent IPN par les contraintes de temps et définir une stratégie pour synthétiser ces contraintes au niveau protocole. De nombreux travaux se font dans cette direction [KAP 91a, KHO 94a, NAT 95] mais aucun n'utilise le modèle des réseaux de Petri.

6. Un chemin de type PP (resp. TT) est un chemin reliant deux places (resp. deux transitions).

4.7 Conclusion

Nous avons présenté dans ce chapitre une réalisation de notre approche de synthèse, à savoir un ensemble d'outils logiciels pour la synthèse automatique de protocoles [KAH 97c]. Cette réalisation a été expérimentée sur une série d'exemples. Ceci démontre donc la faisabilité de notre démarche.

Chapitre 5

Synthèse de protocoles dans le langage Estelle

Résumé

Nous proposons dans ce chapitre une extension de la méthode de synthèse de protocoles présentée au chapitre 2 pour la dérivation de spécifications de protocoles dans le langage Estelle [KAH 97b]. L'intérêt est que ces spécifications peuvent être directement exploitées par des outils logiciels existants afin de produire du code exécutable ou effectuer des évaluations de performances.

5.1 Introduction

Nous avons présenté au chapitre 2 un algorithme de synthèse de protocoles à partir d'une classe importante de services permettant d'exploiter efficacement un système distribué à un niveau d'abstraction élevé. En effet, cette classe permet d'exprimer : (a) des événements parallèles de façon explicite, puisque le parallélisme est inhérent aux architectures distribuées, (b) des primitives de service paramétrées, (c) des variables virtuellement globales afin de fournir un niveau d'abstraction élevé dans la description des flots de données, (d) des accès concurrents aux variables globales, (e) la duplication de données pour augmenter le degré de disponibilité des données, et (f) des structures sélectives distribuées afin de pouvoir spécifier des choix déterministes ou non déterministes entre plusieurs primitives de service en faisant abstraction de leur localisation. Par conséquent, l'algorithme que nous avons présenté considère, dans les spécifications de protocole synthétisées, simultanément : (1) le flot de contrôle avec prise en compte de la concurrence et du choix distribué, et (2) le flot de données avec prise en compte du contrôle de la cohérence.

Dans le contexte de la synthèse orientée service, plusieurs méthodes ont été proposées dans la littérature (cf. chapitre 1). Nous distinguons les approches basées sur LOTOS [BOC 86] [GOT 90a] [KAN 93a] [KHO 95], les approches basées sur les automates à états finis [CHU 88] [HIG 93] [SAL 90] et les approches basées sur les réseaux de Petri [KAH 96] [YAM 95]. Seules les approches basées sur LOTOS ont la possibilité de produire des protocoles dans un langage de spécification standard (à savoir LOTOS). Néanmoins, toutes ces méthodes ne considèrent pas un ou plusieurs éléments de la classe de services citée plus

haut[1]. De plus, certaines méthodes telles que [KHO 95] et [NAT 95] produisent des spécifications qui ne sont pas directement exploitables car elles introduisent de nouveaux concepts qui n'existent pas dans la version standard du langage[2].
Pour aboutir à une approche qui, à la fois, prend en compte la classe des services désirables (a)-(f), et produit des spécifications dans un langage standard, nous proposons, dans ce chapitre une extension de [KAH 96] pour la synthèse de spécifications dans le langage Estelle [ISO 9074]. L'avantage est que ces spécifications peuvent être directement exploitées par des outils existants[3], tels que EDT[BUD 92], afin de générer automatiquement du code exécutable ou d'effectuer des simulations et des évaluations de performances.

Estelle constitue l'une des techniques de description formelle (FDTs) conçue par l'Organisation Internationale de Standardisation (ISO) pour spécifier formellement les protocoles de communication et les systèmes distribués. Une spécification Estelle est composée de plusieurs *modules* pouvant communiquer et évoluer en parallèle. La communication se fait à travers des *points d'interaction.* Chaque point d'interaction est typé par un *canal* spécifiant les messages qui peuvent y être échangés. Les modules Estelle sont des entités génériques, ils doivent par conséquent être instanciés et ensuite connectés afin de pouvoir communiquer. Le comportement d'un module est défini par un automate à états finis étendu par les types de données du langage Pascal. Pour une présentation plus détaillée du langage Estelle se reporter à [BUD 87][ISO 9074].
Dans ce chapitre, nous présentons les techniques utilisées pour construire une spécification Estelle à partir de la spécification IPN d'un protocole. La démarche s'articule principalement autour de deux étapes. La première étape consiste à établir une relation de correspondance entre les objets du modèle IPN et les objets du langage Estelle. La deuxième étape consiste à produire, selon une approche "top-down", les spécifications du protocole dans le langage Estelle.

Dans la section 5.2, nous proposons une relation de correspondance entre le modèle IPN et le langage Estelle. Dans les sections 5.3 et 5.4, nous développons les techniques utilisées pour dériver des spécifications Estelle. Nous terminons par un exemple illustratif.

5.2 Relation de correspondance IPN-Estelle

La définition d'une relation de correspondance entre les objets IPN et les objets Estelle est facilitée par : (a) l'utilisation d'une architecture ISO pour les protocoles, et (b) l'utilisation d'un modèle de réseaux de Petri ayant une interprétation qui le rend très proche du langage Estelle. La relation de correspondance que nous avons adoptée se résume comme suit :

1. la spécification du protocole dans sa globalité correspond à un module racine *non attribué.*

1. Les limites des méthodes existantes sont détaillées au chapitre 1.
2. Un concept de variable global est introduit dans [KHO 95], et un concept de temps est introduit dans [NAT 95].
3. Une liste assez détaillée sur les outils logiciels existants autour d'Estelle et LOTOS est donnée dans [CHA 93].

2. Une entité de protocole dans le modèle IPN correspond à un module Estelle attribué *systemprocess* ou *systemactivity.*

3. Une entité "utilisateur" correspond à un module Estelle attribué *systemprocess* ou *systemactivity.*

4. Le médium de communication reliant les entités de protocole correspond à un module Estelle attribué *systemprocess* ou *systemactivity.*

5. Un point d'accès au service du modèle IPN correspond à un point d'interaction Estelle.

6. Une transition du modèle IPN correspond à une transition Estelle.

7. Chaque variable locale à une entité de protocole correspond à une variable Estelle locale au module associé à cette entité.

8. Chaque ressource utilisée par une entité de protocole correspond à une variable Estelle et un ensemble de procédures et fonctions opérant sur cette variable.

La non attribution du module racine et l'attribution des autres modules en *systemprocess* et *systemactivity* a pour effet de définir un système global asynchrone. C'est bien le cas d'un protocole de communication où les entités évoluent de manière totalement asynchrone l'une de l'autre.

5.3 Spécification du système global

Considérons une spécification IPN d'un protocole à n entités: $SPEC_p = \langle EP^1, ..., EP^n \rangle$. Compte tenu de la relation de correspondance établie dans la section 5.2, la structure générale du module racine correspondant au protocole $SPEC_p$ est donnée par la Figure 5.1. Cette structure définit l'architecture globale[4] du protocole $SPEC_p$ dans le langage Estelle.
A chaque module `EP[i]` sont associés deux points d'interaction: (a) un point `G` qui lui permet d'émettre des messages à travers le médium et donc de communiquer avec les autres entités, et (b) un point `SAP` qui lui permet d'interagir avec l'utilisateur local `U[i]`. Le point d'interaction `G` est typé par un canal définissant tous les messages qui peuvent être échangés entre les entités de protocole. Puisque dans cette architecture, un module ne peut pas connaître l'identité de l'émetteur quand il reçoit un message à travers le point `G`, les messages échangés sont augmentés par deux paramètres supplémentaires à savoir l'émetteur et le destinataire. Donc la structure du canal typant `G` se présente comme suit:

```
CHANNEL Canal_1(role_1,role_2)
  BY role_1, role_2 :
     (* Liste de tous les messages échangés entre les entités de protocole *)
     m_1(<liste paramètres typés>; <émetteur>,<récepteur>: INTEGER);
     ...
     m_k(<liste paramètres typés>; <émetteur>,<récepteur>: INTEGER);
END
```

4. L'architecture d'une spécification Estelle décrit l'ensemble des modules qui composent cette spécification ainsi que les différentes connexions entre ces modules.

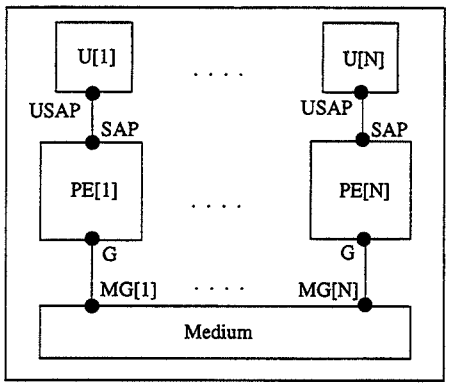

FIG. 5.1 – *Structure générale du module racine associé à $SPEC_p$*

Quant au point d'interaction SAP, il est typé par un canal spécifiant tous les événements entrants et sortants à travers l'ensemble des SAPs de $SPEC_p$. A titre d'exemple, considérons un protocole utilisant deux événements à travers ses SAPs : $\downarrow TConnectRequest()$, $\uparrow TConnectIndication()$. Ces événements permettent, respectivement, de demander et d'indiquer une connexion de transport ISO. Dans ce cas, le canal typant le point d'interaction SAP est spécifié comme suit :

```
CHANNEL Canal_1(role_1, role_2)
  BY role_1 :
     TConnectRequest();
  BY role_2 :
     TConnectIndication();
END
```

Chaque module U[i], représentant un utilisateur, est muni d'un seul point d'interaction USAP qui lui permet d'interagir avec l'entité de protocole locale EP[i]. Le point USAP est donc typé par le même canal que SAP mais ayant un rôle opposé.
La tâche du module Medium est d'aiguiller les messages entre les entités de protocole. Son interface est définie par n points d'interaction, chaque point MG[i] lui permet de recevoir et d'émettre les messages concernant l'entité EP[i]. Nous donnons dans ce qui suit la spécification générale du module racine :

```
SPECIFICATION SPEC_p;
  DEFAULT INDIVIDUAL QUEUE;
  CONST
   (* N = <constante entière> *)
  CHANNEL Canal_1(role_1,role_2);
     BY role_1,role_2 :
        (* Liste de tous les messages échangés entre les entités de protocole *)
  END;
  CHANNEL Canal_2(role_1,role_2);
     BY role_1 :
        (* Liste des événements externes entrants à travers les SAPs *)
     BY role_2 :
```

```
        (* Liste des événements externes sortants à travers les SAPs *)
  END;
  MODULE Protocole SYSTEMPROCESS;
     IP
        G :   Canal_1(role_1);
        SAP : Canal_2(role_1);
  END;
  BODY B_EP_1 FOR Protocole EXTERNAL ; ... ; BODY B_EP_N FOR Protocole EXTERNAL;
  MODULE Utilisateur SYSTEMPROCESS;
     IP
        USAP: Canal_2(role_2);
  END;
  BODY B_U_1 FOR Utilisateur EXTERNAL ; ... ; BODY B_U_N FOR Utilisateur EXTERNAL;
  MODULE MediumFifo SYSTEMPROCESS;
    IP
        MG : ARRAY [1..N] OF Canal_1(role_2);
  END;
  BODY B_Medium FOR MediumFifo EXTERNAL;
  MODVAR EP : ARRAY[1..N] OF Protocole;
         U  : ARRAY[1..N] OF Utilisateur;
         Medium: MediumFifo;
  INITIALIZE BEGIN
     INIT EP[1] WITH B_EP_1; ... ; INIT EP[N] WITH B_EP_N;
     INIT U[1]  WITH B_U_1 ; ... ; INIT U[N]  WITH B_U_N;
     INIT Medium  WITH B_Medium;
     ALL i: 1..N DO BEGIN
        CONNECT EP[i].G   TO Medium.MG[i] ;
        CONNECT EP[i].SAP TO U[i].USAP
     END
  END
END.
```

Le mot clé EXTERNAL permet d'indiquer que la définition du corps d'un module sera fournie ultérieurement. L'utilisation de ce mot clé permet ainsi de générer les spécifications selon une approche "top-down". En effet, dans un premier temps un module racine est produit en faisant abstraction des comportements de ses modules composites, ensuite, dans une deuxième étape, sont produites toutes les spécifications des modules composites. Si ces derniers sont à leur tour composés de sous-modules, alors le principe est réitéré jusqu'aux modules feuilles.

5.4 Spécification du médium et des entités de protocole

Nous décrivons dans ce qui suit les techniques permettant de construire les corps des différents modules introduits dans la section précédente; à savoir le médium et les entités de protocole.

5.4.1 Spécification du comportement du médium

Le rôle principal du médium de communication est de faire transiter les messages entre les différentes entités de protocole. Compte tenu des hypothèses qui en sont faites au chapitre

2, à savoir: médium FIFO sans perte ni duplication de messages, son comportement peut être défini par la transition Estelle suivante:

```
TRANS
   ANY i : 1..N DO
   WHEN MG[i].m(x1,...,xp,i,j)
   BEGIN
     OUTPUT MG[j].m(x1,...,xp,i,j)
   END
```

Quand le médium reçoit, à travers `MG[i]`, un message `m` destiné à `EP[j]`, il l'ajoute dans la file de réception du module `EP[j]`.
Le médium a été spécifié séparément à l'aide d'un module Estelle. Ceci a pour avantage d'offrir la possibilité au spécifieur de le modifier afin de: tenir compte des délais de transit, simuler des pertes de messages, ou encore spécifier un médium fiable à l'aide d'un protocole approprié.

5.4.2 Spécification du comportement d'une entité de protocole

Le comportement d'un module étant défini principalement par un ensemble de transitions, nous commençons d'abord par développer la méthode de construction des transitions Estelle, ensuite nous présentons la spécification générale du comportement d'une entité de protocole. Étant donnée une transition t de EP^i, $\mathcal{X}^i(t)$ définit la primitive de service étiquetant t. Dans une entité de protocole deux types de primitives de service peuvent être utilisés:

1. Une primitive de service interne, dans ce cas la forme de $\mathcal{X}^i(t)$ serait:

$$\langle r_j(m1(x1,...,xp)), Cond, Act, s_k(m2(E1,...,Eq))\rangle$$

   où $x1...xp$ sont des variables paramétrant le message $m1$ et $E1...Eq$ sont des termes paramétrant le message $m2$.

2. Une primitive de service externe, dans ce cas la forme de $\mathcal{X}^i(t)$ serait:

$$\langle SAP, \downarrow evt1(x1,...,xp), Cond, Act, \uparrow evt2(E1,...,Eq)\rangle$$

   où $evt1$ et $evt2$ sont deux événements paramétrés comme dans le cas précédent; l'un est en entrée et l'autre est en sortie à travers le point d'accès SAP.

Les transitions Estelle qui correspondent à t, respectivement dans les cas 1 et 2, sont données comme suit:

```
TRANS (* Cas 1 *)
   NAME t :
   PROVIDED S(t) AND Cond
   WHEN  G.m1(x1,...,xp,j,i)
       BEGIN
         Act;
         OUTPUT G.m2(E1,...,Eq,i,k);
         Mettre_a_jour_marquage(t);
       END
```

```
TRANS (* Cas 2 *)
   NAME t :
   PROVIDED S(t) AND Cond
   WHEN  SAP.evt_1(x1,...,xp);
       BEGIN
         Act;
         OUTPUT SAP.evt_2(E1,...,Eq);
         Mettre_a_jour_marquage(t);
       END
```

`S(t)` est un prédicat qui renvoie "vrai" si la transition `t` est *sensibilisée* dans le réseau de Petri PN^i sous-jacent à l'entité EP^i, et "faux" dans le cas contraire. Donc l'expression de `S(t)` est donnée comme suit :

$$\forall p \in P^i : M^i(p) \geq W^i(p,t)$$

où W^i représente la matrice d'incidence[5] et M^i le vecteur du marquage courant[6].
La procédure `Mettre_a_jour_marquage(t)` consiste à modifier le marquage courant du réseau de Petri PN^i ; elle est donnée par l'expression suivante :

$$\forall p \in P^i : M^i(p) := M^i(p) - W^i(p,t) + W^i(t,p)$$

Quant aux ressources, elles sont traduites par des variables Estelle auxquelles vient s'ajouter un ensemble de procédures et fonctions opérant sur ces variables. Une opération sur une ressource correspond alors à un simple appel local de procédure. Par conséquent, la spécification du comportement d'une entité de protocole EP^i, c'est-à-dire le corps `B_EP_i`, est donnée comme suit :

```
BODY B_EP_i FOR Protocole;
   CONST
     (* Déclaration de la matrice d'incidence W_i associée au réseau de Petri
          sous-jacent à EP_i *)
   VAR
     (* Déclaration du vecteur marquage M_i du réseau de Petri sous-jacent à EP_i *)
     (* Déclaration des variables locales de EP_i *)
     (* Déclaration des variables correspondant aux ressources utilisées dans EP_i
        ainsi que les procédures et fonctions opérant sur chaque ressource *)
   INITIALIZE
     BEGIN
       (* Initialisation du marquage M_i par le marquage initial *)
     END

  (* Liste des transitions de EP_i *)
END
```

5. La matrice d'incidence d'un réseau de Petri définit pour chaque couple *(place, transition)* le nombre de jetons qu'il faut extraire de la *place* pour franchir la *transition*, et pour chaque couple *(transition, place)*, le nombre de jetons qu'il faut ajouter à la *place* après avoir franchi la *transition*.

6. Le vecteur du marquage courant donne pour chaque place le nombre de jetons qu'elle contient dans l'état courant du réseau.

Les différents comportements d'un module Estelle associé à une entité de protocole (i.e., module EP[i]) correspondent donc aux différents chemins du *graphe d'accessibilité* du réseau de Petri sous-jacent à cette entité de protocole. Par conséquent, à chaque séquence d'exécution de transitions Estelle de EP[i] correspond une séquence de transitions IPN dans EP^i et réciproquement. Il existe donc une équivalence de traces entre la spécification Estelle d'un module EP[i] et la spécification de l'entité de protocole EP^i qui lui correspond dans le modèle IPN.

5.5 Exemple illustratif

Dans cette section nous présentons la synthèse des spécifications Estelle d'un protocole de transport ISO entre deux entités EP^1 et EP^2. Il s'agit d'une version simplifiée du protocole présenté au chapitre 3. Dans le but de simplifier les spécifications, nous posons les hypothèses suivantes :

1. La connexion et la déconnexion sont demandées par EP^1.
2. Le fournisseur de service ne peut pas refuser une demande de connexion.
3. Le transfert se fait dans un seul sens : de EP^1 vers EP^2.
4. EP^1 n'envoie une nouvelle donnée que si la précédente a été reçue par EP^2.

La spécification IPN du service est présentée sur la Figure 5.2a. Les événements utilisés dans cette spécification sont notées : *TConReq*, *TConInd*, *TConResp*, *TConConf*, *TDataReq*, *TDataInd*, *TDiscReq*, et *TDiscInd*. Ces événements désignent respectivement : une demande de connexion de transport, une indication d'une connexion, une réponse à une demande de connexion, une confirmation d'une connexion, une demande de transfert de données, une indication de données, une demande de déconnexion et une indication de déconnexion. Ces événements sont paramétrés. Le paramètre *cg* désigne l'adresse du point d'accès de l'entité appelante, *cd* désigne l'adresse du point d'accès de l'entité appelée, *qos*1 et *qos*2 représentent des qualités du service, *d*1 et *d*2 désignent des données, *rg* désigne l'adresse du point d'accès avec lequel la connexion de transport a été établie, et *r* représente la raison de la déconnexion. Partant de la spécification du service, deux entités de protocole sont construites pour être implantées sur l'architecture cible représentée sur la Figure 5.2b. Deux points d'accès *a* et *b* sont, respectivement, associés aux entités de protocole EP^1 et EP^2. Les spécifications IPN des entités de protocole sont données par les Figures 5.3a et 5.3b. Ces spécifications sont construites en appliquant l'algorithme de synthèse présenté dans le chapitre 2.
En appliquant les techniques décrites dans ce chapitre, nous obtenons les spécifications Estelle suivantes du protocole de transport considéré.

• Spécification du module racine :

```
SPECIFICATION SPEC_p;
  DEFAULT INDIVIDUAL QUEUE;
  CONST
     N=2;
  TYPE
```

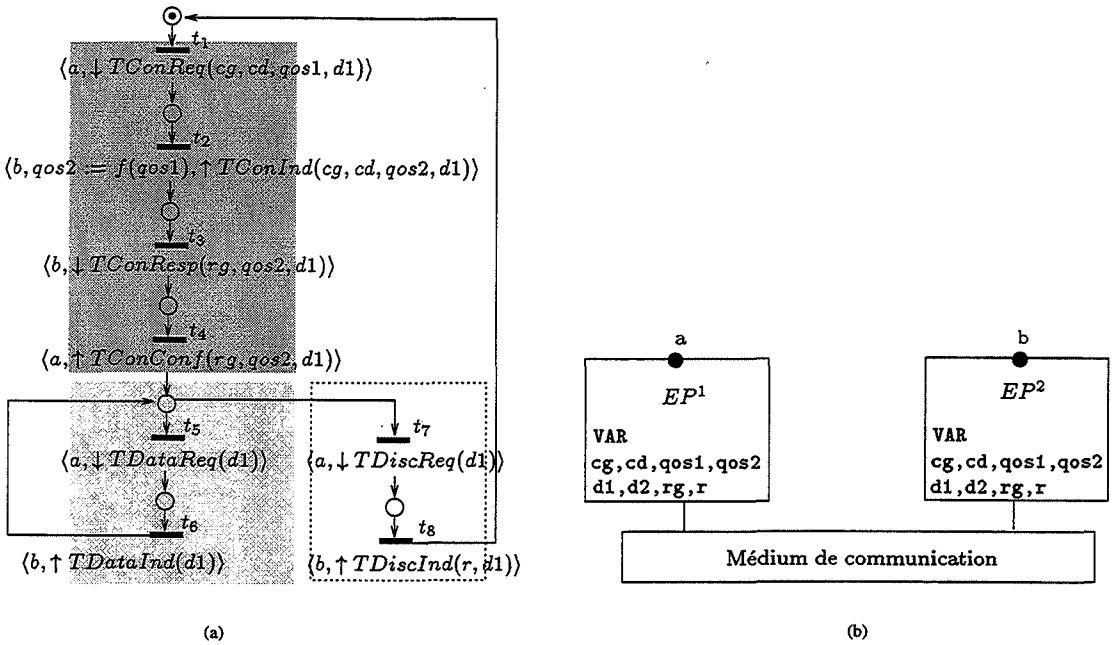

FIG. 5.2 – *Spécification IPN simplifiée du service de transport ISO (a) Service (b) Architecture cible*

```
 type_id_trans=INTEGER;
 type_adr_sap=ARRAY[1..12]OF INTEGER;
 type_qos=INTEGER;
 type_data=ARRAY[1..100]OF CHAR;
 type_raison=ARRAY[1..100]OF CHAR;
CHANNEL Canal_1(role_1,role_2);
   BY role_1,role_2 :
      (* Liste de tous les messages échangés entre les entités de protocole *)
      M3_1(t:type_id_trans; cg,cd:type_adr_sap; qos:type_qos; d:type_data; i,j:INTEGER);
      M3_2(t:type_id_trans; rg:type_adr_sap; qos:type_qos; d:type_data; i,j:INTEGER);
      M3_3(t:type_id_trans; d:type_data; i,j:INTEGER);
      M1(t:type_id_trans; i,j:INTEGER);
END;
CHANNEL Canal_2(role_1,role_2);
   BY role_1 :
      (* Liste des événements externes entrants à travers les SAPs *)
      TConReq(cg,cd:type_adr_sap; qos:type_qos; d:type_data);
      TconResp(rg:type_adr_sap; qos:type_qos; d:type_data);
      TDataReq(d:type_data);
      TDiscReq(d:type_data);
   BY role_2 :
      (* Liste des événements externes sortants à travers les SAPs *)
      TConInd(cg,cd:type_adr_sap; qos:type_qos; d:type_data);
      TConConf(rg:type_adr_sap; qos:type_qos; d:type_data);
      TDataInd(d:type_data);
      TDiscInd(r:type_raison; d:type_data);
END;
```

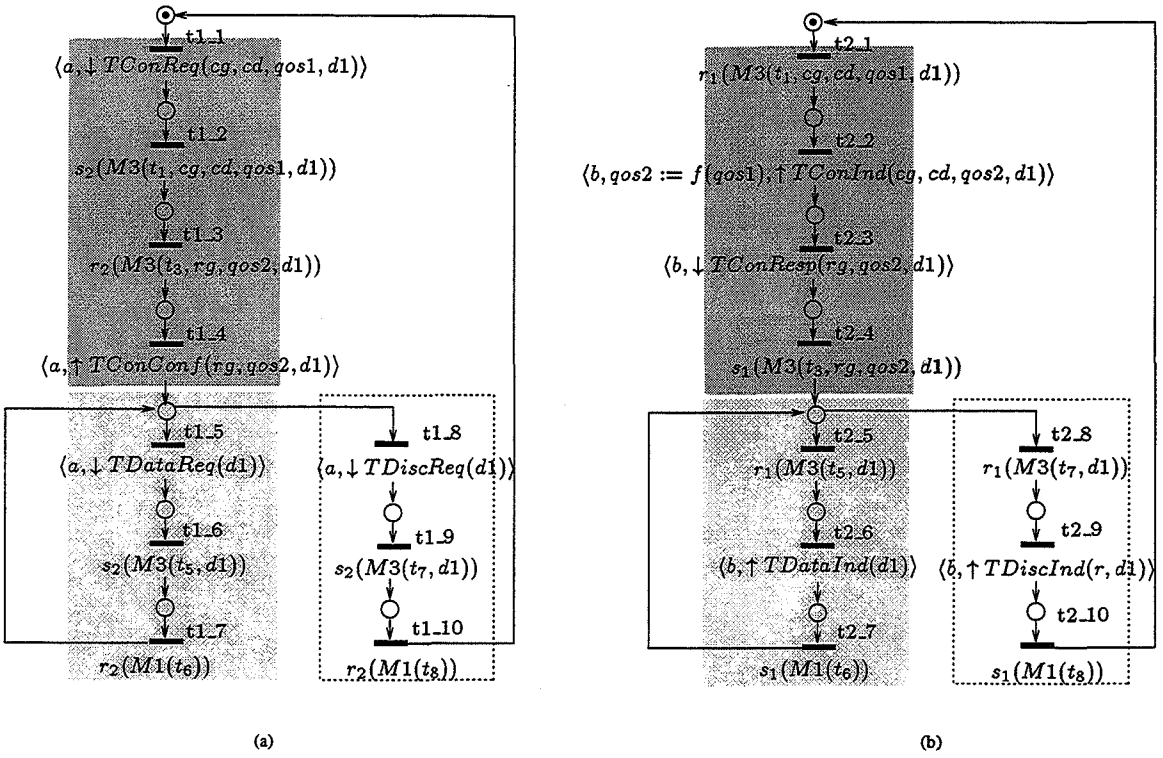

FIG. 5.3 – *Spécification IPN simplifiée du protocole de transport ISO (a) Entité EP^1 (b) Entité EP^2*

```
MODULE Protocole SYSTEMPROCESS;
   IP
      G :   Canal_1(role_1);
      SAP : Canal_2(role_1);
END;
BODY B_EP_1 FOR Protocole EXTERNAL; BODY B_EP_2 FOR Protocole EXTERNAL;

MODULE MediumFifo SYSTEMPROCESS;
  IP
      MG : ARRAY [1..N] OF Canal_1(role_2);
END;
BODY B_Medium FOR MediumFifo EXTERNAL;

MODVAR EP : ARRAY[1..N] OF Protocole;
       U  : ARRAY[1..N] OF Utilisateur;
       Medium: MediumFifo;
INITIALIZE BEGIN
   INIT EP[1] WITH B_EP_1; INIT EP[2] WITH B_EP_2;
   INIT U[1]  WITH B_U_1 ; INIT U[2]  WITH B_U_2;
   INIT Medium  WITH B_Medium;
   ALL i: 1..N DO BEGIN
      CONNECT EP[i].G   TO Medium.UG[i] ;
      CONNECT EP[i].SAP TO U[i].USAP
   END
```

```
  END
END. (* Spec_p *)
```

• Spécification du comportement du module EP[1] :

```
BODY B_EP_1 FOR Protocole;
    CONST
       (* Déclaration de la matrice d'incidence W_1 associée au réseau de Petri
          sous-jacent à EP_1 *)
       W_1= . . .
       (* Identités des transitions de service utilisées *)
       t_1=1; t_3=3; t_5=5; t_6=6; t_7=7; t_8=8;
       (* Nombre de places *)
       NB_PLACES=9;
       EP1=1; EP2=2;
    VAR
       (* Déclaration du vecteur marquage M_1 du réseau de Petri sous-jacent à EP_1 *)
       M_1: ARRAY[1..NB_PLACES]OF INTEGER;   k:INTEGER;
       (* Déclaration des variables locales à EP_1 *)
       cd,cg,rg: type_adr_sap;
       qos1,qos2: type_qos;
       d1,d2:type_data;
       (* Déclaration des procédures et fonctions *)
       PROCEDURE Mise_a_jour_marquage(t:INTEGER);
       FUNCTION S(t:INTEGER):BOOLEAN;
    INITIALIZE
       BEGIN
         (* Initialisation du marquage M_1 par le marquage initial *)
         M_1[1]:=1; FOR k:=2 TO NB_PLACES DO M_1[k]:=0;
       END
    (* Liste des transitions de EP_1 *)
    TRANS NAME t1_1:  PROVIDED S(1)
     WHEN  a.TConReq(cg,cd,qos1,d1)
       BEGIN Mise_a_jour_marquage(1) END
    TRANS NAME t1_2: PROVIDED S(2)
       BEGIN Mise_a_jour_marquage(2); OUTPUT G.M3_1(t_1,cg,cd,qos1,d1,EP1,EP2) END
    TRANS NAME t1_3: PROVIDED S(3)
     WHEN  G.M3_2(t_3,rg,qos2,d1,EP2,EP1)
       BEGIN Mise_a_jour_marquage(3) END
    TRANS NAME t1_4: PROVIDED S(4)
       BEGIN Mise_a_jour_marquage(4); OUTPUT a.TConConf(rg,qos2,d1) END
    TRANS NAME t1_5: PROVIDED S(5)
     WHEN  a.DataReq(d1)
       BEGIN Mise_a_jour_marquage(5) END
    TRANS NAME t1_6: PROVIDED S(6)
       BEGIN Mise_a_jour_marquage(6); OUTPUT G.M3_3(t_5,d1,EP1,EP2) END
    TRANS NAME t1_7: PROVIDED S(7)
     WHEN  G.M1(t_6,EP2,EP1)
       BEGIN Mise_a_jour_marquage(7) END
    TRANS NAME t1_8: PROVIDED S(8)
     WHEN  a.TDiscReq(d1)
       BEGIN Mise_a_jour_marquage(8) END
    TRANS NAME t1_9: PROVIDED S(9)
       BEGIN Mise_a_jour_marquage(9); OUTPUT G.M3_3(t_7,d1,EP1,EP2) END
    TRANS NAME t1_10:PROVIDED S(10)
     WHEN  G.M1(t_8,EP2,EP1)
```

```
      BEGIN Mise_a_jour_marquage(10) END
END
```

• Spécification du comportement du module EP[2] :

```
BODY B_EP_2 FOR Protocole;
    CONST
       (* Déclaration de la matrice d'incidence W_2 associée au réseau de Petri
          sous-jacent à EP_2 *)
       W_2= . . .
       (* Identités des transitions de service utilisées *)
       t_1=1; t_3=3; t_5=5; t_6=6; t_7=7; t_8=8;
       (* Nombre de places *)
       NB_PLACES=9;
       EP1=1; EP2=2;
    VAR
       (* Déclaration du vecteur marquage M_2 du réseau de Petri sous-jacent à EP_2 *)
       M_2: ARRAY[1..NB_PLACES]OF INTEGER; k:INTEGER;
       (* Déclaration des variables locales de EP_2 *)
       cd,cg,rg: type_adr_sap;
       qos1,qos2: type_qos;
       d1,d2:type_data;
       r: type_raison;
       (* Déclaration des procédures et fonctions *)
       PROCEDURE Mise_a_jour_marquage(t:INTEGER);
       FUNCTION S(t:INTEGER):BOOLEAN;
       FUNCTION f(qos:type_qos):type_qos;
    INITIALIZE
       BEGIN
         (* Initialisation du marquage M_2 par le marquage initial *)
         M_2[1]:=1; FOR k:=2 TO NB_PLACES DO M_2[k]:=0;
       END
    (* Liste des transitions de EP_2 *)
    TRANS NAME t2_1:  PROVIDED S(1)
     WHEN  G.M3_1(t_1,cg,cd,qos1,d1,EP1,EP2)
        BEGIN Mise_a_jour_marquage(1) END
    TRANS NAME t2_2: PROVIDED S(2)
        BEGIN Mise_a_jour_marquage(2); qos2:=f(qos1);OUTPUT b.TConInd(cg,cd,qos2,d1) END
    TRANS NAME t2_3: PROVIDED S(3)
     WHEN  b.TConResp(rg,qos2,d1)
        BEGIN Mise_a_jour_marquage(3) END
    TRANS NAME t2_4: PROVIDED S(4)
        BEGIN Mise_a_jour_marquage(4); OUTPUT G.M3_2(t_3,rg,qos2,d1,EP2,EP1) END
    TRANS NAME t2_5: PROVIDED S(5)
     WHEN  G.M3_3(t_5,d1,EP1,EP2)
        BEGIN Mise_a_jour_marquage(5) END
    TRANS NAME t2_6: PROVIDED S(6)
        BEGIN Mise_a_jour_marquage(6); OUTPUT b.TDataInd(d1) END
    TRANS NAME t2_7: PROVIDED S(7)
        BEGIN Mise_a_jour_marquage(7); OUTPUT G.M1(t_6,EP2,EP1) END
    TRANS NAME t2_8: PROVIDED S(8)
     WHEN  G.M3_3(t_7,d1,EP1,EP2)
        BEGIN Mise_a_jour_marquage(8) END
    TRANS NAME t2_9: PROVIDED S(9)
        BEGIN Mise_a_jour_marquage(9); OUTPUT b.TDiscInd(r,d1) END
    TRANS NAME t2_10:PROVIDED S(10)
```

```
        BEGIN Mise_a_jour_marquage(10); OUTPUT G.M1(t_8,EP2,EP1) END
END
```

• **Spécification du comportement du médium :**

```
BODY B_Medium FOR MediumFifo
   CONST
   N=2;
   VAR
       cd,cg,rg: type_adr_sap;
       qos: type_qos;
       d:type_data;
       t,i,j: INTEGER;
   TRANS
     ANY k : 1..N DO
     WHEN MG[k].M3_1(t,cg,cd,qos,d,i,j)
        BEGIN OUTPUT MG[j].M3_1(t,cg,cd,qos,d,i,j) END
     WHEN MG[k].M3_2(t,rg,qos,d,i,j);
        BEGIN OUTPUT MG[j].M3_2(t,rg,qos,d,i,j);
     WHEN MG[k].M3_3(t,d,i,j);
        BEGIN OUTPUT MG[j].M3_3(t,d,i,j) END
     WHEN MG[k].M1(t,i,j)
        BEGIN OUTPUT MG[j].M3_3(t,i,j) END
END
```

La production du code Estelle est facilement automatisable. Pour ce faire, un module de génération de code Estelle devra être développé et intégré aux outils STEPS (cf. chapitre 4).

5.6 Conclusion

Nous avons présenté dans ce chapitre une méthodologie de synthèse de protocole dans le langage Estelle à partir d'une spécification de service écrite dans le modèle IPN. La spécification de service est d'abord raffinée par l'algorithme donné au chapitre 2 pour construire les spécifications IPN des entités de protocole. Les spécifications IPN du protocole sont ensuite translatées dans le langage Estelle. Les spécifications Estelle obtenues sont équivalentes, selon l'équivalence de traces, aux spécifications IPN du protocole. Ces dernières sont d'une part libres de toute erreur logique, et d'autre part équivalentes, selon l'équivalence de traces, aux spécifications de service. Ceci garantit la correction de la méthodologie globale. Les spécifications Estelle synthétisées peuvent être exploitées directement par les outils logiciels existants tels que EDT[BUD 92] pour produire du code exécutable ou effectuer une évaluation de performances du protocole.

Conclusion

Le développement des protocoles de communication est une tâche cruciale pour la mise en œuvre de systèmes distribués modernes et à grande échelle. Actuellement, il est établi que les méthodes formelles présentent un intérêt certain pour faciliter cette tâche. Cependant, les méthodes existantes font souvent des restrictions qui les rendent quasiment non applicables dans un monde réel. Nous nous sommes intéressés pendant notre étude à définir, dans un premier temps, les exigences qu'une méthode formelle doit satisfaire afin qu'elle soit facilement applicable dans l'industrie. Nous avons résumé ces exigences en introduction de ce document. Partant de cette base, nous avons établi que la synthèse de protocoles satisfait mieux ces exigences qu'une approche analytique, ce qui nous a amené à explorer cette direction. Les contributions de cette thèse dans le domaine du développement de protocoles de communication se résument comme suit :

Nous avons élaboré dans un premier temps un état de l'art détaillé sur le développement des protocoles de communication en mettant l'accent sur la "synthèse de protocoles". Dans la classe des protocoles nous avons inclus les applications réparties et les convertisseurs de protocoles. Un document donnant une vision aussi globale et une étude aussi détaillée n'existe pas à notre connaissance. Cette étude nous a servis à dégager les questions essentielles à aborder dans notre thèse.

Nous avons proposé une méthode de synthèse automatique de spécifications de protocoles à partir de spécifications de services dans un modèle de réseaux de Petri interprétés. L'originalité de notre approche réside dans l'élaboration d'un nouveau concept de raffinement de réseaux de Petri qui assure la correction des protocoles construits de façon incrémentale. Nous avons ainsi défini un ensemble de règles de synthèse de base servant de constructeurs pour les règles de synthèse que nous avons appelées contextuelles. La capacité d'étendre l'un ou l'autre de ces deux ensembles de règles offre à notre approche un contexte de synthèse extensible. Ceci constitue l'avantage principal de notre approche par rapport aux méthodes existantes. Une autre originalité de notre méthode est l'utilisation d'un modèle de réseaux de Petri interprétés pour la spécification des services et des protocoles. La puissance d'expression des réseaux de Petri pour la description formelle des protocoles de communication est déjà reconnue. Nous avons enrichi ce modèle de base par de nouveaux concepts propres à la spécification des services et des protocoles, tels que : les points d'accès au service, les primitives de service, les événements internes et externes, les variables et les ressources. Partant d'une description de service dans ce modèle, notre technique de synthèse permet

de dériver automatiquement au niveau protocole l'ensemble des fonctionnalités suivantes :

- traitements répartis, tels que les appels distants, les mises à jour de ressources distantes et éventuellement dupliquées, etc.
- flot de contrôle avec parallélisme et choix distribué,
- flot de données avec contrôle de la cohérence.

Notre technique est de ce fait plus générale que les méthodes existantes qui ignorent certaines fonctionnalités. Elle constitue, à notre avis, un "bon" compromis entre le pouvoir d'expression du modèle de spécification des services et le pouvoir de synthèse. Nous avons prouvé que les protocoles produits par notre algorithme de synthèse sont corrects. Nous avons également montré que la complexité en temps de notre algorithme est polynômiale.

Afin de démontrer l'utilité pratique de notre approche de synthèse, nous avons présenté une application réelle. Il s'agit de la conception du protocole de transport ISO classe 0 (ISO 8073). Le point de départ de la conception étant la spécification du service de transport ISO (ISO 8072).

Pour démontrer la faisabilité de notre approche de synthèse, nous avons, d'une part, proposé une méthodologie, et d'autre part, développé un ensemble d'outils logiciels la supportant, appelé STEPS (Software Tool-set for automatEd Protocol Synthesis). L'apport principal de STEPS par rapport aux outils existants est la prise en compte des spécifications de service dans le développement des protocoles et des applications réparties. L'intérêt de cet apport est double : d'une part, il permet de diminuer le coût de production des protocoles et des applications réparties, et d'autre part, il garantit la correction des spécifications produites.

La sémantique de notre modèle de spécification étant assez proche de celle du langage Estelle, nous avons proposé une extension de notre démarche pour la synthèse de spécifications de protocoles dans le langage Estelle. Ceci constitue une autre originalité de notre travail. L'intérêt est que les spécifications produites peuvent être directement exploitées par des outils existants autour d'Estelle afin de générer automatiquement du code exécutable ou d'effectuer des simulations et des évaluations de performances.

Nous estimons que l'ensemble de ces contributions satisfont les objectifs que nous nous étions fixés au départ. Cependant, des ouvertures méritent d'être explorées suite à notre travail, elles se résument comme suit :

Certaines fonctionnalités protocolaires, telles que le recouvrement d'erreurs et le contrôle du flux de données, ne peuvent pas être spécifiées au niveau service, et ne peuvent donc pas être synthétisées automatiquement par notre algorithme de synthèse. Ceci requiert donc l'extension de notre approche de conception afin de pouvoir prendre en compte des fonctionnalités intrinsèques au niveau protocolaire. Une démarche a été esquissée au chapitre 4.

Le temps est un élément important dans la spécification des systèmes réels. Une extension intéressante de notre travail consisterait à introduire des contraintes de temps dans le modèle de spécification des services. Ceci impliquerait l'extension de la technique de synthèse

pour prendre en compte les contraintes de temps à la fois dans les spécifications IPN et les spécifications Estelle des protocoles construits. Une démarche consisterait à étendre le modèle IPN par des contraintes de temps telles que celles utilisées dans le modèle des réseaux de Petri temporels [BER 91]; à savoir chaque transition est munie d'un intervalle temps spécifiant les dates au plus tôt et au plus tard de tir de la transition. Partant des intervalles de temps associés aux transitions de la spécification de service, il s'agit de trouver une stratégie permettant de dériver les intervalles de temps associés aux transitions des entités de protocole.
D'autres extensions peuvent aussi être envisagées pour le modèle IPN, telles que la modularité et la généricité. L'intérêt de la modularité est de pouvoir construire de nouveaux services en réutilisant des services existants. La généricité permet de réduire la spécification de systèmes symétriques, elle peut être introduite en paramétrant les points d'accès au service. Pour cela, on peut introduire une coloration dans le modèle IPN pour paramétrer les transitions par les points d'accès au services. Ces points méritent d'être étudiées de manière approfondie.

Étant donnée une transition d'une spécification de service, le problème qui consiste à dériver un ensemble de sous-réseaux de Petri coopérants implantant la transition sur l'architecture cible admet plusieurs solutions. En effet, si la transition invoque une opération sur une ressource distante, cette opération peut être exécutée sur tout site qui détient une copie de la ressource. D'autre part, la synthèse du flot de données peut être réalisée de plusieurs façons en fonction des sites choisis à cet effet (voir section 3.2 du chapitre 2). Parmi les solutions possibles pour dériver les sous-réseaux de Petri, il serait intéressant de trouver une stratégie permettant de choisir la solution qui minimise le nombre de messages total échangés entre les entités de protocoles.

Enfin, l'application de notre technique de synthèse pour la construction de convertisseurs de protocoles n'a pas été étudiée. Nous avons vu au chapitre 1 que la synthèse automatique de ce type de logiciels est très attractive, mais malheureusement, les techniques existantes introduisent des complexités de calculs très élevées. Ce point mérite d'être étudié plus en avant.

Bibliographie

[ALG 93] B. ALGAYRES, V. COELHO, L. DOLDI, H. GARAVEL, Y. LEJEUNE and C. RODRIGUEZ, « VESAR: a pragmatic approach to formal specification and verification », *Computer Networks and ISDN Systems 25 (1993)*, pp. 779-790.

[AMY 91] O. AMYAY, « Méthodologie de spécification d'activités de communication dans une architecture multi-couches. Vers la définition d'une base de connaissances », *Thèse de Doctorat de l'Université Paul Sabatier*, Toulouse, 1991.

[BAR 79] W. A. BARRETT, J. D. COUCH, « Compiler Construction: Theory and Practice », *Science Research Associates (Editor)*, Chicago 1979.

[BER 89] J.M. BERNARD, J.L. MOUNIER, « Atelier logiciel AMI, un environnement multi-utilisateurs, multi-sessions pour une architecture distribuée », *Séminaire Franco-Brésilien sur les systèmes informatiques répartis,* 1989.

[BER 87] P.A. BERNSTEIN, V. HADZILACOS, N. GOODMAN, « Concurrency control and recovery in database systems », *Reading MA: Addison-Wesley*, 1987.

[BER 86] G. BERTHELOT, « Transformations and decomposisitions of nets », *Advances in Petri Nets, LNCS 254, Springer-Verlag*, pp.359-376, 1986.

[BER 91] B. BERTHOMIEU, M. DIAZ, « Modeling and Verification of Time Dependent Systems Using Time Petri Nets », *IEEE Transactions on Software Engineering*, Vol. 17, N. 3, March 1991, pp. 259-273.

[BOC 78] G.v. BOCHMANN, « Finite state description of communication protocols », *Computer Networks* 2 (4/5), Septembre 1978, pp. 361-372.

[BOC 86] G. v. BOCHMANN, R. GOTZHEIN, « Deriving Protocol Specifications From service Specifications », *Computer Communication Review*, Vol. 16, No. 3, pp 148-156, ACM Press, August 1986.

[BOL 87] T. BOLOGNESI, E. BRINKSMA, « Introduction to the ISO specification language LOTOS », *in Computer Networks and ISDN Systems*, Vol.14, N. 1, 1987, pp. 25-59.

[BUD 87] S. BUDKOWSKI, P. DEMBINSKI « An Introduction to Estelle: A Specification Language for Distributed Systems », *Computer Networks and ISDN Systems*, Vol. 14, 1987, pp.3-23.

[BUD 92] S. BUDKOWSKI, « Estelle development toolset (EDT) », *Computer Networks and ISDN Systems*, Vol. 25, 1992, pp. 63-82, North-Holland.

[CAI 97] B. CAILLAUD, P. CASPI, A. GIRAULT, C. JARD, « Distributing automata for asynchronous networks of processors » *RAIRO - APII - JESA*, Vol. 31, No. 3, 1997, pp. 503-524.

[CAL 89] K. L. CALVERT, S. S. LAM, « Deriving a Protocol Converter: a Top-Down Method », *Computer Communication Review*, Vol. 19, No. 4, pp 247-258, ACM Press, September 1989.

[CAL 90] K. L. CALVERT, S. S. LAM, « Formal Methods for Protocol Conversion », *IEEE Journal on Selected Areas in Communications*, ISSN 0733-8716, Vol. 8, No. 1, pp 127-142, 1990.

[CCITT 88] CCITT, recommendations Z.100: Specification and Description Language SDL, APIX-35, 1988.

[CHA 93] S. T. CHANSON, A. A. F. LOUREIRO, S. T. VUONG, « On tools supporting the use of formal description techniques in protocol development », *Computer Network and ISDN Systems*, 25 (1993), pp. 723-739.

[CHA 94a] D. Y. CHAO, D. T. WANG, « A synthesis technique for general Petri nets », *Journal of Systems Integration*, Vol. 4, No.1, pp.67-102, 1994.

[CHA 94b] D. Y. CHAO, D. T. WANG, « An Interactive Tool for Design, Simulation, Verification, and Synthesis of Protocols », *Software Practice and Experience*, Vol. 24, No. 8, pp. 747-783, August 1994.

[CHU 88] P. M. CHU, M. T. LIU, « Synthesizing Protocol Specifications From Service Specifications in FSM Model », *Proc. of Computer Networking Symposium '88*, pp 173-182, April 1988.

[COU 92] J.P COURTIAT, P. SAQUI-SANNES, « ESTIM : an integrated environment for the simulation and the verification of OSI protocols specified in Estelle* », *Computer Networks and ISDN Systems* 25(1992) 83-98, North-Holland.

[COU 96] J.P. COURTIAT, P. DEMBINSKI, G.J. HOLZMANN, L. LOGRIPPO, H. RUDIN, P. ZAVE, « Formal methods after 15 years: Status and Trends. A paper based on contributions of the panelists at the FORmal TEchnique'95 Conference, Montreal, October 1995 », *Computer Networks and ISDN Systems* 28(1996), pp. 1845-1855.

[DAV 92] R. DAVID, H. ALLA, « Du Grafcet aux réseaux de Petri », *Edition Hermes*, 1992.

[EIJ-89] P.v. EIJK, « Tools for LOTOS specification style transformation », *Proc. of FORTE'89*, S.T Vuong (Editor), Vancouver, 5-8 December, 19989, pp. 43-51, North-Holland, 1990.

[GOT 90a] R. GOTZHEIN, G. v. BOCHMANN, « Deriving protocol specifications from service specifications including parameters », *ACM Transactions on Computer Systems*, ISSN 0734-2071, 1990, Vol. 8, No. 4, pp 255-283.

[GOT 90b] R. GOTZHEIN, « Specifying communication services with temporal logic », *in L. Logrippo, R.L. Probert and H. Ural (Editors), Protocol Specification Testing and Verification, IFIP WG6.1, 10^{th} International Symposium*, Ottawa, Canada, 12-15 June 1990, pp. 295-309.

[GOU 84] M. G. GOUDA, Y. T. YU, « Synthesis of communicating finite state machine with guaranteed progress », *IEEE Transactions on Communications*, Vol.32, N.7, pp. 779-788, 1984.

[GRE 86] P. GREEN, « Protocol Conversion », *IEEE transactions on Communications*, Vol. 34, N. 3, pp. 257-268, March 1986.

[HAL 86] J. Y. HALPERN, « Reasoning about Knowledge, An Overview », *J.Y. Halpern, editor, Reasoning about knowledge*, Kaufmann 1986.

[HAN 89] H. HANSON, B. JONSSON, F. ORAVA, B. PEHRSON. « Specification for Verification », *Proc. of 2nd International Conference on Formal Description Techniques*, Eds. S. Vuong, pp. 347-364, December 1989.

[HIG 93] T. HIGASHINO, K. OKANO, H. IMAJO, K. TANIGUCHI, « Deriving Protocol Specifications from Service Specifications in Extended FSM Models », *Proc. of 13th International Conference on Distributed Computing Systems*, pp 141-149, May 1993, Pittsburgh Pennsylvania.

[HOA 85] C. A. R. HOARE, « Communicating Sequential Processes », *Prentice-Hall*, 1985.

[ISO 7498] ISO/CCITT, Information Processing Systems - Open Systems Interconnection - Basic Reference Model, ISO 7498 / CCITT X.200.

[ISO 8072] ISO, Information Processing System - Open Systems Interconnection - Transport Service Definition IS 8072, 1985.

[ISO 8073] ISO, Information Processing System - Open Systems Interconnection - Connection Oriented Transport Protocol Specification, IS 8073, 1985.

[ISO 8807] ISO-IS 8807, *LOTOS a Formal Description Technique based on the temporal ordering of observational behaviour.* 1988.

[ISO 9074] ISO 9074. Information Processing Systems - Open Systems Interconnection - *Estelle: A Formal Description Technique Based on an Extended State Model.* 1989.

[ISO 9646] ISO 9646, OSI Conformance Testing Methodology and Framework.

[JUA 91] G. JUANOLE, et. al., « Interconnexion de réseaux - Concepts - Exemples », *Revue Réseaux et Informatique Répartie*, Vol.1, N. 1, pp. 7-57, 1991.

[KAH 95] H. KAHLOUCHE, « HSSL : Un Formalisme pour la Spécification et la Vérification des Systèmes Distribués Symétriques et Hiérarchiques », *Actes des 7ièmes Rencontres Francophones du Parallélisme (RenPar'7)*, Faculté Polytechniques de Mons (Belgique), Mai 1995.

[KAH 96] H. KAHLOUCHE, J.J. GIRARDOT, « A Stepwise Refinement Based Approach for Synthesizing Protocol Specifications in an Interpreted Petri Net Model », *Proc. of IEEE INFOCOM'1996*, San Francisco, March 24-28, 1996, pp. 1165-1173.

[KAH 97a] H. KAHLOUCHE, J.J. GIRARDOT, « Design of the ISO Class 0 Transport Protocol: A Stepwise Refinement based Approach », *Proc. of IEEE International Performance, Computing and Communications Conference (IPCCC'97)*, Phoenix (USA), 5-7 February, 1997, pp. 363-370.

[KAH 97b] H. KAHLOUCHE, J.J. GIRARDOT, « Automated Protocol Synthesis in ESTELLE Specification Language », *Proc. of IASTED/ISMM International Conference on Modeling and Simulation*, Pittsburgh (USA), May 14-17, 1997, pp. 66-70.

[KAH 97c] H. KAHLOUCHE, « STEPS: a Software Toolset for automatEted Protocol Synthesis », *Proc. of IEEE 6th International Conference on Computer Communications and Networks (ICCCN'97)*, Las Vegas (USA), September 22-25, 1997.

[KAK 94] Y. KAKUDA, H. IGARASHI, T. KIKUNO, « Automated synthesis of protocol specifications with message collisions and verification of timeliness », *Proc. of 1994 International Conference on Network Protocols,* Massachusetts, October 25-28, 1994.

[KAN 93a] C. KANT, T. HIGASHINO, G. v. BOCHMANN, « Deriving Protocol Specifications from Service Specifications Written in LOTOS », *Proc. of 12th International IEEE Phoenix Conference on Computers and Communications*, 1993.

[KAN 93b] C. KANT, « Deriving Protocol Specifications from Service Specifications », *Ph. D Thesis*, University de Montreal, 1993.

[KAP 91a] M. KAPUS-KOLAR, « Deriving Protocol Specifications from Service Specifications with Heterogeneous Timing Requirements », *Proc. of IEEE International Conference on Software Engineering for Real Time Systems*, pp. 266-270, Royaume-Uni, 1991.

[KEL 93] S. G. KELEKAR, G. W. HART, « Synthesis of Protocols and Protocol Converters using the Submodule Construction Approach », *Proc. of Protocol Specification Testing and Verification*, IFIP 1993, Elsevier Science Publishers B. V. (North-Holland).

[KHE 89] F. KHENDEK, G. v BOCHMANN, C. KANT, « New Results in Deriving Protocol Specifications From Service Specifications », *Computer Communication Review*, Vol. 19, No. 4, pp 136-145, ACM Press, September 1989.

[KHO 94a] A. KHOUMSI, G. v. BOCHMANN, R. DSSOULI, « Dérivation de Spécifications de protocole à partir de spécifications de service avec des contraintes temps réel », *Réseaux et Informatique Répartie*, Vol. 4, N. 1, pp. 7-29, 1994.

[KHO 94b] A. KHOUMSI, G.v. BOCHMANN, R. DSSOULI, « On specifying services and synthesizing protocols for real-time applications », *Proc. of 14th International Workshop on Protocol Specification, Testing and Verification*, 1994, pp. 185-200.

[KHO 94c] A. KHOUMSI, G.v. BOCHMANN, R. DSSOULI, « Contrôle et extension des systèmes à événements discrets totalement et partiellement observables », *Proc. of Third Maghrebian Conference on Software Engineering and Artificial Intelligence*, Rabat, Avril 1994, pp. 161-470.

[KHO 95] A. KHOUMSI, G.v. BOCHMANN, « Protocol Synthesis using Basic Lotos and Global Variables », *Proc. of International Conference on Network Protocols (ICPN'95)*, pp. 126-133. Nov 7-10, 1995, Tokyo (Japan).

[KOV 91] A.V. KOVALYOV, « On complete reducibility of some classes of Petri nets », *Proc. of 11th International Conference on Application and Theory of Petri Nets*, pp. 352-366, 1991.

[KOV 92] A. V. KOVALYOV, « Concurrency relation and the safety problem for Petri nets » *Proc. of 13th International Conference on Application and Theory of Petri Nets*, LNCS 616, 1992.

[KOV 95] A.V. KOVALYOV, J. ESPARZA, « A polynomial algorithm to compute the concurrency relation of free-choice Signal Transition Graphs », *Proc. of the Workshop on Discrete Event Systems (WODES'96)*, Edinburgh, August 96, 1996.

[LAM 78] L. LAMPORT, « Time, clocks, and ordering of events in a distributed systems » *Communication of the ACM*, n.21(7), pp 558-564 (1978).

[LAN 90] R. LANGERAK, « Decomposition of functionality; a correctness-preserving LOTOS transformations », *Proc. of Tenth International IFIP WG 6.1 Symposium on PSTV*, Ottawa, June 1990.

[LIN 87] F.J. LIN, P.M. CHU, M.T. LIU, « Protocol verification using reachability analysis: the state space explosion problem and relief strategies », *Proc. of ACM SIGCOMM'87 Workshop*, pp. 126-135, 1987.

[LIU 89] M.T. LIU, « Protocol engineering », *in Advances in Computers*, Vol. 29, pp. 79-195, 1989.

[MAN 84] Z. MANNA, P. WOLPER, « Synthesis of communicating processes from temporal logic specifications », *ACM Transactions on Programming Languages and Systems*, Vol. 6, No. 1, Jan. 1984, pp. 68-93.

[MER 83] P. MERLIN, G. v. BOCHMANN, « On the construction of submodule specifications and communication protocols », *ACM TOPLAS*, Vol.5, N.1, pp 1-25, 1983.

[MIL 89] R. MILNER, « Communication and Concurrency », *Prentice-Hall* 1989.

[MUR 89] T. MURATA, « Petri Nets : Properties, Analysis and Applications », *Proceedings of the IEEE*, Vol.77 No.4, pp. 541-580 (1989).

[NAT 95] A. NATAKA, T. HIGASHINO, K. TANIGUSHI, « Protocol Synthesis from Timed and Structured Specifications », *Proc. of International Conference on Network Protocols (ICPN'95)*, pp. 74-81. Nov 7-10, 1995, Tokyo (Japan).

[NOO 92] M. NOOSONG, « Interconnexion de Réseaux Informatiques Hétérogènes : Méthode Formelle de Conversion de Protocoles », *Thèse de Doctorat*, École Nationale Supérieure des Télécommunications, Paris, 1992.

[OKU 86] K. OKUMURA, « A formal protocol conversion method », *Proc. of ACM SIGCOMM'86*, Stowe, VT, 1986.

[PEY 96] M. PEYRAVIAN, C.T. LEA, « Deriving deadlock and unspecified reception free protocol converters from message mapping sets », *Computer Networks and ISDN Systems*, 28 (1996) 1831-1844.

[PRO 91] R. PROBERT, K. SALEH, « Synthesis of Communication protocols: Survey and Assessment », *IEEE Transaction on Computing*, Vol. C-40, No 4, pp. 468-475, 1991.

[RAJ 91] M. RAJAGOPAL, R.E. MILLER « Synthesizing a Protocol Converter from executable protocol traces », *IEEE transactions on Computers*, ISSN 0018-9340, Vol. C-40, No. 4, pp 487-499, 1991.

[RAM 85] C. V. RAMAMOORTHY, S. T. DONG, Y. USUDA, « An Implementation of an Automated Protocol Synthesizer (APS) and its Application to the X.21 Protocol », *IEEE transactions on Software Engineering*, Vol. SE-11, N. 9, Sept. 1985, pp. 886-908.

[RAM 86] C. V. RAMAMOORTHY, Y. YAW, R. AGGARWAL, J. SONG, « Synthesis of Two Party Error-Recoverable Protocols », *Proc. of ACM SIGCOMM'86 Symposium*, Vermont, USA, pp. 227-235, 1986.

[RIC 87] J.L. RICHIER, C. RODRIGUEZ, J. SIFAKIS, J. VEOIRON, « Verification in XEXAR of the sliding window protocol », *Protocol Specification, Testing and Verification*, VII, IFIP 1987.

[SAL 90] K. SALEH, R. PROBERT, « A service-based method for the synthesis of communications protocols », *International Journal of mini and microcomputers*, Vol 12, NO 3, 1990, pp97-103.

[SAL 91] K. SALEH, « Synthesis methods for the design and validation of communication protocols », *Ph.D dissertation*, University of Ottawa, Jan 1991.

[SAL 95] K. SALEH, M. JARAGH, O. RAFIQ, « A methodology for the synthesis of communication gateways for network interoperability », *Computer Standards & Interfaces* 17(1995), pp. 193-207.

[SCH 92] J.M. SCHNEIDER, L.F. MACKERT, G. ZORNTLEIN, R.J. VELTHUYS, U. BAR, « An integrated environment for developing communication protocols », *Computer Networks and ISDN Systems*, Vol. 25, pp. 43-61, 1992.

[SCH 82] R. L. SCHWARTZ, P. M. MELLIAR-SMITH, « From state machines to temporal logic: specification methods for protocol standards », *IEEE Transaction on Communication*, N.12, Dec. 1982, pp. 2486-2496.

[SET 93] D. SETLIFF, E. KANT, T. CAIN, « Practical Software Synthesis », *IEEE Software*, Vol. 10, N. 3, May 1993.

[SHI 91] N. SHIRATORI, Y. X. ZHANG, K. TAKAHASHI, S. NOGUCHI, « A User Friendly Software Environment for Protocol Synthesis », *IEEE Transactions on Computers*, Vol.40, No.4, April 1991.

[SID 89] D. SIDHU, T.K. LEUNG, «Formal methods for protocol testing: a detailed study», *IEEE Transaction on Software Engineering*, Vol. 15, N. 4, 1989.

[SUZ 83] I. SUZUKI, T. MURATA, « A Method for Stepwise Refinement and Abstraction of Petri Nets », *Journal of Computer and System Sciences* 27, pp. 51-76, 1983.

[VIS 86] C. A. VISSERS, L. LOGRIPPO, « The importance of the service concept in the design of data communication protocols », *in M.Diaz (Editor), Protocol Specification Testing and Verification*, North-Holland, 1986, pp. 3-17.

[VIS 93] C. A. VISSERS, M. v. SINDEREN, L. F. PIRES, « What Makes Industries Believe in Formal Methods », *Protocol Specification, Testing and Verification, XIII (C-16)*, A. Danthine, G. Leduc and P. Wolper (Editors), Elsevier Science Publishers B. V. (North-Holland), IFIP 1993.

[YAM 95] H. YAMAGUCHI, K. OKANO, T. HIGASHINO, K. TANIGUCHI, « Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers », *Proc. of International Conference on Distributed Computing Systems ICDCS'95*, May 30 - June 2, 1995, Vancouver, Canada.

[YAO 90] Y. W. YAO, W. S. CHEN, M. T. LIU, « A Modular approach to constructing Protocol Converters », *Proc. of INFOCOM'90*, pp 572-579, June 1990.

[YAW 87] Y. YAW, « Analysis and Synthesis of Distributed Systems and Protocols », *Ph.D Dissertation*, dept. EECS, University of California, Berkeley, 1987.

[YEN 91] H. C. YEN, « A Polynomial Time Algorithm to Decide Pairwise Concurrency of Transitions for 1-Bounded Conflict Free Petri Nets », *Information Processing Letters*, 38:71-76, 1991.

[YUA 88] M. C. YUANG, « Survey of Protocol Verification Techniques Based on Finite State Machine Models », *Proc. of Computing Networking Symposium*, pp 164-172, 1988.

[ZAF 80] P. ZAFIROPULO, C. H. WEST, H. RUDIN, D. D. COWAN, D. BRAND, « Towards analyzing and synthesizing protocols », *IEEE Transactions on Communications*, Vol. COM-28, No.4, April 1980, pp. 651-661.

[ZHA 88] Y.X. ZHANG, C. H. WEST, H. RUDIN, D. D. COWAN, D. BRAND, « An Interactive protocol synthesis algorithm using a global state transition graph », *IEEE Transactions on Software Engineering*, Vol. SE-14, No.3, pp. 394-404, March 1988.

[ZOU 91] B. ZOUARI, « PETRISTELLE : transformation et analyse de spécifications Estelle », *Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'91)*, Pau (France), 17-19 Septembre, 1991, Editions Hermes, pp. 89-106.

Annexe A

Agglomération de transitions

Dans cette annexe nous présentons les définitions des post-agglomération et agglomération latérale [BER 86].

Définition 21 (Post-agglomération) *Soit un réseau ordinaire. Un sous-ensemble de transitions G est post-agglomérable à un sous-ensemble de transitions H si et seulement si il existe une place p telle que les quatre conditions suivantes soient satisfaites :*

1. *$\forall g \in G$, la seule entrée de g est p, et p n'est pas une sortie de g,*
2. *une transition au moins de G a au moins une place en sortie,*
3. *$\forall h \in H$, p n'est pas une entrée de h, et p est une sortie de h,*
4. *hormis celles de G et celles de H aucune transition n'est connectée à p.*

■

Définition 22 (Agglomération latérale) *Soit un réseau ordinaire. Deux transitions tg et td sont latéralement agglomérables si et seulement s'il existe deux places pg et pd et une transition te telles que :*

- *ou bien tc précède tg et td et alors les sept conditions suivantes sont vérifiées :*
  1. *tc est la seule transition ayant pd et pg pour sortie,*
  2. *tg et td ont au moins une entrée et une sortie,*
  3. *tg est la seule transition ayant pg en entrée,*
  4. *td est la seule transition ayant pd en entrée,*
  5. *si tg a une sortie alors td n'a pas d'autre entrée que pd et réciproquement,*
  6. *pg et pd ont mêmes marquages initiaux,*
  7. *tg et td ne partagent pas leurs entrées avec d'autres transitions,*
- *ou bien td et tg précèdent tc et il faut reprendre tous les points précédents en échangeant entrées et sorties sauf pour g.*

■

Annexe B

Syntaxe du langage IPNL

```
/*-------------------------------------------------------------*/
/*                                                             */
/*              Grammaire du langage IPNL                      */
/*              Selon la syntaxe YACC                          */
/*              Fichier : IPNLgrammar.y                        */
/*              Auteur: H. Kahlouche                           */
/*              Date: 25/09/1996                               */
/*                                                             */
/*-------------------------------------------------------------*/


%token  /*----- Hots Cles du langage -----*/

IDENTIFICATEUR SERVICE CONST VAR SAP RESOURCE PROCEDURE FUNCTION
READ WRITE INTERACTION BY TRANS FROM TO WHEN PROVIDED DO OUTPUT
TARGET SITE END  ACQUAINTANCE LEQ GEQ NEQ OR AND NOT NOMBRE ON
SET PROTOCOL INTEGER BOOLEAN INIT MACROTRANS BLOCK FOR;

       /*---- Priorite des operateurs -----*/
%left '-' '+'
%left '*' '/'
%left '<' '>' '=' LEQ NEQ GEQ
%left OR
%left AND
%left NOT
%left '.'

%%    /*------  Grammaire IPNL ------*/


specif                  : SERVICE  IDENTIFICATEUR
                            corps_service
                          END'.'
                        | PROTOCOL IDENTIFICATEUR
                            corps_protocole
                          END'.';

corps_service           : declaration_constantes
                          declaration_types
                          declaration_saps
                          declaration_interactions
                          declaration_variables
                          declaration_ressources
                          liste_trans_service
                          marquage_initial
                          architecture_cible
                        ;

corps_protocole         : declaration_accointances
                          declaration_constantes
                          declaration_types
                          declaration_saps
                          declaration_interactions
                          declaration_variables
                          declaration_ressources
                          liste_trans_protocole
                          marquage_initial
                         ;

/* declaration des constantes */

declaration_constantes  : CONST liste_const
                        |;
liste_const             : liste_const const
                        | const
                        ;
const                   : IDENTIFICATEUR '=' NOMBRE ';'
                        ;

/* declaration des types */

declaration_types       : declaration_types dec_type
                        |
                        ;
dec_type                : IDENTIFICATEUR '=' type
                        ;

/* les types */

type                    : INTEGER
                        | BOOLEAN
                        ;

/* Declaration des saps */

declaration_saps        : SAP liste_sap ';'
                        |;
liste_sap               : liste_sap ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;

/* Declaration des interactions */

declaration_interactions: INTERACTION liste_dec_interact END ';'
                        |;
liste_dec_interact      : liste_dec_interact dec_interact
                        | dec_interact
                        ;
dec_interact            : BY liste_id_sap ':' liste_interact
                        ;
liste_id_sap            : liste_id_sap ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;
liste_interact          : liste_interact interact ';'
                        | interact ';'
                        ;
interact                : IDENTIFICATEUR  suite_interact
                        ;
suite_interact          : '('liste_param_interact')'
                        |
                        ;
liste_param_interact    : liste_param_interact ',' param_interact
                        | param_interact
                        ;
param_interact          : IDENTIFICATEUR ':' type
                        ;

/* Declaration des variables */

declaration_variables   : VAR liste_var
                        |
                        ;
liste_var               : liste_var dec_var
                        | dec_var
                        ;
dec_var                 : liste_id_var ':' type init_var ';'
                        ;
init_var                : SET TO IDENTIFICATEUR
```

```
                    |;
liste_id_var        : liste_id_var ',' IDENTIFICATEUR
                    | IDENTIFICATEUR
                    ;

/* Declaration des ressources */

declaration_ressources  : declaration_ressources ressource
                        |;
ressource               : RESOURCE IDENTIFICATEUR
                              liste_operations
                          END ';'
                        ;
liste_operations        : liste_operations operation ';'
                        | operation ';'
                        ;
operation               : READ procedure
                        | WRITE procedure
                        | READ fonction
                        | fonction
                        ;
fonction                : FUNCTION IDENTIFICATEUR suite_fonction
                        ;
suite_fonction          : '(' liste_param_entree ')' ':' type
                        | '(' ')' ':' type
                        ;
procedure               : PROCEDURE IDENTIFICATEUR suite_procedure
                        ;
suite_procedure         : '(' liste_param_entree dec_param_sortie ')'
                        | '(' dec_param_sortie ')'
                        ;
liste_param_entree      : liste_param_entree ';' param
                        | param
;
dec_param_sortie        : '|' liste_param_sortie
                        |;
liste_param_sortie      : liste_param_sortie ';' param
                        | param
                        ;
param                   : liste_idf_param ':' type
                        ;
liste_idf_param         : liste_idf_param ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;


/* Declaration des transitions */

liste_trans_service     : liste_trans_service trans_service
                        | trans_service
                        ;
trans_service           : TRANS IDENTIFICATEUR
                              clause_from
                              clause_to
                              clause_sap
                              clause_when
                              clause_provided
                              clause_do
                              clause_output
                        ;
clause_from             : FROM liste_entrees
                        ;
clause_to               : TO liste_sorties
                        ;
clause_sap              : SAP  IDENTIFICATEUR
                        ;
liste_entrees           : liste_entrees ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;
liste_sorties           : liste_sorties  ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;
clause_when             : WHEN IDENTIFICATEUR suite_when
                        ;
suite_when              : '('liste_arg_reception ')'
                        |;
liste_arg_reception     : liste_arg_reception ',' arg_reception
                        | arg_reception
                        ;
arg_reception           : reference_var
                        | constante
                        ;
clause_provided         : PROVIDED expression_cond
                        |;
clause_do               : DO idf_ressource '.' idf_procedure
                              arg_procedure
                        |;
idf_ressource           : IDENTIFICATEUR
                        ;
idf_procedure           : IDENTIFICATEUR
                        ;
arg_procedure           : '('arg_entree arg_sortie ')'
                        | '('arg_sortie')'
                        ;
arg_entree              : arg_entree ',' terme
                        | terme
                        ;
arg_sortie              : '|' liste_arg_sortie
                        |;
liste_arg_sortie        : liste_arg_sortie ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;
clause_output           : OUTPUT IDENTIFICATEUR  suite_output
                        ;
suite_output            : '('liste_exp_sortie')'
                        |;
liste_exp_sortie        : liste_exp_sortie ',' terme
                        | terme
                        ;

/* Harquage initial */

marquage_initial        : INIT liste_marquages ';'
                        |;
liste_marquages         : liste_marquages ',' idf_place
                        | idf_place
                        ;
idf_place               : IDENTIFICATEUR
                        ;

/* Architecture cible */

architecture_cible      : TARGET liste_sites END
                        |;
liste_sites             : liste_sites site ';'
                        | site ';'
                        ;
site                    : SITE IDENTIFICATEUR
                              allocation_saps
                              allocation_ressources
                          END
                        ;
allocation_saps         : SAP liste_saps_alloues ';'
                        |;
liste_saps_alloues      : liste_saps_alloues ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;
allocation_ressources   : RESOURCE liste_resources_allouees ';'
                        |;
liste_resources_allouees: liste_resources_allouees ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;

/* Termes et Expressions */

terme                   : reference_var
                        | constante
                        | idf_ressource '.' idf_fonction arg_terme
                        ;
reference_var           : IDENTIFICATEUR
                        ;
constante               : NOMBRE
                        ;
idf_fonction            : IDENTIFICATEUR
                        ;
arg_terme               : '('')'
                        | '('liste_arg_terme ')'
                        ;
liste_arg_terme         : liste_arg_terme ',' terme
                        | terme
                        ;
expression_cond         : expression_cond AND expression_cond
                        | expression_cond OR  expression_cond
                        | NOT expression_cond
                        | expression_comp
                        | '(' expression_cond ')'
                        | terme
                        ;
expression_comp         : terme op_comp terme
                        ;
op_comp                 : '<'
                        | '>'
                        | LEQ
                        | GEQ
```

```
                        | '='
                        | NEQ
                        ;

/* Declaration des aquaintances */

declaration_accointances: ACQUAINTANCE liste_entites ';'
                        ;
liste_entites           : liste_entites ',' IDENTIFICATEUR
                        | IDENTIFICATEUR
                        ;

/* Les transitions de protocole */

liste_trans_protocole   : liste_trans_protocole trans_protocole
                        |;
trans_protocole         : trans_simple
                        | macrotransition
                        ;
trans_simple            : TRANS IDENTIFICATEUR
                              clause_from
                              clause_to
                              clause_when suite_when
                              clause_provided
                              clause_do
                              clause_output suite_output
                        ;
suite_when              : ON idf_sap
                        | FROM accointance
                        ;
suite_output            : ON idf_sap
                        | TO accointance
                        ;
idf_sap                 : IDENTIFICATEUR
                        ;
accointance             : IDENTIFICATEUR
                        ;
macrotransition         : MACROTRANS IDENTIFICATEUR
                              clause_from
                              clause_to
                              bloc
                        ;
bloc                    : BLOCK IDENTIFICATEUR FOR idf_trans
                              declaration_constantes
                              declaration_variables
                              liste_trans_protocole
                           END
                        ;
idf_trans               : IDENTIFICATEUR
                        ;
%%              /*--------------- Fin Grammaire ---------------*/

/***
Supprimer toutes les regles qui produisent ": IDENTIFICATEUR"
pour eliminer les conflits "shift/reduces". Ces regles ont ete
introduites pour des raisons de clarte de la grammaire.
***/
```

Annexe C

Expérimentations

C.1 Synthèse de traitements répartis

```
/*************************************************************
      Specifications du protocole 'RPC_avec_HaJ'
      derive a partir du service  'RPC_avec_HaJ.ipn'
      par STEPS (V 1.0)
*************************************************************/


/*-------------------------*/
/* Entite de Protocole PE_0 */
/*-------------------------*/
PROTOCOL PE_0
  ACQUAINTANCE PE_1,PE_2;
  SAP a;
  VAR
    x:INTEGER;
    y:INTEGER;
    z:INTEGER;
  MACROTRANS t_6
    FROM etat_5
    TO etat_5
  BLOCK B_4 FOR t_6
  MACROTRANS ts0_11
  BLOCK B_7 FOR ts0_11
  MACROTRANS ts1_12
  BLOCK B_8 FOR ts1_12
  CONST
    HSG_ID_13 = 13;
    TRANS_ID_0 = 0;
    TERM_ID_2 = 2;
    TERM_ID_3 = 3;
    HSG_ID_4 = 4;
    HSG_ID_17 = 17;
    TERM_ID_4 = 4;
  TRANS noname_13
    OUTPUT send(HSG_ID_13, TRANS_ID_0, TERM_ID_2, x, TERM_ID_3, y)
           TO PE_2
  TRANS noname_25
    OUTPUT send(HSG_ID_4, TRANS_ID_0) TO PE_2
  TRANS noname_51
    TO noname_56
    WHEN receive(HSG_ID_17,TRANS_ID_0,TERM_ID_4,z) FROM PE_2
  TRANS ts7_55
    FROM noname_56
  END /*block B_8*/
  END /*block B_7*/
  END /*block B_4*/
INIT etat_5;
END.
/*-------------------------*/
/* Entite de Protocole PE_1 */
/*-------------------------*/
PROTOCOL PE_1
  ACQUAINTANCE PE_0,PE_2;
  SAP   VAR
    x:INTEGER;
    y:INTEGER;
    z:INTEGER;
  RESOURCE Serveur
    WRITE PROCEDURE P(x:INTEGER, y:INTEGER | z:INTEGER);
  END;
  MACROTRANS t_8
    FROM etat_7
    TO etat_7
  BLOCK B_5 FOR t_8
  MACROTRANS noname_57
  BLOCK B_10 FOR noname_57
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_0 = 0;
    TERM_ID_2 = 2;
    TERM_ID_3 = 3;
  TRANS noname_37
    TO noname_44
    WHEN receive(HSG_ID_16,TRANS_ID_0,TERM_ID_2,x,TERM_ID_3,y)
         FROM PE_2
  MACROTRANS ts3_45
    FROM noname_44
  BLOCK B_11 FOR ts3_45
  CONST
    HSG_ID_7 = 7;
  TRANS noname_59
    TO noname_60
    DO Serveur.P(x, y|z)
  TRANS noname_61
    FROM noname_60
    OUTPUT send(HSG_ID_7, TRANS_ID_0) TO PE_2
  END /*block B_11*/
  END /*block B_10*/
  END /*block B_5*/
INIT etat_7;
END.
/*-------------------------*/
/* Entite de Protocole PE_2 */
/*-------------------------*/
PROTOCOL PE_2
  ACQUAINTANCE PE_0,PE_1;
  SAP   VAR
    x:INTEGER;
    y:INTEGER;
    z:INTEGER;
  RESOURCE Serveur
    WRITE PROCEDURE P(x:INTEGER, y:INTEGER | z:INTEGER);
  END;
  MACROTRANS t_10
    FROM etat_9
    TO etat_9
  BLOCK B_6 FOR t_10
  MACROTRANS noname_58
  BLOCK B_9 FOR noname_58
  CONST
    HSG_ID_13 = 13;
    TRANS_ID_0 = 0;
    TERM_ID_2 = 2;
    TERM_ID_3 = 3;
    HSG_ID_4 = 4;
    HSG_ID_17 = 17;
    TERM_ID_4 = 4;
    HSG_ID_16 = 16;
    HSG_ID_7 = 7;
  TRANS noname_18
    TO noname_30
    WHEN receive(HSG_ID_13,TRANS_ID_0,TERM_ID_2,x,TERM_ID_3,y)
         FROM PE_0
  TRANS noname_27
```

```
    TO noname_31
    WHEN receive(HSG_ID_4,TRANS_ID_0) FROM PE_0
  TRANS noname_29
    FROM noname_30,noname_31
    TO noname_46
    DO Serveur.P(x, y|z)
  TRANS noname_32
    FROM noname_48
    OUTPUT send(HSG_ID_17, TRANS_ID_0, TERM_ID_4, z) TO PE_0
  TRANS noname_35
    FROM noname_46
    TO noname_47
    OUTPUT send(HSG_ID_16, TRANS_ID_0, TERM_ID_2, x, TERM_ID_3, y)
           TO PE_1
  TRANS noname_49
    FROM noname_47
    TO noname_48
    WHEN receive(HSG_ID_7,TRANS_ID_0) FROM PE_1
  END /*block B_9*/
  END /*block B_6*/
INIT etat_9;
END.
```

C.2 Synthèse du flot de contrôle

```
/*****************************************************************
      Specifications du protocole 'Ordonnancement_de_taches'
      derive a partir du service  'Ordonnancement_de_taches.ipn'
      par STEPS (V 1.0)
*****************************************************************/


/*--------------------------*/
/* Entite de Protocole PE_0 */
/*--------------------------*/
PROTOCOL PE_0
  ACQUAINTANCE PE_1,PE_2;
  SAP a;
  HACROTRANS tache1_15
    FROM etat1_10
    TO etat2_11,etat3_12
  BLOCK B_4 FOR tache1_15
  HACROTRANS ts0_40
  BLOCK B_19 FOR ts0_40
  HACROTRANS ts7_45
  BLOCK B_20 FOR ts7_45
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_0 = 0;
  TRANS noname_46
    OUTPUT send(HSG_ID_1, TRANS_ID_0) TO PE_1
  TRANS noname_54
    OUTPUT send(HSG_ID_1, TRANS_ID_0) TO PE_2
  END /*block B_20*/
  END /*block B_19*/
  END /*block B_4*/
  TRANS tache2_16
    FROM etat2_11
    TO etat4_13
  TRANS tache3_17
    FROM etat2_11
    TO etat4_13
  TRANS tache4_18
    FROM etat3_12
    TO etat5_14
  HACROTRANS tache5_19
    FROM etat4_13,etat5_14
    TO etat1_10
  BLOCK B_16 FOR tache5_19
  HACROTRANS noname_98
  BLOCK B_33 FOR noname_98
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_9 = 9;
  TRANS noname_92
    TO noname_95
    WHEN receive(HSG_ID_1,TRANS_ID_9) FROM PE_2
  TRANS ts9_96
    FROM noname_95
  END /*block B_33*/
  END /*block B_16*/
INIT etat1_10;
END.
/*--------------------------*/
/* Entite de Protocole PE_1 */
/*--------------------------*/
PROTOCOL PE_1
  ACQUAINTANCE PE_0,PE_2;
  SAP b;
  HACROTRANS tache1_25
    FROM etat1_20
    TO etat2_21,etat3_22
  BLOCK B_5 FOR tache1_25
  HACROTRANS noname_61
  BLOCK B_21 FOR noname_61
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_0 = 0;
  TRANS noname_49
    TO noname_52
    WHEN receive(HSG_ID_1,TRANS_ID_0) FROM PE_0
  TRANS ts9_53
    FROM noname_52
  END /*block B_21*/
  END /*block B_5*/
  HACROTRANS tache2_26
    FROM etat2_21
    TO etat4_23
  BLOCK B_8 FOR tache2_26
  HACROTRANS ts0_41
  BLOCK B_23 FOR ts0_41
  HACROTRANS ts7_63
  BLOCK B_24 FOR ts7_63
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_4 = 4;
  TRANS noname_64
    OUTPUT send(HSG_ID_1, TRANS_ID_4) TO PE_2
  END /*block B_24*/
  END /*block B_23*/
  END /*block B_8*/
  HACROTRANS tache3_27
    FROM etat2_21
    TO etat4_23
  BLOCK B_11 FOR tache3_27
  HACROTRANS ts0_42
  BLOCK B_26 FOR ts0_42
  HACROTRANS ts7_74
  BLOCK B_27 FOR ts7_74
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_6 = 6;
  TRANS noname_75
    OUTPUT send(HSG_ID_1, TRANS_ID_6) TO PE_2
  END /*block B_27*/
  END /*block B_26*/
  END /*block B_11*/
  TRANS tache4_28
    FROM etat3_22
    TO etat5_24
  TRANS tache5_29
    FROM etat4_23,etat5_24
    TO etat1_20
INIT etat1_20;
END.
/*--------------------------*/
/* Entite de Protocole PE_2 */
/*--------------------------*/
PROTOCOL PE_2
  ACQUAINTANCE PE_0,PE_1;
  SAP c;
  HACROTRANS tache1_35
    FROM etat1_30
    TO etat2_31,etat3_32
  BLOCK B_6 FOR tache1_35
  HACROTRANS noname_62
  BLOCK B_22 FOR noname_62
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_0 = 0;
  TRANS noname_55
    TO noname_58
    WHEN receive(HSG_ID_1,TRANS_ID_0) FROM PE_0
  TRANS ts9_59
    FROM noname_58
  END /*block B_22*/
  END /*block B_6*/
  HACROTRANS tache2_36
    FROM etat2_31
```

```
    TO etat4_33
  BLOCK B_9 FOR tache2_36
  HACROTRANS noname_73
  BLOCK B_25 FOR noname_73
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_4 = 4;
  TRANS noname_67
    TO noname_70
    WHEN receive(HSG_ID_1,TRANS_ID_4) FROM PE_1
  TRANS ts9_71
    FROM noname_70
  END /*block B_25*/
  END /*block B_9*/
  HACROTRANS tache3_37
    FROM etat2_31
    TO etat4_33
  BLOCK B_12 FOR tache3_37
  HACROTRANS noname_84
  BLOCK B_28 FOR noname_84
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_6 = 6;
  TRANS noname_78
    TO noname_81
    WHEN receive(HSG_ID_1,TRANS_ID_6) FROM PE_1
  TRANS ts9_82
    FROM noname_81
  END /*block B_28*/
  END /*block B_12*/
  TRANS tache4_38
    FROM etat3_32
    TO etat5_34
  HACROTRANS tache5_39
    FROM etat4_33,etat5_34
    TO etat1_30
  BLOCK B_18 FOR tache5_39
  HACROTRANS ts0_44
  BLOCK B_31 FOR ts0_44
  HACROTRANS ts7_88
  BLOCK B_32 FOR ts7_88
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_9 = 9;
  TRANS noname_89
    OUTPUT send(HSG_ID_1, TRANS_ID_9) TO PE_0
  END /*block B_32*/
  END /*block B_31*/
  END /*block B_18*/
INIT etat1_30;
END.
```

C.3 Synthèse du flot de données par variables

```
/*****************************************************************
       Specifications du protocole 'Connexion_transport'
       derive a partir du service  'Connexion_transport.ipn'
       par STEPS (V 1.0)
*****************************************************************/


/*---------------------------*/
/* Entite de Protocole PE_0 */
/*---------------------------*/
PROTOCOL PE_0
  ACQUAINTANCE PE_1;
  SAP a;
  VAR
    qos:INTEGER;
  HACROTRANS requete_7
    FROM etat1_5
    TO etat2_6
  BLOCK B_3 FOR requete_7
  HACROTRANS ts0_13
  BLOCK B_7 FOR ts0_13
  TRANS noname_15
    TO noname_17
    WHEN TConnectRequest(qos) ON a
  HACROTRANS ts5_16
    FROM noname_17
  BLOCK B_8 FOR ts5_16
  CONST
    HSG_ID_3 = 3;
    TRANS_ID_0 = 0;
    TERM_ID_3 = 3;
  TRANS noname_18
    OUTPUT send(HSG_ID_3, TRANS_ID_0, TERM_ID_3, qos) TO PE_1
  END /*block B_8*/
  END /*block B_7*/
  END /*block B_3*/
  HACROTRANS indication_8
    FROM etat2_6
    TO etat1_5
  BLOCK B_5 FOR indication_8
  HACROTRANS noname_42
  BLOCK B_13 FOR noname_42
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_4 = 4;
  TRANS noname_36
    TO noname_39
    WHEN receive(HSG_ID_1,TRANS_ID_4) FROM PE_1
  TRANS ts9_40
    FROM noname_39
  END /*block B_13*/
  END /*block B_5*/
INIT etat1_5;
END.
/*---------------------------*/
/* Entite de Protocole PE_1 */
/*---------------------------*/
PROTOCOL PE_1
  ACQUAINTANCE PE_0;
  SAP b;
  VAR
    qos:INTEGER;
  HACROTRANS requete_11
    FROM etat1_9
    TO etat2_10
  BLOCK B_4 FOR requete_11
  HACROTRANS noname_28
  BLOCK B_9 FOR noname_28
  CONST
    HSG_ID_3 = 3;
    TRANS_ID_0 = 0;
    TERM_ID_3 = 3;
  TRANS noname_22
    WHEN receive(HSG_ID_3,TRANS_ID_0,TERM_ID_3,qos) FROM PE_0
  END /*block B_9*/
  END /*block B_4*/
  HACROTRANS indication_12
    FROM etat2_10
    TO etat1_9
  BLOCK B_6 FOR indication_12
  HACROTRANS ts0_14
  BLOCK B_10 FOR ts0_14
  HACROTRANS ts2_29
  BLOCK B_11 FOR ts2_29
  TRANS noname_30
    TO noname_32
    OUTPUT TConnectIndication(qos) ON b
  HACROTRANS ts7_31
    FROM noname_32
  BLOCK B_12 FOR ts7_31
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_4 = 4;
  TRANS noname_33
    OUTPUT send(HSG_ID_1, TRANS_ID_4) TO PE_0
  END /*block B_12*/
  END /*block B_11*/
  END /*block B_10*/
  END /*block B_6*/
INIT etat1_9;
END.
```

C.4 Synthèse du flot de données par ressources

```
/*****************************************************************
       Specifications du protocole 'Flot_de_donnees'
       derive a partir du service  'Flot_de_donnees.ipn'
       par STEPS (V 1.0)
*****************************************************************/
```

```
/*---------------------------*/
/* Entite de Protocole PE_0 */
/*---------------------------*/
PROTOCOL PE_0
  ACQUAINTANCE PE_1,PE_2;
  SAP a;
  RESOURCE R3
    READ PROCEDURE p( | y:INTEGER);
  END;
  MACROTRANS t1_11
    FROM etat1_9
    TO etat2_10
  BLOCK B_4 FOR t1_11
  MACROTRANS ts0_21
  BLOCK B_10 FOR ts0_21
  MACROTRANS ts6_23
  BLOCK B_11 FOR ts6_23
  CONST
    HSG_ID_2 = 2;
    TRANS_ID_0 = 0;
    PE_ID_0 = 0;
    TERH_ID_4 = 4;
    HSG_ID_3 = 3;
    PE_ID_1 = 1;
    TERH_ID_5 = 5;
    TERH_ID_7 = 7;
  VAR
    V4:INTEGER;
    V7:INTEGER;
  TRANS noname_24
    OUTPUT send(HSG_ID_2, TRANS_ID_0, PE_ID_0, TERH_ID_4) TO PE_1
  TRANS noname_37
    WHEN receive(HSG_ID_3,TRANS_ID_0,TERH_ID_4,V4) FROM PE_1
  TRANS noname_40
    OUTPUT send(HSG_ID_2, TRANS_ID_0, PE_ID_1, TERH_ID_5) TO PE_2
  TRANS noname_54
    OUTPUT send(HSG_ID_2, TRANS_ID_0, PE_ID_0, TERH_ID_7) TO PE_1
  TRANS noname_61
    WHEN receive(HSG_ID_3,TRANS_ID_0,TERH_ID_7,V7) FROM PE_1
  END /*block B_11*/
  END /*block B_10*/
  END /*block B_4*/
  MACROTRANS t2_12
    FROM etat2_10
    TO etat1_9
  BLOCK B_7 FOR t2_12
  MACROTRANS ts0_22
  BLOCK B_15 FOR ts0_22
  TRANS noname_68
    TO noname_70
    PROVIDED V4>=V7
  MACROTRANS ts1_69
    FROM noname_70
  BLOCK B_16 FOR ts1_69
  TRANS noname_71
    TO noname_73
    DO R3.p(|y)
  TRANS ts7_72
    FROM noname_73
  END /*block B_16*/
  END /*block B_15*/
  END /*block B_7*/
INIT etat1_9;
END.
/*---------------------------*/
/* Entite de Protocole PE_1 */
/*---------------------------*/
PROTOCOL PE_1
  ACQUAINTANCE PE_0,PE_2;
  RESOURCE R1
    READ FUNCTION f():INTEGER;
    READ FUNCTION g(x:INTEGER):INTEGER;
  END;
  MACROTRANS t1_15
    FROM etat1_13
    TO etat2_14
  BLOCK B_5 FOR t1_15
  MACROTRANS noname_64
  BLOCK B_12 FOR noname_64
  CONST
    HSG_ID_2 = 2;
    TRANS_ID_0 = 0;
    PE_ID_0 = 0;
    TERH_ID_4 = 4;
    HSG_ID_3 = 3;
    TERH_ID_5 = 5;
    TERH_ID_7 = 7;
  VAR
    V5:INTEGER;
  TRANS noname_29
    TO noname_36
    WHEN receive(HSG_ID_2,TRANS_ID_0,PE_ID_0,TERH_ID_4) FROM PE_0
  TRANS noname_30
    FROM noname_36
    OUTPUT send(HSG_ID_3, TRANS_ID_0, TERH_ID_4, R1.f()) TO PE_0
  TRANS noname_51
    TO noname_60
    WHEN receive(HSG_ID_3,TRANS_ID_0,TERH_ID_5,V5) FROM PE_2
  TRANS noname_56
    TO noname_59
    WHEN receive(HSG_ID_2,TRANS_ID_0,PE_ID_0,TERH_ID_7) FROM PE_0
  TRANS noname_57
    FROM noname_59,noname_60
    OUTPUT send(HSG_ID_3, TRANS_ID_0, TERH_ID_7, R1.g(V5)) TO PE_0
  END /*block B_12*/
  END /*block B_5*/
  TRANS t2_16
    FROM etat2_14
    TO etat1_13
INIT etat1_13;
END.
/*---------------------------*/
/* Entite de Protocole PE_2 */
/*---------------------------*/
PROTOCOL PE_2
  ACQUAINTANCE PE_0,PE_1;
  RESOURCE R2
    READ FUNCTION h():INTEGER;
  END;
  MACROTRANS t1_19
    FROM etat1_17
    TO etat2_18
  BLOCK B_6 FOR t1_19
  MACROTRANS noname_65
  BLOCK B_13 FOR noname_65
  CONST
    HSG_ID_2 = 2;
    TRANS_ID_0 = 0;
    PE_ID_1 = 1;
    TERH_ID_5 = 5;
    HSG_ID_3 = 3;
  TRANS noname_43
    TO noname_50
    WHEN receive(HSG_ID_2,TRANS_ID_0,PE_ID_1,TERH_ID_5) FROM PE_0
  TRANS noname_44
    FROM noname_50
    OUTPUT send(HSG_ID_3, TRANS_ID_0, TERH_ID_5, R2.h()) TO PE_1
  END /*block B_13*/
  END /*block B_6*/
  TRANS t2_20
    FROM etat2_18
    TO etat1_17
INIT etat1_17;
END.
```

C.5 Synthèse du contrôle de la cohérence

```
/*************************************************************
      Specifications du protocole 'Acces_concurrents'
      derive a partir du service  'Acces_concurrents.ipn'
      par STEPS (V 1.0)
*************************************************************/
/*---------------------------*/
/* Entite de Protocole PE_0 */
/*---------------------------*/
PROTOCOL PE_0
  ACQUAINTANCE PE_1;
  SAP a;
  VAR
    x:INTEGER;
  RESOURCE R1
    READ FUNCTION Size():INTEGER;
    WRITE PROCEDURE Append(x:INTEGER);
  END;
  MACROTRANS t1_14
    FROM etat1_10
    TO etat2_11
  BLOCK B_3 FOR t1_14
  MACROTRANS ts0_26
```

```
  BLOCK B_11 FOR ts0_26
  TRANS noname_30
    TO noname_32
    WHEN GetValue(x) ON a
  HACROTRANS ts7_31
    FROH noname_32
  BLOCK B_12 FOR ts7_31
  HACROTRANS ts9_33
  BLOCK B_13 FOR ts9_33
  CONST
    HSG_ID_9 = 9;
    TRANS_ID_4 = 4;
    HSG_ID_8 = 8;
  TRANS noname_35
    OUTPUT send(HSG_ID_9, TRANS_ID_4)
  TRANS noname_38
    WHEN receive(HSG_ID_8,TRANS_ID_4)
  END /*block B_13*/
  END /*block B_12*/
  END /*block B_11*/
  END /*block B_3*/
  HACROTRANS t2_15
    FROH etat2_11
    TO etat1_10
  BLOCK B_5 FOR t2_15
  HACROTRANS ts0_27
  BLOCK B_14 FOR ts0_27
  HACROTRANS ts1_40
  BLOCK B_15 FOR ts1_40
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_4 = 4;
    TERH_ID_3 = 3;
    HSG_ID_7 = 7;
  TRANS noname_41
    TO noname_54
    DO R1.Append(x)
  HACROTRANS ts2_42
    FROH noname_58
  BLOCK B_18 FOR ts2_42
  TRANS noname_64
    TO noname_66
    OUTPUT SizeIndication(R1.Size()) ON a
  HACROTRANS ts7_65
    FROH noname_66
  BLOCK B_19 FOR ts7_65
  HACROTRANS ts8_68
  BLOCK B_20 FOR ts8_68
  CONST
    HSG_ID_10 = 10;
  TRANS noname_69
    OUTPUT send(HSG_ID_10, TRANS_ID_4) TO PE_1
  END /*block B_20*/
  END /*block B_19*/
  END /*block B_18*/
  TRANS noname_43
    FROH noname_54
    TO noname_57
    OUTPUT send(HSG_ID_16, TRANS_ID_4, TERH_ID_3, x) TO PE_1
  TRANS noname_55
    FROH noname_57
    TO noname_58
    WHEN receive(HSG_ID_7,TRANS_ID_4) FROH PE_1
  END /*block B_15*/
  END /*block B_14*/
  END /*block B_5*/
  TRANS t3_16
    FROH etat3_12
    TO etat4_13
  TRANS t4_17
    FROH etat4_13
    TO etat3_12
INIT etat1_10,etat2_11;
END.
/*---------------------------*/
/* Entite de Protocole PE_1 */
/*---------------------------*/
PROTOCOL PE_1
  ACQUAINTANCE PE_0;
  SAP b;
  VAR
    x:INTEGER;
  RESOURCE R1
    READ FUNCTION Size():INTEGER;
    WRITE PROCEDURE Append(x:INTEGER);
  END;
  TRANS t1_22
    FROH etat1_18
    TO etat2_19
  HACROTRANS t2_23
    FROH etat2_19
    TO etat1_18
  BLOCK B_6 FOR t2_23
  HACROTRANS noname_59
  BLOCK B_16 FOR noname_59
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_4 = 4;
    TERH_ID_3 = 3;
  TRANS noname_47
    TO noname_52
    WHEN receive(HSG_ID_16,TRANS_ID_4,TERH_ID_3,x) FROH PE_0
  HACROTRANS ts3_53
    FROH noname_52
  BLOCK B_17 FOR ts3_53
  CONST
    HSG_ID_7 = 7;
  TRANS noname_60
    TO noname_61
    DO R1.Append(x)
  TRANS noname_62
    FROH noname_61
    OUTPUT send(HSG_ID_7, TRANS_ID_4) TO PE_0
  END /*block B_17*/
  END /*block B_16*/
  END /*block B_6*/
  HACROTRANS t3_24
    FROH etat3_20
    TO etat4_21
  BLOCK B_8 FOR t3_24
  HACROTRANS ts0_28
  BLOCK B_21 FOR ts0_28
  TRANS noname_71
    TO noname_73
    WHEN GetSize ON b
  HACROTRANS ts7_72
    FROH noname_73
  BLOCK B_22 FOR ts7_72
  HACROTRANS ts9_74
  BLOCK B_23 FOR ts9_74
  CONST
    HSG_ID_9 = 9;
    TRANS_ID_9 = 9;
    HSG_ID_8 = 8;
  TRANS noname_76
    OUTPUT send(HSG_ID_9, TRANS_ID_9)
  TRANS noname_79
    WHEN receive(HSG_ID_8,TRANS_ID_9)
  END /*block B_23*/
  END /*block B_22*/
  END /*block B_21*/
  END /*block B_8*/
  HACROTRANS t4_25
    FROH etat4_21
    TO etat3_20
  BLOCK B_10 FOR t4_25
  HACROTRANS ts0_29
  BLOCK B_24 FOR ts0_29
  HACROTRANS ts2_81
  BLOCK B_25 FOR ts2_81
  TRANS noname_82
    TO noname_84
    OUTPUT SizeIndication(R1.Size()) ON b
  HACROTRANS ts7_83
    FROH noname_84
  BLOCK B_26 FOR ts7_83
  HACROTRANS ts8_86
  BLOCK B_27 FOR ts8_86
  CONST
    HSG_ID_10 = 10;
    TRANS_ID_9 = 9;
  TRANS noname_87
    OUTPUT send(HSG_ID_10, TRANS_ID_9) TO PE_0
  END /*block B_27*/
  END /*block B_26*/
  END /*block B_25*/
  END /*block B_24*/
  END /*block B_10*/
INIT etat1_18,etat2_19;
END.
```

Annexe D

Application : Saisie de données dans un environnement réparti

```
/*************************************************************
       Specifications du protocole
       derive a partir du service 'Data_acquisition_service'
       par STEPS (V 1.0)
*************************************************************/


/*--------------------------*/
/* Entite de Protocole PE_0 */
/*--------------------------*/
PROTOCOL PE_0
  ACQUAINTANCE PE_1,PE_2;
  SAP a;
  VAR
    x:INTEGER;
    y:INTEGER;
  RESOURCE R1
    READ FUNCTION size():INTEGER;
    WRITE PROCEDURE append(x:INTEGER);
  END;
  MACROTRANS t1_22
    FROM etat0_16
    TO etat1_17
  BLOCK B_4 FOR t1_22
  MACROTRANS ts0_52
  BLOCK B_22 FOR ts0_52
  TRANS noname_58
    TO noname_60
    WHEN GetValue(x) ON a
  MACROTRANS ts7_59
    FROM noname_60
  BLOCK B_23 FOR ts7_59
  MACROTRANS ts9_61
  BLOCK B_24 FOR ts9_61
  CONST
    HSG_ID_9 = 9;
    TRANS_ID_4 = 4;
    HSG_ID_8 = 8;
    HSG_ID_2 = 2;
    TRANS_ID_0 = 0;
    PE_ID_0 = 0;
    TERM_ID_7 = 7;
    HSG_ID_3 = 3;
    TRANS_ID_8 = 8;
  VAR
    V7:INTEGER;
  TRANS noname_63
    OUTPUT send(HSG_ID_9, TRANS_ID_4)
  TRANS noname_66
    TO noname_84
    WHEN receive(HSG_ID_8,TRANS_ID_4)
  TRANS noname_68
    FROM noname_84
    OUTPUT send(HSG_ID_2, TRANS_ID_0, PE_ID_0, TERM_ID_7) TO PE_2
  TRANS noname_81
    WHEN receive(HSG_ID_3,TRANS_ID_0,TERM_ID_7,V7) FROM PE_2
  TRANS noname_85
    OUTPUT send(HSG_ID_9, TRANS_ID_8)
  TRANS noname_87
    WHEN receive(HSG_ID_8,TRANS_ID_8)
  END /*block B_24*/
  END /*block B_23*/
  END /*block B_22*/
  END /*block B_4*/
  MACROTRANS t2_23
    FROM etat1_17
    TO etat2_18
  BLOCK B_7 FOR t2_23
  MACROTRANS ts0_53
  BLOCK B_26 FOR ts0_53
  CONST
    HSG_ID_10 = 10;
    TRANS_ID_8 = 8;
  TRANS noname_89
    TO noname_94
    PROVIDED R1.size()>V7
  MACROTRANS ts1_90
    FROM noname_95
  BLOCK B_27 FOR ts1_90
  CONST
    HSG_ID_3 = 3;
    TRANS_ID_4 = 4;
    TERM_ID_3 = 3;
    HSG_ID_4 = 4;
    HSG_ID_17 = 17;
  TRANS noname_96
    OUTPUT send(HSG_ID_3, TRANS_ID_4, TERM_ID_3, x) TO PE_2
  TRANS noname_105
    OUTPUT send(HSG_ID_4, TRANS_ID_4) TO PE_2
  TRANS noname_128
    TO noname_131
    WHEN receive(HSG_ID_17,TRANS_ID_4) FROM PE_2
  MACROTRANS ts7_130
    FROM noname_131
  BLOCK B_31 FOR ts7_130
  CONST
    HSG_ID_1 = 1;
  TRANS noname_138
    OUTPUT send(HSG_ID_1, TRANS_ID_4) TO PE_2
  MACROTRANS ts8_145
  BLOCK B_33 FOR ts8_145
  TRANS noname_147
    OUTPUT send(HSG_ID_10, TRANS_ID_4) TO PE_1
  END /*block B_33*/
  END /*block B_31*/
  END /*block B_27*/
  TRANS noname_91
    FROM noname_94
    TO noname_95
    OUTPUT send(HSG_ID_10, TRANS_ID_8) TO PE_1
  END /*block B_26*/
  END /*block B_7*/
  MACROTRANS t3_24
    FROM etat1_17
    TO etat2_18
  BLOCK B_10 FOR t3_24
  MACROTRANS ts0_54
  BLOCK B_34 FOR ts0_54
  CONST
    HSG_ID_10 = 10;
```

```
    TRANS_ID_4 = 4;
  TRANS noname_148
    TO noname_153
    PROVIDED R1.size()<=V7
  HACROTRANS ts1_149
    FROH noname_154
  BLOCK B_35 FOR ts1_149
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_8 = 8;
    TERH_ID_3 = 3;
    HSG_ID_7 = 7;
  TRANS noname_155
    TO noname_168,noname_181
    DO R1.append(x)
  HACROTRANS ts7_156
    FROH noname_172,noname_184
  BLOCK B_40 FOR ts7_156
  CONST
    HSG_ID_1 = 1;
  TRANS noname_195
    OUTPUT send(HSG_ID_1, TRANS_ID_8) TO PE_2
  HACROTRANS ts8_202
  BLOCK B_42 FOR ts8_202
  TRANS noname_204
    OUTPUT send(HSG_ID_10, TRANS_ID_8) TO PE_1
  END /*block B_42*/
  END /*block B_40*/
  TRANS noname_157
    FROH noname_168
    TO noname_171
    OUTPUT send(HSG_ID_16, TRANS_ID_8, TERH_ID_3, x) TO PE_1
  TRANS noname_169
    FROH noname_171
    TO noname_172
    WHEN receive(HSG_ID_7,TRANS_ID_8) FROH PE_1
  TRANS noname_173
    FROH noname_181
    TO noname_183
    OUTPUT send(HSG_ID_16, TRANS_ID_8, TERH_ID_3, x) TO PE_2
  TRANS noname_182
    FROH noname_183
    TO noname_184
    WHEN receive(HSG_ID_7,TRANS_ID_8) FROH PE_2
  END /*block B_35*/
  TRANS noname_150
    FROH noname_153
    TO noname_154
    OUTPUT send(HSG_ID_10, TRANS_ID_4) TO PE_1
  END /*block B_34*/
  END /*block B_10*/
  HACROTRANS t4_25
    FROH etat2_18,etat3_19
    TO etat0_16,etat5_20
  BLOCK B_13 FOR t4_25
  HACROTRANS noname_224
  BLOCK B_46 FOR noname_224
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_9 = 9;
  TRANS noname_212
    TO noname_215
    WHEN receive(HSG_ID_1,TRANS_ID_9) FROH PE_2
  TRANS ts9_216
    FROH noname_215
  END /*block B_46*/
  END /*block B_13*/
  HACROTRANS t5_26
    FROH etat4_21
    TO etat3_19
  BLOCK B_16 FOR t5_26
  HACROTRANS noname_259
  BLOCK B_50 FOR noname_259
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_12 = 12;
    TERH_ID_14 = 14;
  TRANS noname_235
    TO noname_240
    WHEN receive(HSG_ID_16,TRANS_ID_12,TERH_ID_14,y) FROH PE_1
  HACROTRANS ts3_241
    FROH noname_240
  BLOCK B_52 FOR ts3_241
  CONST
    HSG_ID_7 = 7;
  TRANS noname_261
    TO noname_262
    DO R1.append(y)
  TRANS noname_263
    FROH noname_262
    OUTPUT send(HSG_ID_7, TRANS_ID_12) TO PE_1
  END /*block B_52*/
  END /*block B_50*/
  END /*block B_16*/
  TRANS t6_27
    FROH etat5_20
    TO etat4_21
INIT etat0_16,etat5_20;
END.
/*-------------------------*/
/* Entite de Protocole PE_1 */
/*-------------------------*/
PROTOCOL PE_1
  ACQUAINTANCE PE_0,PE_2;
  SAP b;
  VAR
    y:INTEGER;
    x:INTEGER;
  RESOURCE R1
    READ FUNCTION size():INTEGER;
    WRITE PROCEDURE append(x:INTEGER);
  END;
  RESOURCE R2
    READ FUNCTION size():INTEGER;
    WRITE PROCEDURE append(x:INTEGER);
  END;
  TRANS t1_34
    FROH etat0_28
    TO etat1_29
  HACROTRANS t2_35
    FROH etat1_29
    TO etat2_30
  BLOCK B_8 FOR t2_35
  HACROTRANS noname_132
  BLOCK B_29 FOR noname_132
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_4 = 4;
    TERH_ID_3 = 3;
  TRANS noname_116
    TO noname_121
    WHEN receive(HSG_ID_16,TRANS_ID_4,TERH_ID_3,x) FROH PE_2
  HACROTRANS ts3_122
    FROH noname_121
  BLOCK B_30 FOR ts3_122
  CONST
    HSG_ID_7 = 7;
  TRANS noname_134
    TO noname_135
    DO R2.append(x)
  TRANS noname_136
    FROH noname_135
    OUTPUT send(HSG_ID_7, TRANS_ID_4) TO PE_2
  END /*block B_30*/
  END /*block B_29*/
  END /*block B_8*/
  HACROTRANS t3_36
    FROH etat1_29
    TO etat2_30
  BLOCK B_11 FOR t3_36
  HACROTRANS noname_185
  BLOCK B_36 FOR noname_185
  CONST
    HSG_ID_16 = 16;
    TRANS_ID_8 = 8;
    TERH_ID_3 = 3;
  TRANS noname_161
    TO noname_166
    WHEN receive(HSG_ID_16,TRANS_ID_8,TERH_ID_3,x) FROH PE_0
  HACROTRANS ts3_167
    FROH noname_166
  BLOCK B_38 FOR ts3_167
  CONST
    HSG_ID_7 = 7;
  TRANS noname_187
    TO noname_188
    DO R1.append(x)
  TRANS noname_189
    FROH noname_188
    OUTPUT send(HSG_ID_7, TRANS_ID_8) TO PE_0
  END /*block B_38*/
  END /*block B_36*/
  END /*block B_11*/
  HACROTRANS t4_37
```

```
  FROH etat2_30,etat3_31
  TO etat0_28,etat5_32
BLOCK B_14 FOR t4_37
HACROTRANS noname_225
BLOCK B_47 FOR noname_225
CONST
  HSG_ID_1 = 1;
  TRANS_ID_9 = 9;
TRANS noname_218
  TO noname_221
  WHEN receive(HSG_ID_1,TRANS_ID_9) FROH PE_2
TRANS ts9_222
  FROH noname_221
END /*block B_47*/
END /*block B_14*/
HACROTRANS t5_38
  FROH etat4_33
  TO etat3_31
BLOCK B_17 FOR t5_38
HACROTRANS ts0_56
BLOCK B_48 FOR ts0_56
TRANS noname_226
  TO noname_228
  WHEN GetValue(y) ON b
HACROTRANS ts1_227
  FROH noname_228
BLOCK B_49 FOR ts1_227
CONST
  HSG_ID_16 = 16;
  TRANS_ID_12 = 12;
  TERH_ID_14 = 14;
  HSG_ID_7 = 7;
TRANS noname_229
  TO noname_242,noname_255
  DO R1.append(y)
HACROTRANS ts7_230
  FROH noname_246,noname_258
BLOCK B_54 FOR ts7_230
CONST
  HSG_ID_1 = 1;
TRANS noname_269
  OUTPUT send(HSG_ID_1, TRANS_ID_12) TO PE_2
HACROTRANS ts8_276
BLOCK B_56 FOR ts8_276
CONST
  HSG_ID_10 = 10;
TRANS noname_278
  OUTPUT send(HSG_ID_10, TRANS_ID_12) TO PE_0
END /*block B_56*/
END /*block B_54*/
TRANS noname_231
  FROH noname_242
  TO noname_245
  OUTPUT send(HSG_ID_16, TRANS_ID_12, TERH_ID_14, y) TO PE_0
TRANS noname_243
  FROH noname_245
  TO noname_246
  WHEN receive(HSG_ID_7,TRANS_ID_12) FROH PE_0
TRANS noname_247
  FROH noname_255
  TO noname_257
  OUTPUT send(HSG_ID_16, TRANS_ID_12, TERH_ID_14, y) TO PE_2
TRANS noname_256
  FROH noname_257
  TO noname_258
  WHEN receive(HSG_ID_7,TRANS_ID_12) FROH PE_2
END /*block B_49*/
END /*block B_48*/
END /*block B_17*/
HACROTRANS t6_39
  FROH etat5_32
  TO etat4_33
BLOCK B_20 FOR t6_39
HACROTRANS ts0_57
BLOCK B_57 FOR ts0_57
HACROTRANS ts7_280
BLOCK B_58 FOR ts7_280
HACROTRANS ts9_281
BLOCK B_59 FOR ts9_281
CONST
  HSG_ID_9 = 9;
  TRANS_ID_12 = 12;
  HSG_ID_8 = 8;
TRANS noname_283
  OUTPUT send(HSG_ID_9, TRANS_ID_12)
TRANS noname_286
  WHEN receive(HSG_ID_8,TRANS_ID_12)
  END /*block B_59*/
  END /*block B_58*/
  END /*block B_57*/
  END /*block B_20*/
INIT etat0_28,etat5_32;
END.
/*-------------------------*/
/* Entite de Protocole PE_2 */
/*-------------------------*/
PROTOCOL PE_2
  ACQUAINTANCE PE_0,PE_1;
  SAP c;
  VAR
    x:INTEGER;
    y:INTEGER;
  RESOURCE R1
    READ FUNCTION size():INTEGER;
    WRITE PROCEDURE append(x:INTEGER);
  END;
  RESOURCE R2
    READ FUNCTION size():INTEGER;
    WRITE PROCEDURE append(x:INTEGER);
  END;
  HACROTRANS t1_46
    FROH etat0_40
    TO etat1_41
  BLOCK B_6 FOR t1_46
  HACROTRANS noname_88
  BLOCK B_25 FOR noname_88
  CONST
    HSG_ID_2 = 2;
    TRANS_ID_0 = 0;
    PE_ID_0 = 0;
    TERH_ID_7 = 7;
    HSG_ID_3 = 3;
  TRANS noname_73
    TO noname_80
    WHEN receive(HSG_ID_2,TRANS_ID_0,PE_ID_0,TERH_ID_7) FROH PE_0
  TRANS noname_74
    FROH noname_80
    OUTPUT send(HSG_ID_3, TRANS_ID_0, TERH_ID_7, R2.size()) TO PE_0
  END /*block B_25*/
  END /*block B_6*/
  HACROTRANS t2_47
    FROH etat1_41
    TO etat2_42
  BLOCK B_9 FOR t2_47
  HACROTRANS noname_133
  BLOCK B_28 FOR noname_133
  CONST
    HSG_ID_3 = 3;
    TRANS_ID_4 = 4;
    TERH_ID_3 = 3;
    HSG_ID_4 = 4;
    HSG_ID_17 = 17;
    HSG_ID_16 = 16;
    HSG_ID_7 = 7;
  TRANS noname_100
    TO noname_110
    WHEN receive(HSG_ID_3,TRANS_ID_4,TERH_ID_3,x) FROH PE_0
  TRANS noname_107
    TO noname_111
    WHEN receive(HSG_ID_4,TRANS_ID_4) FROH PE_0
  TRANS noname_109
    FROH noname_110,noname_111
    TO noname_123
    DO R2.append(x)
  TRANS noname_112
    FROH noname_125
    OUTPUT send(HSG_ID_17, TRANS_ID_4) TO PE_0
  TRANS noname_114
    FROH noname_123
    TO noname_124
    OUTPUT send(HSG_ID_16, TRANS_ID_4, TERH_ID_3, x) TO PE_1
  TRANS noname_126
    FROH noname_124
    TO noname_125
    WHEN receive(HSG_ID_7,TRANS_ID_4) FROH PE_1
  END /*block B_28*/
  HACROTRANS noname_146
  BLOCK B_32 FOR noname_146
  CONST
    HSG_ID_1 = 1;
    TRANS_ID_4 = 4;
  TRANS noname_140
    TO noname_143
    WHEN receive(HSG_ID_1,TRANS_ID_4) FROH PE_0
```

```
TRANS ts9_144
  FROM noname_143
END /*block B_32*/
END /*block B_9*/
MACROTRANS t3_48
  FROM etat1_41
  TO etat2_42
BLOCK B_12 FOR t3_48
MACROTRANS noname_186
BLOCK B_37 FOR noname_186
CONST
  MSG_ID_16 = 16;
  TRANS_ID_8 = 8;
  TERM_ID_3 = 3;
TRANS noname_174
  TO noname_179
  WHEN receive(MSG_ID_16,TRANS_ID_8,TERM_ID_3,x) FROM PE_0
MACROTRANS ts3_180
  FROM noname_179
BLOCK B_39 FOR ts3_180
CONST
  MSG_ID_7 = 7;
TRANS noname_191
  TO noname_192
  DO R1.append(x)
TRANS noname_193
  FROM noname_192
  OUTPUT send(MSG_ID_7, TRANS_ID_8) TO PE_0
END /*block B_39*/
END /*block B_37*/
MACROTRANS noname_203
BLOCK B_41 FOR noname_203
CONST
  MSG_ID_1 = 1;
  TRANS_ID_8 = 8;
TRANS noname_197
  TO noname_200
  WHEN receive(MSG_ID_1,TRANS_ID_8) FROM PE_0
TRANS ts9_201
  FROM noname_200
END /*block B_41*/
END /*block B_12*/
MACROTRANS t4_49
  FROM etat2_42,etat3_43
  TO etat0_40,etat5_44
BLOCK B_15 FOR t4_49
MACROTRANS ts0_55
BLOCK B_43 FOR ts0_55
MACROTRANS ts2_205
BLOCK B_44 FOR ts2_205
TRANS noname_206
  TO noname_208
  OUTPUT SizeIndication(R1.size(), R2.size()) ON c
MACROTRANS ts7_207
  FROM noname_208
BLOCK B_45 FOR ts7_207
CONST
  MSG_ID_1 = 1;
  TRANS_ID_9 = 9;
TRANS noname_209
  OUTPUT send(MSG_ID_1, TRANS_ID_9) TO PE_0
TRANS noname_217
  OUTPUT send(MSG_ID_1, TRANS_ID_9) TO PE_1
END /*block B_45*/
END /*block B_44*/
END /*block B_43*/
END /*block B_15*/
MACROTRANS t5_50
  FROM etat4_45
  TO etat3_43
BLOCK B_18 FOR t5_50
MACROTRANS noname_260
BLOCK B_51 FOR noname_260
CONST
  MSG_ID_16 = 16;
  TRANS_ID_12 = 12;
  TERM_ID_14 = 14;
TRANS noname_248
  TO noname_253
  WHEN receive(MSG_ID_16,TRANS_ID_12,TERM_ID_14,y) FROM PE_1
MACROTRANS ts3_254
  FROM noname_253
BLOCK B_53 FOR ts3_254
CONST
  MSG_ID_7 = 7;
TRANS noname_265
  TO noname_266
  DO R1.append(y)
TRANS noname_267
  FROM noname_266
  OUTPUT send(MSG_ID_7, TRANS_ID_12) TO PE_1
END /*block B_53*/
END /*block B_51*/
MACROTRANS noname_277
BLOCK B_55 FOR noname_277
CONST
  MSG_ID_1 = 1;
  TRANS_ID_12 = 12;
TRANS noname_271
  TO noname_274
  WHEN receive(MSG_ID_1,TRANS_ID_12) FROM PE_1
TRANS ts9_275
  FROM noname_274
END /*block B_55*/
END /*block B_18*/
TRANS t6_51
  FROM etat5_44
  TO etat4_45
INIT etat0_40,etat5_44;
END.
```

Résumé

Nous proposons dans notre thèse une nouvelle méthodologie pour le développement des protocoles de communication et des applications réparties en explorant une approche de synthèse automatisée.
Après avoir effectué un état de l'art sur les techniques de synthèse de protocoles, nous présentons, dans la première partie de la thèse, une nouvelle méthode de synthèse automatique de spécifications de protocoles à partir de spécifications de services dans un modèle de réseaux de Petri interprétés. Notre approche de synthèse est basée sur un nouveau concept de raffinement de réseaux de Petri qui assure la correction des protocoles construits de façon incrémentale. L'intérêt principal de cette méthode par rapport aux techniques classiques basées sur l'analyse est qu'elle évite la phase supplémentaire de validation du protocole qui passe souvent par une analyse exhaustive dont le problème fondamental est l'explosion combinatoire. De plus, le coût de construction des protocoles est considérablement diminué. Partant d'une description de service, notre technique de synthèse permet de dériver automatiquement au niveau protocole l'ensemble des fonctionnalités suivantes : (1) traitements répartis, (2) flot de contrôle, (3) flot de données, (4) contrôle de la cohérence des données, (5) et choix distribué. Elle constitue, de ce fait, un "bon" compromis entre le pouvoir de synthèse et le pouvoir d'expression des services relativement aux techniques existantes.
Dans la deuxième partie de la thèse nous présentons une application réelle de notre méthode de synthèse à la conception du protocole de transport ISO de base (ISO 8073), le point de départ de la conception étant la spécification du service de transport ISO 8072.
Dans la troisième partie de la thèse, nous présentons un ensemble d'outils logiciels pour la synthèse automatique de protocoles que nous appelons STEPS (Software Toolset for automatEd Protocol Synthesis). Nous avons développé cet ensemble d'outils afin de démontrer l'utilité et la faisabilité de l'approche de synthèse que nous proposons.
La sémantique de notre modèle de spécification étant assez proche de celle du langage Estelle, nous proposons une extension de notre démarche pour la synthèse de spécifications de protocoles dans le langage Estelle. L'intérêt est que ces spécifications peuvent être directement exploitées par des outils existants afin de générer automatiquement du code exécutable ou d'effectuer des simulations et des évaluations de performances.

Mots clés : *Protocole de communication, service de communication, application répartie, ingénierie des protocoles, synthèse de protocoles, réseaux de Petri, vérification des protocoles.*