# CONTRIBUTION A LA SPÉCIFICATION DES SYSTÈMES TEMPS RÉELS : L'APPROCHE UML/PNO

Jérôme Delatour

Année : 2003
N° d'ordre :

**THÈSE**
Présentée au Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)
Du CNRS en vue de l'obtention du titre de

**Docteur de l'université Paul Sabatier de Toulouse**

Spécialité : Automatique et Informatique Industrielle

Par :
**Jérôme DELATOUR**
Ingénieur EERIE (École pour les Études et la Recherche en Informatique)

# CONTRIBUTION A LA SPECIFICATION DES SYSTÈMES TEMPS RÉELS :

# L'APPROCHE UML/PNO

Soutenue le 30 septembre 2003 devant le jury composé de :

| | |
|---|---|
| Président | Gérard AUTHIE |
| Rapporteurs | Jean-Pierre ELLOY |
| | Jean-Louis SOURROUILLE |
| Examinateurs | Robert VALETTE |
| Directeur de thèse | Mario PALUDETTO |

A Mamido et Grand-père qui sont partis,
A Selma et Luc qui sont arrivés.

# Avant Propos

Le travail présenté dans ce mémoire a été effectué au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS) et s'est poursuivi à l'Ecole Supérieure d'Electronique de l'Ouest (ESEO). Aussi, je tiens à remercier Messieurs J.C. Laprie et M. Ghallab, directeurs successifs du LAAS, pour m'avoir accueilli dans leur laboratoire, ainsi que Monsieur V. Hamon, directeur de l'ESEO, pour m'avoir engagé dans son école.

Je remercie l'administration et les services techniques (réseau informatique, documentation, édition, magasin) de ces deux établissements pour leur aide et leur attention.

J'ai effectué ces années de thèse au LAAS au sein du groupe de recherche : Organisation et Conduite des Systèmes Discrets (OCSD). Je remercie les responsables successifs, J.C. Hennet et F. Roubellat de m'y avoir accueilli. Je tiens à exprimer ma reconnaissance à l'ensemble des membres du groupe OCSD pour leur gentillesse.

Je remercie Monsieur G. Authié, professeur à l'université Paul Sabatier, d'avoir accepté de présider le jury de soutenance. Je le remercie aussi pour son aide lors de mes réinscriptions au cours de ce parcours atypique de thèse. Je suis extrêmement reconnaissant envers Messieurs J.P. Elloy et J.L. Sourouille pour avoir accepté la charge d'être rapporteur des travaux présentés. Je remercie Monsieur R. Valette de m'avoir fait l'honneur de participer au jury en tant qu'examinateur. Son soutien a été constant durant toutes ces années de thèse et je n'oublierai pas ses encouragements et avis. Une partie de mes travaux n'aurait pas vu le jour sans son aide.

Ce travail a été dirigé par Mario Paludetto à qui j'exprime ma gratitude pour ses conseils et son encadrement. Je retiendrais aussi sa confiance, sa sympathie, sa patience et l'autonomie qu'il m'a donnée.

Durant ces années de thèse, j'ai eu la grande chance de rencontrer des personnes extrêmement enrichissantes. Les citer toutes, sans oubli, est une bien difficile gageure. Pardon d'avance à ceux que j'aurais omis :

Tout d'abord, je tiens à remercier les enseignants avec qui j'ai travaillé, ceux de l'INSA de Toulouse et tout particulièrement G. Mottet qui a été le premier à me faire confiance, mais aussi B. Pradin et J. Erschler pour m'avoir permis d'être moniteur. Je n'oublie pas la fine équipe de maîtrise de l'université Paul Sabatier (Gérard, Hamid, Jean-Claude, Jean-Michel, Mario, Michel, Philipe ) ainsi que mes collègues actuels de l'ESEO.

Un grand salut à toute l'équipe des doctorants du temps où le groupe OCSD s'appelait encore SP… Adel, Christian, Eric, Isabelle, François, les deux Gilles, JP, Marco, Pierre et à ceux qui arrivèrent un peu plus tard… Dimitri, Luc, Marc, Toto… Merci pour les rires, les joies et les discussions tardives que nous avons partagé. Un gros clin d'œil à Christian, imprimeur multicarte, pour sa gouaille et sa bonne humeur

Aux complices rencontrés, de la communauté objet temps réel (tout spécialement à Sébastien pour ces providentiels coups de pouce) aux Nantais (David et Sébastien), en passant par l'équipe SIMONA de l'université technologique de Delft (tout particulièrement René pour son accueil).

A mes anciens étudiants, en particulier à Florent et Fred pour leur aide qui est allée beaucoup plus loin que le développement d'ArgoPNO.

A ma famille et mes amis proches. Je me réserve le privilège de les remercier de vive-voix.

# Table des matières

# Table des figures

# Abréviations utilisées

| | |
|---|---|
| § | Chapitre ou paragraphe |
| AGL | Atelier de Génie Logiciel |
| BPN | Behavioural Petri Net (Réseau de Petri de comportement) |
| CC | Compound Class(es) (classe d'objet composé) |
| CO | Compound Object(s) (objet composé) |
| CT | Contrainte(s) Temporelle(s) |
| CTA | Contrainte(s) Temporelle(s) Attendue(s) |
| CTP | Contrainte(s) Temporelle(s) Posée(s) |
| DEA | Diplôme d'Etude Approfondie |
| DUT | Delft University of Technology |
| eBPN | external Behavioural Petri Net(s) (RdP de comportement externe) |
| EPN | Environmental Petri Net(s) (RdP d'environnement) |
| FPN | Functional Petri Net(s) (RdP fonctionnel) |
| HOOD | Hierarchical Object Oriented Design |
| iBPN | internal Behavioural Petri Net(s) (RdP de comportement interne) |
| MDA | Model Driven Approach |
| ms | milli-seconde(s) |
| OMG | Object Management Group |
| OO | Orienté(e) Objet |
| PNO | Petri Net Object(s) (Objet(s) à réseau de Petri) |
| RdP | Réseau(x) de Petri |
| RFP | Request For Proposal (Appel à proposition) |
| SdF | Sûreté de fonctionnement |
| SDL | Specification Description Language |
| SIMONA | International Institute for SImulation, MOtion and NAvigation |
| SRS | SIMONA Research Simulator |
| STFA | Sémantique de Tir FOrte |
| STFO | Sémantique de Tir FAible |
| STR | Système(s) Temps Réel |
| UML | Unified Modelling Language |
| XML | eXtensible Markup Language |

# Introduction générale

## Contexte

Le développement de systèmes temps réel est de plus en plus important dans la société actuelle. L'essor croissant des technologies permet de définir des systèmes informatiques sophistiqués d'une complexité de plus en plus difficile à maîtriser. Ainsi, la conduite automatisée de métro, la supervision d'une centrale nucléaire, d'une chaîne de fabrication… sont autant d'exemples de cette informatisation croissante des systèmes complexes.

Ce sont aux problèmes intervenant dans les premières phases du développement logiciel de ce type de systèmes que nous nous intéressons. Il en est un qui pose un défi par l'antinomie de la modélisation qui se doit d'être à la fois, mathématiquement correcte et compréhensible par toutes les personnes impliquées dans le développement.

En effet, pour des critères essentiellement économiques, le système doit être modélisé à l'aide de langages dit formels très tôt dans le cycle de vie. Cela permet d'assurer sa cohérence et sa faisabilité. Le surcoût indéniable entraîné par l'utilisation de ce type de langages reste largement inférieur à l'impact d'un dysfonctionnement lors de l'exécution du système ou d'une correction tardive au cours du développement.

Toutefois, les langages de ce type se révèlent complexes à utiliser, leur compréhension restant réservée à des spécialistes de la modélisation formelle. Or, le développement de systèmes temps réel fait appel à différents métiers, que ce soit en informatique (sûreté de fonctionnement, protocole, temps réel, base de donnée, ergonomie), en automatique, en électronique… Les langages formels sont alors peu adaptés aux échanges métiers pour lesquels il est nécessaire de favoriser la communication entre les différentes spécialités. De plus, ils obligent généralement l'analyste-concepteur à entrer rapidement dans des niveaux de détails qui ne facilitent pas une compréhension globale du système en construction.

Pour ces raisons, l'utilisation conjointe de notations semi-formelles et de langages formels (approches dites mixtes) est de plus en plus demandée. Les notations semi-formelles simplifient la capture des besoins, le dialogue entre les différents spécialistes intervenant dans le développement, le passage des fossés sémantiques (*semantic gap*) que représentent les transitions entre les phases d'analyse des besoins (*requirements*), de spécification du système (*analysis*) et de conception préliminaire (*global design*). En outre, les nécessaires activités de validation sont facilitées afin de s'assurer que le système en cours de développement est bien celui attendu par les clients.

Avec l'apparition de la notation UML (*Unified Modelling Language*) [OMG-1997b], la question du choix d'une représentation semi-formelle ne se pose plus. Son statut de standard « de facto », sa richesse d'expression, ses capacités d'extension, son large spectre de couverture des phases du cycle de vie, le nombre d'outils informatiques l'implémentant et son adoption par la plupart des industries travaillant dans le domaine du temps réel font qu'UML semble tout désigné pour remplir les rôles demandés à une notation semi-formelle.

Dans ce contexte se pose le problème de coupler cette notation à un ou des modèles formels. De plus, UML, défini en premier lieu pour des systèmes informatiques de gestion d'informations,

ne permet pas de prendre en compte tous les aspects spécifiques du temps réel. Il faut donc proposer des extensions permettant une meilleure description de la dynamique du système, telles que la représentation, du parallélisme, des synchronisations des activités du système ou des contraintes temporelles.

# Choix de recherche

Cette thèse propose une approche mixte de spécification de systèmes temps réels basée sur la notation UML et le formalisme des réseaux de Petri (RdP) [Petri-1962]. Nous avons choisi ce langage formel car il permet de répondre simultanément à deux besoins principaux : l'un concerne l'enrichissement du pouvoir d'expression d'UML pour les aspects dynamiques et l'autre, l'intégration d'un modèle formel asynchrone disposant de nombreux outils de vérification et de validation.

Nous avons appelé cette approche UML/PNO (*UML with Petri Net Object*[1]). Le choix de ce nom a été effectué afin de refléter, d'une part, l'utilisation de la notation UML avec les réseaux de Petri, et d'autre part, sa parenté avec les travaux sur HOOD/PNO [Paludetto-1991] menés au sein de notre groupe de recherche.

L'approche HOOD/PNO a déjà défini un ensemble de principes et de concepts qui permettent de répondre à quelques-unes de nos problématiques. En effet, les problèmes liés au parallélisme, à la distribution et à la complexité ont déjà été abordés par cette méthode. Il nous semble donc tout à fait cohérent de profiter d'une partie de ses concepts, particulièrement sains, en les réutilisant dans notre approche. Ainsi, nous retrouvons dans UML/PNO une partie de la philosophie HOOD/PNO, à savoir : une approche globalement descendante orientée contrôle, l'utilisation des RdP pour la description du comportement des objets, la prépondérance de la structuration de la modélisation par une hiérarchie de composition…

Néanmoins, HOOD/PNO ne traitait pas explicitement les aspects temporels. Dans UML/PNO, cet aspect temporel est pris en compte tout au long du développement. C'est principalement là que réside une des originalités de notre approche. Notre démarche de développement est architecturée autour du souci de construire le système en respectant constamment les contraintes temporelles. En ce sens, nous nous écartons des approches traditionnelles pour lesquelles la vérification des contraintes temporelles est effectuée à des étapes particulières du développement, généralement en phase de conception (typiquement à l'étape de l'évaluation des performances). Dans UML/PNO, cette vérification est possible à chaque incrément de la modélisation et ce, dès la phase d'analyse.

# Organisation du document

Ce mémoire comprend cinq parties :

- La première partie détaille nos choix et nos motivations dans le travail d'élaboration d'une approche mixte UML pour les systèmes temps réel. Un rapide état de l'art des approches mixtes UML pour le temps réel est donné.
- La seconde partie présente nos propositions concernant la structuration du système. Elle définit des mécanismes orientés objets adaptés au temps réel, permettant une meilleure gestion de la complexité en offrant des vues plus ou moins détaillées du système.
- La partie III décrit la formalisation des aspects dynamiques du système à l'aide des

[1] UML avec des objets à réseaux de Petri.

réseaux de Petri. Ces derniers sont considérés comme un nouveau type de diagramme UML, dit diagramme à RdP. Ils sont basés sur une classe de réseaux de Petri à places et transitions temporelles, afin d'offrir une grande richesse de description des contraintes temporelles.

- La partie IV montre comment, à partir des éléments UML, nous dérivons semi-automatiquement les diagrammes à RdP. Des aides à la validation et à la vérification de la cohérence des diagrammes UML sont proposées.
- En partie V, une technique de vérification des contraintes temporelles est proposée. Elle s'appuie sur des techniques existantes de vérification (graphe de classe et logique linéaire). Nous montrons comment nous adaptons ces techniques à la classe particulière des RdP utilisée dans UML/PNO. L'outil ArgoPNO, implémentant cette démarche de vérification, est présenté.

# Une application support d'UML/PNO : SIMONA

Tout au long de ce mémoire, les concepts utilisés sont illustrés à l'aide d'exemples tirés d'une application réelle étudiée durant cette thèse. Il nous a semblé intéressant de présenter cette application, car elle a eu une influence sur notre travail et nous a permis d'expérimenter les principes de notre approche.

Ce cas d'étude est un simulateur de vol temps réel développé à l'université Technique de Delft (DUT[2]) aux Pays-Bas. Il s'agit d'un projet de recherche destiné à offrir une plate-forme de test, ouverte à différentes disciplines scientifiques (électronique, automatique, mécanique, aéronautique, informatique…), afin que celles-ci puissent développer leurs techniques sur une application réelle. Ce projet, désigné sous le nom de SIMONA (*International Institute for SImulation, MOtion and NAvigation*), a été monté par plusieurs facultés de la DUT (Facultés d'Électronique, de Mécanique, d'Aéronautique et Astronautique, de Mathématiques et d'Informatique Technique) en partenariat avec des industriels des secteurs électroniques, électriques et mécaniques (Hewlett-Packard, Siemens, Hydraudine, ...), des compagnies aéronautiques (Boeing, Cessna Aircraft Corporation, Fokker, KLM Royal Dutch Airlines) et des groupes de recherche ou institutionnels (American Institute for Aeronautics and Astronautics, European Space Agency, NASA Arms Research Center, MIT…).

Au cours d'un séjour de trois mois dans le département de recherche *Control and Simulation* de la faculté d'Aéronautique et Astronautique de Delft, une première analyse globale du simulateur a été menée. Cette analyse a été effectuée en utilisant UML/PNO.

L'ambition des concepteurs de SIMONA est grande. Il s'agit de développer le premier simulateur de vol orienté objet, bénéficiant des dernières avancées technologiques, aussi bien sur le plan matériel que sur le plan logiciel. La confrontation à des problèmes réels et à des besoins industriels non triviaux ou encore peu abordés a été immédiate.

Cette expérience m'a convaincu de la pertinence d'une méthode de spécification pour les systèmes temps réel, traitant en premier lieu la complexité et la dynamique des systèmes et favorisant le dialogue entre les différentes personnes parties prenantes du développement. Elle m'a permis de constater un besoin pour des aides aux activités de validation et de vérification qui devaient rester simples d'utilisation et la nécessité d'expliquer, plus que leur fondement mathématique, le cadre de leur utilisation et les moments où ces aides devaient être employées. UML/PNO tente de répondre à ces besoins.

[2] Delft University of Technology.

# Partie I Vers une approche mixte UML pour le temps réel



Cette partie a pour but de présenter nos choix et nos motivations dans ce travail d'élaboration d'une approche mixte UML pour les systèmes temps réel.

En § I.1, nous précisons à quelle classe de problèmes s'applique notre approche. Cette identification nécessite une présentation du domaine des applications temps réel, ainsi que de la terminologie employée.

Puis, en § I.2 nous détaillons les problèmes que nous avons abordés et motivons les choix effectués dans ce travail de recherche.

En § I.3, nous présentons une étude des approches mixtes UML, basées sur des modèles formels asynchrones et dédiés à l'analyse et à la conception de systèmes temps réel.

En conclusion (§ I.4), nous résumons nos choix et introduisons la partie suivante de ce mémoire.

# I.1. Terminologie

De nombreuses disciplines scientifiques se sont intéressées aux problématiques soulevées par les Systèmes Temps Réel (STR). Chacune étudiant un domaine différent, utilisant un vocabulaire spécifique et définissant des concepts qui lui sont propres, la terminologie associée au temps réel ne fait pas actuellement l'objet d'un consensus. Dans le domaine informatique, le terme même de "temps réel" suscite encore différentes interprétations [Stankovic-1988].

Afin d'éviter toute ambiguïté d'interprétation, le présent chapitre présente la terminologie et les définitions utilisées dans ce mémoire. Notre intention n'est pas de fixer «la bonne terminologie», mais tout simplement de préciser, dans la mesure du possible, son emploi.

## I.1.1. Définition d'un système temps réel

De nombreuses définitions ont été données pour les STR (voir par exemple [Elloy-1989] [Laplante-1991], DIN44300, [CNRS-1988], [Dorseuil-1991]...). Nous reprenons la définition de J. A. Stankovic, car elle nous paraît très générale et adaptée aux domaines des systèmes industriels sur lesquels nous travaillons :

Un système temps réel est un système dont l'exactitude (*correctness*) dépend non seulement des résultats logiques, mais également des instants où ces résultats sont fournis [Stankovic-1988].

Cette définition permet de se détacher du sens courant de « temps réel » souvent assimilé à tort au seul fait que ce type de système doit être très rapide. Un système ayant des délais à respecter exprimés en heures peut être considéré comme un STR (système de conduite de hauts fourneaux par exemple). Ce n'est donc pas uniquement la « rapidité des temps de réponse » (au sens valeurs très faibles des durées) qui caractérise un STR. Mais le fait qu'une production précoce ou tardive d'un résultat exact est alors considérée comme incorrecte. Néanmoins, la plupart des STR ont des temps de réponse exprimés en milli-secondes (ms). Le respect de telles contraintes de promptitude rend complexe le développement des systèmes et constitue une des caractéristiques importantes des STR.

Cette définition s'explique essentiellement par le fait que les STR sont des systèmes appartenant à la classe des systèmes dits réactifs. Les systèmes réactifs (ou interactifs) [Harel-1985] sont utilisés afin de surveiller, commander, piloter un système physique existant (désigné par le terme de procédé). Il s'agit de s'assurer que le comportement naturel de ce procédé (les lois physiques qui le régissent indépendamment du système le pilotant ou le surveillant) reste dans des plages tolérables de fonctionnement.

Par exemple, et de manière simplifiée, le rôle du système réactif de pilotage automatique d'un avion sera justement d'empêcher sa chute, c-à-d de le maintenir à une altitude tolérable (ce que nous appelons le comportement maîtrisé).

Ce sont donc des systèmes fortement dépendants de leur environnement. Le système réactif doit être suffisamment rapide pour pouvoir observer et commander le procédé dont il doit assurer le bon fonctionnement. C'est l'environnement et le ou les procédés le composant qui rythment le fonctionnement du système. Il s'agit là, d'une différence fondamentale avec les systèmes informatiques classiques (dit transformationnels) dont le principal rôle consiste à réaliser une fonction de transfert entre des données d'entrée et des données de sortie. C'est, dans ce cas, la performance du système qui rythme les flux d'entrée et de sorties des données.

Par rapport à cette définition des systèmes réactifs, les STR sont donc des systèmes réactifs pour lesquels les contraintes de temps sont explicites et où le non-respect d'une contrainte temporelle peut conduire l'application à un état catastrophique [Calvez-1990]. Précisons cette

notion de temps et de contraintes temporelles.

## I.1.2. Temps et contraintes temporelles

Le concept de temps est une notion difficile à définir, ainsi que le rappelle saint Augustin : « Qu'est ce donc que le temps ? Si personne ne me le demande, je le sais ; mais si on me le demande et que je veuille l'expliquer, je ne le sais plus. » (*Confession, 11, XV*).

Pour notre part, nous ne donnons pas une définition philosophique ou mathématique du temps. De manière pragmatique, à l'instar de L. Motus [Motus-1992], nous considérons simplement le temps comme un moyen d'exprimer des synchronisations entre des activités concurrentes[3] au sein d'un STR, mais aussi entre ces activités et celles de l'environnement. Nous utilisons, ici, le terme "activité" au sens de fait temporel. Nous définissons l'occurrence d'un événement comme la manifestation (généralement le début ou la fin) d'une activité.

Les indications sur l'ordonnancement des événements temporels sont précisément les contraintes temporelles (CT).

### I.1.2.1. Les types de temps

La communauté informatique distingue deux grands types de temps [CNRS-1988] [Kopetz-1983] :

- Le temps continu. est un temps que l'on suppose généralement dense (et complet), au sens mathématique du terme. Il est supposé qu'entre deux instants t et t', il y a toujours un troisième instant t". L'observation de la simultanéité d'événements n'est donc pas possible. Lorsque ce temps est équipé d'une métrique, le terme de temps physique est employé.
- Le temps logique correspond à une succession d'instants discrets, prédéfinis et ordonnés. Ces instants peuvent être caractérisés par des événements ou par leur périodicité. Le temps logique entre deux instants n'est pas défini, il est non dense. La simultanéité des événements est donc possible. Lorsqu'une métrique y est associée (ensemble des entiers relatifs par exemple), c'est un temps échantillonné.

Souvent, le temps physique est utilisé pour représenter les caractéristiques temporelles de l'environnement et du monde physique (d'où son nom), tandis que le temps échantillonné représente celui du temps discret manipulé par le système informatique.

Ces deux types de temps peuvent être précisés selon deux aspects :

- Temps local/global. Cette définition n'a de sens que dans un contexte où plusieurs horloges, généralement distribuées[4], existent dans le système. Elle décrit le fait que "l'écoulement du temps" n'est pas mesuré de la même façon dans chacun des nœuds du système. Il s'agit d'un temps global si l'écoulement du temps est le même pour tous les nœuds du système sinon il s'agit d'un temps local à un nœud.
- Temps relatif/absolu. Il dépend de la référence initiale de ce temps selon que cette référence est partagée par tous les temps (temps absolu) ou est spécifique à chacun des temps.

[3] A strictement parler, il faudrait, selon les spécialistes en calcul parallèle, distinguer la notion d'activités parallèles (*parallel*) et concurrentes (*concurrency*). Dans le premier cas, les activités se déroulent de manière simultanée sans communiquer, tandis que dans le second cas, il y a toujours parallélisme, mais avec synchronisations par communications entre certaines activités. Pour notre part, nous ne faisons pas de différence entre ces deux termes, que nous emploierons toujours avec le sens de concurrence.

[4] Un système distribué (ou réparti) est défini comme un ensemble de nœuds de traitements ou processeurs (avec mémoires incorporées) interconnectés par un réseau de communication [Le Lann-1994].

### I.1.2.2. Les types de contraintes temporelles

#### I.1.2.2.a) Par rapport aux types de temps utilisés

Les CT pouvant être exprimées avec les différents types de temps, nous retrouvons les mêmes catégories que celle du temps. Par exemple, l'ordonnancement des événements peut être formulé :

- En reliant les évènements temporels à des dates (utilisation d'un temps physique ou échantillonné),
- En les reliant à des événements de référence (utilisation d'un temps logique absolu),
- Ou en les reliant entre eux par des relations d'ordre (utilisation d'un temps logique relatif)

Souvent, on parle de contraintes temporelles implicites (ou qualitatives) lorsqu'elles sont exprimées avec un temps sans métrique (temps continu ou logique). On parle de contraintes temporelles explicites (ou quantitatives) lorsqu'elles sont exprimées à l'aide d'un temps avec métrique (physique ou échantillonné).

#### I.1.2.2.b) Par rapport à leurs origines

Lors du développement de STR, il est courant de différencier les contraintes temporelles suivant leurs origines. On parle alors de CT externes et des CT internes :

- Les CT externes sont issues des besoins du système. Elles expriment des relations d'ordre entre le système et des entités qui lui sont extérieures. Elles constituent un ensemble d'informations permettant de caractériser le procédé à piloter. Elles vont se traduire par des exigences sur la façon d'observer et/ou de piloter l'environnement. Ce sont des contraintes définissant le comportement naturel et maîtrisé du procédé. Cela peut être des lois de commande, des vitesses de réaction, des temps de réponse (délai entre la réception d'un stimulus et la génération d'une réponse pour le système). Elles peuvent aussi provenir de besoins non plus dictés par l'environnement mais plutôt par les clients et utilisateurs.
- Les CT internes expriment des relations d'ordre entre les entités internes au système. Ce sont des CT inhérentes au fonctionnement du système. Par exemple : la périodicité (ou non) des actions à effectuer, des synchronisations entre activités, des délais d'attente entre chaque communication (protocole de communication avec attente limité), la validité temporelle d'une donnée, mais aussi des informations concernant les processeurs, leurs vitesses, le système d'exploitation utilisé, les délais de communication, le débit du réseau de communication,…

#### I.1.2.2.c) Par rapport à leur criticité

Une dernière différenciation des CT peut être faite suivant la criticité de leur non-respect. La Figure I-1 représente les trois types de CT de cette catégorie.

Nous avons modélisé, pour chacune de ces CT, l'instant désiré de terminaison (dénommé « l'instant idéal de réponse »). Les courbes des trois figures représentent les conséquences du respect ou non de cette CT sur la qualité de service d'un système. Un écart par rapport à l'instant idéal (dénommé « intervalle de tolérance ») est permis. Il définit l'intervalle temporel durant lequel la CT est considérée respectée (encadré « Correct » sur la figure). En dehors de cet intervalle (encadré « Trop tôt » ou « Trop tard »), il y a non-respect de la contrainte temporelle.

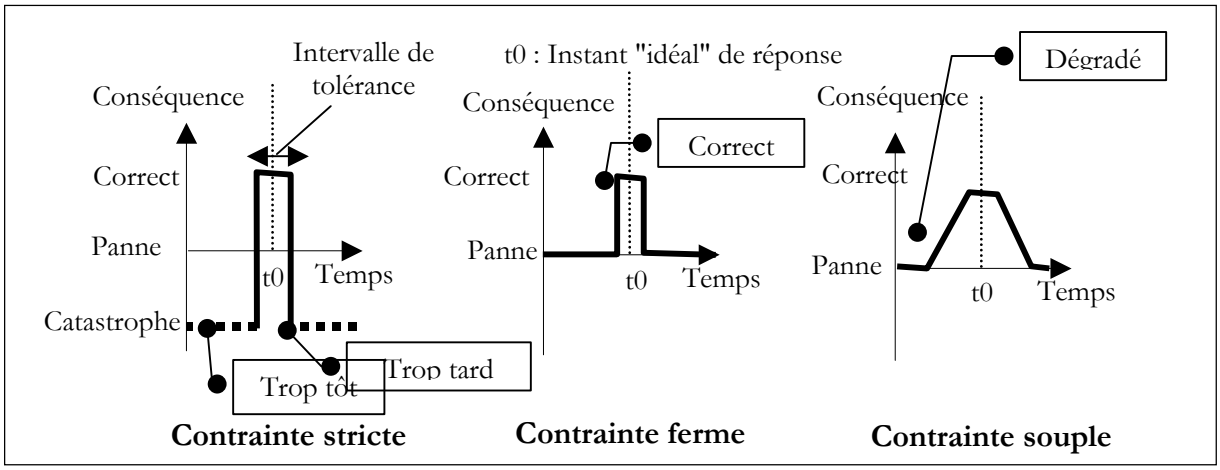


**Figure I-1 : Distinction entre contraintes temporelles strictes, fermes et souples**

Trois cas sont distingués suivant la criticité de cette contrainte temporelle :

- Dans les deux premiers cas (CT stricte et ferme), le non-respect de la contrainte par le système n'est pas envisageable. Le système atteint immédiatement un état de panne. Si les conséquences de ce non-respect mènent à des défaillances catastrophiques, nous parlons de CT critiques. Dans le cas contraire, il s'agit de CT fermes.
- Dans le troisième cas (CT souple), le non-respect de la contrainte temporelle mène à une dégradation progressive des performances du système. Le système entre dans des modes de fonctionnement dégradés jusqu'à un état de panne. Ce type de défaillance peut être tolérable lorsqu'il est peu préjudiciable au système ou à l'environnement contrôlé.

Cette différenciation des CT permet de définir différentes classes de systèmes temps réel.

## I.1.3. Les classes de systèmes temps réel

Une distinction [Shin-1994] est faite entre les STR suivant la criticité des temps de réponse de ces systèmes. On parle alors de :

- STR critiques ou stricts (*hard real-time)*,
- STR fermes ou durs (*firm real-time)*,
- STR souples, lâches ou mous (*soft real-time).*

D'autres critères de différenciation existent entre ces catégories : par exemple, des critères statistiques sur les temps moyens des contraintes temporelles. Cependant, la criticité des contraintes temporelles est une condition nécessaire et suffisante pour distinguer les trois types de STR présentés [Hoogeboom-1991].

Naturellement, le développement de STR stricts ou fermes nécessite impérativement des activités de vérification et de validation du système en construction.

La validation[5] (au sens de validation des besoins) consiste à s'assurer que le système en construction est bien celui attendu par les clients. Le "bon" système est construit. La vérification (au sens de vérification des propriétés mathématiques) consiste à s'assurer que le système est cohérent. Le système est "bien" (correctement) construit.

Afin de remplir ces objectifs, des modèles formels doivent être utilisés [Stankovic-1988]. Par

[5] Les mots "validation" et "vérification" peuvent avoir des sens inverses suivant la communauté qui les emploie. Pour notre part, nous utilisons le sens donné par la communauté informatique.

modèle, nous entendons une description (ou représentation) abstraite du problème, de la réalité et/ou des solutions intermédiaires suivant différentes vues [Wilson-1988]. Ce modèle est dit formel si le langage (ou notation) le décrivant, possède une syntaxe ainsi qu'une sémantique rigoureuse et précise. Pour ce faire, ce type de modèles repose le plus souvent sur une définition mathématique. De ce fait, il est alors possible de lui appliquer des traitements systématiques (parfois désignés par les termes d'outils ou de techniques formelles) afin de l'analyser, le transformer, le vérifier ou le valider.

# I.2. Problèmes abordés et choix de l'axe de recherche

## I.2.1. Principes directeurs

Nous avons focalisé notre recherche sur la gestion de la complexité et la maîtrise des aspects dynamiques en nous intéressant plus particulièrement aux premières phases de développement (phase de spécification et de conception préliminaire) de la modélisation logicielle de STR fermes ou souples. En effet, on constate un besoin croissant de prouver le bien fondé et la cohérence des modèles et ce, très tôt dans le cycle de développement. Cette tendance, d'abord dictée par des contraintes économiques, se voit renforcée par le désir de garantir une bonne évolutivité et une grande pérennité de la solution retenue.

### I.2.1.1. Gestion de la complexité

Concernant la gestion de la complexité[6], nous nous sommes plus particulièrement concentrés sur :

- Les difficultés de l'obtention précoce d'une compréhension globale et synthétique du système en cours de développement,
- La définition ou l'utilisation de représentations communes afin de faciliter le dialogue entre les clients et les différents métiers d'ingénieurs spécialisés dans le développement de parties du système,
- Les problèmes de structuration posés par la gestion d'un grand nombre d'informations de modélisation.

En adoptant les principes de la modélisation orientée objet, nous apportons une première réponse à la structuration du système et par conséquent à cette gestion de la complexité. La technologie des objets et des composants a montré qu'elle est garante de certains facteurs de qualité tels que la modularité, la flexibilité, la réactivité et le suivi des besoins [Booch-1994] [Blair-1998]. Néanmoins, l'application des concepts de la modélisation à objets au domaine du temps réel n'est pas immédiate. En effet, bien que les concepts OO (Orienté Objet) puissent être étendus afin de prendre en compte directement le parallélisme, un certain nombre de difficultés demeurent. Nous détaillerons des adaptations possibles en § I.2.2.

En choisissant la notation UML, une deuxième réponse est apportée concernant la gestion de la complexité du système. UML est utilisé comme un langage commun facilitant les échanges et la communication entre les différentes équipes (commanditaires et clients compris) intervenant sur le développement du système. En § I.2.3, les avantages, mais aussi les limites, de l'utilisation d'UML pour le temps réel sont détaillés.

### I.2.1.2. Maîtrise des aspects dynamiques

Par maîtrise de l'aspect dynamique, nous entendons la gestion du parallélisme et des

[6] Nous reprenons l'utilisation de G. Booch de ce mot [Booch-1994]. Sous le terme complexité, nous regroupons les nombreux problèmes à traiter simultanément concernant les activités de développement (gestion de projet, gestion humaine,…), la difficulté de compréhension de l'environnement et du système (de par sa taille, sa nature...) …

synchronisations entre activités du système. De par la définition que nous avons donné précédemment aux contraintes temporelles, nous considérons que cette problématique (détaillée en § I.2.3) est étroitement liée à celle posée par la modélisation, la vérification et la validation des contraintes temporelles.

Cette nécessité de vérification et de validation conduit à l'utilisation de modèles formels. En effet, dans le cas des STR que nous étudions, il ne suffit pas de prouver que le système se comporte logiquement selon sa spécification. Il doit aussi répondre dans les délais de temps impartis. En § I.2.4 nous présentons les différents modèles formels qui pourraient être utilisés pour remplir ces objectifs et nous motivons notre choix pour l'un d'eux.

Toutefois, l'utilisation seule de méthodes formelles pose certains problèmes. L'un des plus importants reste la lisibilité et la compréhension des modèles, trop souvent réservées à des spécialistes et peu adaptées au franchissement des fameux « *semantic gaps* » [Fraser-1991]. Il devient difficile de communiquer avec les clients, les différents corps de métiers engagés dans le projet (automaticien, électronicien, spécialiste réseaux, spécialiste base de données…) et les utilisateurs. Or, ce souci de communication entre intervenants sur le projet est important dans la construction de systèmes aussi complexes et hétérogènes que les STR.

#### I.2.1.3. Vers une approche mixte

Les objectifs choisis, décrits dans les paragraphes précédents, nous ont donc incités à proposer une approche mixte mêlant l'utilisation de représentations UML (simples et facilement utilisables par un ingénieur) avec des modèles mathématiques formels. En effet, nous pensons qu'il est essentiel d'offrir aux utilisateurs une représentation la plus simple possible (semi-formelle) mais qui resterait issue d'une modélisation formelle.

Néanmoins, ce couplage soulève en lui-même un certain nombre de questions. Par exemple, comment rendre compatibles les modèles formels avec la représentation semi-formelle ? Quelles passerelles, transformations ou relations entre les représentations formelles et semi-formelles peuvent être définies ? Ces différents points sont détaillés au § I.2.5.

### I.2.2. La modélisation objet et le temps réel

L'approche OO se distingue de l'approche fonctionnelle, par le fait qu'un système est considéré comme une collection d'entités « autonomes », composées d'un état et fournissant des services au monde extérieur. Ces entités, appelées plus communément des objets, collaborent à la réalisation des fonctionnalités du système. La communication entre les objets s'effectue selon le concept de client-serveur. Ce procédé a pour effet de protéger l'état des objets contre des accès accidentels de la part d'autres entités. L'objet est vu comme un module autonome, fortement cohérent et entretenant le minimum de relations de dépendances avec d'autres objets. En outre, l'approche OO offre deux mécanismes puissants d'abstraction à travers les hiérarchies de composition et d'héritage.

Cependant, cette utilisation des mécanismes OO dans le domaine du temps réel nécessite quelques adaptations. Comme dans la programmation OO concurrente [Briot-1996], deux grandes approches de modélisation des objets (nous parlerons de représentations) se retrouvent :

- La représentation applicative fournit une bibliothèque d'objets dédiés à la prise en compte des mécanismes de bas niveau pour la gestion du temps réel. Des classes comme « Ordonnanceur », « processus » et « sémaphores » sont ainsi définies. Cette représentation a tendance à laisser l'analyste-concepteur en charge de deux tâches distinctes : d'une part, la modélisation de l'application en terme d'objets et, d'autre part, la gestion du parallélisme et de la répartition (également exprimée en terme d'objets, mais pas les mêmes !).
- La représentation intégrée, à la différence de la précédente, vise à intégrer les mécanismes objets avec les mécanismes du parallélisme et de la répartition, au sein d'une même entité. Trois niveaux d'intégration sont constatés :
  - Le premier niveau intègre la notion d'objet et d'activité (processus ou tâche).

L'objet (nommé « objet actif ») est considéré comme doté d'une ressource de calcul propre. Suivant les approches, l'objet traitera les requêtes de manière séquentielle ou simultanée. Dans le premier cas, la concurrence provient uniquement des activités indépendantes des divers objets (on parle alors de concurrence inter-objets). Si l'on permet à l'objet de traiter plusieurs requêtes simultanément, on parle de concurrence intra-objet.

- Une deuxième intégration associe la synchronisation à l'activation des objets. La transmission de messages est considérée comme une synchronisation implicite entre l'émetteur et le récepteur. On parle alors d'objets synchronisés. Différents protocoles de communication sont définies (synchrones, asynchrones…) et des mécanismes de contrôle de la concurrence des invocations sont précisés (par exemple en associant une garde au niveau de chaque méthode de l'objet).
- Enfin, le troisième niveau d'intégration considère l'objet comme unité de répartition, on parle alors d'objets répartis. Les objets sont vus comme des entités pouvant être distribuées et dupliquées sur différents sites. La métaphore de la transmission de message est étendue à la communication à travers un réseau quelconque.

La représentation intégrée nous semble la plus intéressante. Outre le fait de simplifier la modélisation et d'éviter une structuration « à plat », elle permet d'intégrer les mécanismes du temps réel et de la distribution sur une seule entité : l'objet actif, synchronisé et réparti. Néanmoins, l'utilisation de la représentation intégrée dans le domaine du temps réel reste encore difficile car certains conflits ne sont pas totalement résolus (par exemple, le problème de l'héritage du comportement). Ces difficultés seront détaillées lorsqu'elles seront traitées dans la partie II de ce mémoire (cf. § II.2.1). Les choix concernant la stratégie de modélisation OO adaptée au temps réel ayant été motivés, UML est maintenant présentée.

## I.2.3. UML et le temps réel

UML [OMG-1997b] a été définie par l'OMG (*Object Management Group*) en novembre 1997 comme la notation normalisée des méthodes OO. C'est une notation générique pouvant s'adapter à n'importe quelle méthode OO et pour n'importe quel type d'application.

Elle est devenue rapidement un standard de facto et elle est maintenant très utilisée dans le monde industriel. Différentes versions d'UML ont été depuis publiées. Pour notre part, nous utilisons dans ce mémoire la version 1.3 [OMG-1999b].

Nous supposons que le lecteur possède une première connaissance d'UML. De nombreux ouvrages expliquent la notation UML [Booch-1998] [Muller-1997]. Nous en proposons aussi un résumé [Delatour-2000].

### I.2.3.1. Les bénéfices d'UML

Le principal argument en faveur d'UML est son statut de standard *de facto*. Parmi la multitude de méthodes OO (et les notations qui leur sont associées), disposer d'un consensus dans leur représentation est un avantage certain.

Bien qu'issu des notations OMT et OOD, UML possède un plus grand pouvoir d'expression que ses aînées et couvre une plus grande partie du cycle de développement. Dans la description des aspects statiques du système, la richesse de description est grande et rien ne semble avoir été oublié. La question de la représentation d'un objet ou d'une classe et de ses relations statiques ne se pose plus.

Cet état de fait nous permet d'espérer un meilleur dialogue avec les clients, qui n'ont plus qu'un seul formalisme à connaître (puisque UML est un standard). De plus, les diagrammes UML sont assez intuitifs et aisés à comprendre, car ils sont une reprise de formalismes éprouvés et utilisés depuis une décennie.

Par ailleurs, les possibilités d'extension [Alhir-1999] de cette notation la rendent extrêmement adaptable. Ces mécanismes d'extension sont multiples et très intéressants. En effet, UML repose sur une description en 4 couches afin d'être compatible avec le standard MOF (*Meta-Object Facility*) [OMG-1997a] de l'OMG. Chaque couche est une instance de sa couche supérieure, la plus haute couche étant une instance d'elle-même. Ces quatre couches sont :

- Le méta-méta-modèle : Il définit la structure du méta-modèle et le langage nécessaire à la définition du méta-modèle. Il est défini de manière réflexive.
- Le méta-modèle : C'est une instance du méta-méta-modèle. Il définit le langage de la couche "modèle", en l'occurrence ici les diagrammes et les éléments UML.
- Le modèle : Il représente l'ensemble des modèles permettant de décrire un domaine d'information. Ce sont, par exemple, les classes définies pour le système en construction.
- Les objets utilisateur : C'est une instance du modèle. Ce sont les objets du système exécuté.

Il est ainsi possible de définir des extensions au niveau du modèle ou au niveau du méta-modèle. Cela se fait, par exemple, en spécialisation certains éléments UML (par utilisation des stéréotypes et des valeurs étiquetées). Les extensions du méta-modèle pouvant être regroupées dans un profil UML, elles peuvent être alors directement employées dans des modèles UML adaptés à un domaine d'étude particulier. En § I.2.6.2 d'autres avantages apportés par cette structuration en 4 couches sont détaillés.

Enfin, la disponibilité d'AGL (Atelier de Génie Logiciel) UML libres permettent d'implémenter des prototypes et d'intégrer des techniques et des concepts issues du monde de la recherche.

### I.2.3.2. Les limites d'UML pour le temps réel

Parmi les lacunes d'UML (voir par exemple [Hay-1998] [Simons-1998;Simons-1999] [Sourrouille-1998]), nous n'évoquerons que les plus générales concernant les aspects comportementaux.

#### I.2.3.2.a) Une sémantique UML imprécise

La plus importante critique est liée à l'absence d'une sémantique formelle des modèles UML décrivant le comportement du système [Breu-1997] [Boas-1998] [Lano-1998]. Cette dynamique peut être exprimée, soit à l'aide des diagrammes d'interactions (diagrammes de séquence et de collaboration), soit à l'aide des diagrammes d'états (diagrammes d'états/transitions et les diagrammes d'activités). Les premiers permettent la description d'une histoire (ou à la limite d'un ensemble d'histoires), tandis que les seconds permettent la représentation de tous les comportements possibles d'un objet et, par extension, du système.

En effet, malgré son titre, le document « *UML Semantics* » [OMG-1999a] ne décrit, pour ainsi dire, que la syntaxe d'UML (en précisant son méta-modèle à l'aide de diagrammes de classes et de contraintes OCL[7]). Concernant la sémantique, la description est insuffisamment précise. Elle laisse malheureusement un grand nombre de questions sans réponse. En outre, peu de règles sont définies pour vérifier la cohérence d'un diagramme et, encore moins pour vérifier la cohérence entre les diagrammes.

Ce constat touche particulièrement celui des diagrammes d'états/transitions. Signalons en premier lieu les divergences entre les diagrammes d'états/transitions et le formalisme dont ils

[7] OCL (Object Constraint Language) est un langage permettant de spécifier des contraintes entre des éléments de modélisation UML

sont issus : les Statecharts [Harel-1987]. Les hypothèses des Statecharts (à savoir, communication immédiate, primitives de gestion du temps, actions instantanées, capture immédiate et non-conservation des événements) ne sont plus respectées. Par ailleurs, des extensions comme les états de type *synchStates* ou l'utilisation de fourches d'alternatives font que les techniques existantes de validation pour les Statecharts ne sont donc plus directement applicables. Ainsi, même une simulation des diagrammes d'états/transitions est impossible. De nombreux chercheurs ([Lilius-1999a] [Gogolla-1998] [Gehrke-1998] [Geisler-1998] [Hamie-1998] [Reggio-1999]) ont signalé que de nombreuses règles étaient, soit manquantes (par exemple aucune politique de choix des évènements en attente n'est définie [Börger-2001]), soit simplement incohérentes (priorités de tir des transitions en conflit entre super et sous état [Simons-2000]). Certes, cette extension UML des Statecharts a gagné en convivialité graphique mais au détriment des vérifications formelles qui deviennent impossibles à effectuer sans définir une sémantique cohérente et précise [Schäfer-2001] [Börger-2000].

Afin de corriger cette lacune d'UML, de nombreuses propositions ont donc été effectuées par la communauté scientifique et l'OMG a lancé un RFP[8] « *Action Semantics* » [OMG-1998]. Une première réponse a été apportée [OMG-2000]. Elle introduisait la notion d'occurrence et d'exécution des modèles UML [Pennaneac'h-2001] [Sunyé-2002]. Cette réponse levait alors un certain nombre d'ambiguïtés concernant la sémantique d'UML. Toutefois, ce travail n'a pas été repris dans les spécifications UML (1.4 et 1.5) dans lesquelles seule la partie présentant un langage permettant de spécifier des actions (création, destruction, expression conditionnelle, lecture/écriture…) est présente [OMG-2002a] [OMG-2002c]. Pour ces raisons, nous n'avons pas étudié plus en détail cette réponse au RFP et ne l'avons pas intégrée à notre recherche.

I.2.3.2.b) Une syntaxe insuffisante pour le temps réel

L'autre aspect déficient d'UML, par rapport à son utilisation pour les STR, est lié à sa mauvaise expression de l'aspect dynamique.

*I.2.3.2.b.i) Pas de vue globale et synthétique de la dynamique du système*

Malgré un grand nombre de représentations différentes (quatre) dédiées à la modélisation du comportement, UML ne propose pas de diagramme permettant de synthétiser la plupart de ces informations. Il est par conséquent difficile de disposer d'une vue globale d'un objet ou d'une partie du système. Ainsi, par exemple, le type de communication entre objets (synchrone ou asynchrone) ne peut être graphiquement représenté que sur les diagrammes d'interaction et non sur les diagrammes d'états/transitions[9]. Cette absence de diagramme synthétique est gênante quand il s'agit de communiquer avec plusieurs équipes travaillant sur un même projet. Il faut faire appel à plusieurs types de diagrammes pour parvenir à rassembler toute l'information concernant l'aspect dynamique. Par ailleurs, cela multiplie les risques d'incohérences entre les différentes vues.

*I.2.3.2.b.ii) A propos des diagrammes d'états/transitions UML*

Puisque les diagrammes d'états/transitions sont issus du formalisme des Statecharts, nous retrouvons les mêmes bénéfices et controverses suscités par leurs aînés concernant la description de STR.

En effet, les Statecharts sont bien adaptés à des couches où la plupart des opérations sont séquentielles. En particulier, ils permettent de résoudre graphiquement le problème de l'explosion combinatoire par la généralisation et l'agrégation d'états. Ils permettent également de représenter

[8] RFP (Request For Proposal) : Dans son processus de définition de normes, l'OMG soumet des appels à propositions (nommés RFP). C'est sur la base des réponses à ces propositions et après un cycle plus ou moins long de révisions, qu'une spécification est proposée puis normalisée.

[9] Et même textuellement, la différence ne se fait que sur le Statechart appelant où l'appel est fait par un « callEvent » (dans le cas d'une communication synchrone) ou par un « SignalEvent » (dans le cas d'une communication asynchrone).

les modes de défaillance et de reprise notamment grâce aux entrées initialisation et historique.

En revanche, dès que les niveaux d'ordonnancement de processus, de tâches ou de structuration abstraite sont nécessaires, le parallélisme et la distribution deviennent conséquents. La lisibilité des Statecharts par des non-spécialistes est alors remise en question. Ils n'offrent, en particulier, aucun mécanisme graphique simple de représentation explicite des contraintes et des dépendances temporelles (délais, chiens de garde...). Dans les Statecharts, comme dans les diagrammes d'états/transitions UML, le temps est introduit à l'aide de fonctions renvoyant un booléen lorsqu'un certain laps de temps s'est écoulé. Les relations temporelles entre les différents événements sont alors implicitement traduites dans les diagrammes d'états/transitions. Il devient malaisé de suivre les chemins de commutations depuis un état initial jusqu'à un état final ou inversement [Giese-1999]. Pourtant, c'est un besoin important dans la modélisation des STR. En effet, dans le premier cas, cela permettrait de suivre l'ordre des événements. Dans le second cas, il serait alors possible de connaître les chemins et les états initiaux susceptibles de conduire le système dans un état donné (non désiré par exemple).

Par ailleurs, une autre critique concerne l'abstraction de comportement introduite par les Statecharts [Paludetto-1999]. La structuration offerte est typiquement celle d'une approche fonctionnelle. De ce fait, elle peut induire une répartition centralisée du contrôle : un objet centralisateur, représentatif d'un niveau système, distribue le contrôle à des objets de niveau inférieur. Or, une telle vision des liens comportementaux va à l'encontre de l'indépendance spatiale et temporelle[10] préconisée par l'approche OO. En effet, dans une vision OO, un système se trouve dans un macro-état donné, défini par le vecteur d'états de ses sous-systèmes. En terme de dynamique, ce sont ces derniers qui, par leur coopération et leurs sous-états, définissent l'état de l'objet système.

*I.2.3.2.b.iii) Une représentation limitée du temps*

L'expression des CT explicites reste limitée en UML 1.3.

Sur les diagrammes de séquence, le temps s'écoule de haut en bas. L'axe vertical peut être gradué afin d'exprimer des contraintes temporelles. Deux primitives (*Event.ReceiveTime* et *Event.SendTime*) permettent de différencier les instants où les messages entre objets sont envoyés et reçus. Pour le reste, des notes dans un format non spécifié, peuvent être utilisées afin de définir des contraintes temporelles.

Dans les diagrammes d'états, en plus d'*Event.ReceiveTime* et d'*Event.SendTime*, les primitives *When* et *After* peuvent être utilisées. *When* (date = janvier 1999) permet de spécifier un temps logique (ou échantillonné) absolu. *After* (x unités de temps) permet de représenter un temps relatif.

Il n'est donc pas possible, sans créer une extension à UML [Flake-2002a] [Sendall-2001] [Bodeveix-2002], de spécifier, par exemple : des événements ou des activités périodiques, de donner des informations probabilistes, de définir des relations entre temps physiques et temps logiques, de différencier simplement les CT internes et externes… Dans le chapitre suivant (§ I.2.4), les besoins que nous avons identifiés concernant la modélisation des contraintes temporelles sont détaillés.

Conscient de ce fait, l'OMG a lancé le RFP « *UML Profile for Schedulability, Performance and Time* » qui définit, entre autres, une syntaxe plus riche à UML sur cet aspect temporel. Une réponse [OMG-2002d], en phase finale d'adoption, est disponible. Les exemples donnés portent

[10] Idéalement, l'indépendance temporelle veut que l'objet soit défini indépendamment du contexte dans lequel il sera appelé et quel que soit l'instant de l'appel. L'indépendance spatiale traduit le fait que tous les aspects relatifs à l'objet lui appartiennent physiquement. Cette propriété facilite la localisation et la trace (*traceability*).

sur un enrichissement des diagrammes de séquence et d'activités par un ensemble de notes précisant le comportement temporel des éléments (événements et actions) de ces diagrammes (périodicité, durée, probabilité…). Nous avons pu vérifier que nos travaux sur les contraintes temporelles (en particulier sur leur représentation) sont compatibles avec cette réponse. Nous avons donc pu adapter, dans la mesure du possible, nos notations concernant les CT afin de respecter celles proposées par cette réponse. Cependant, cette adaptation ne reste que syntaxique et n'est pas complète pour trois raisons :

- La réponse est arrivée récemment et nous n'avons pas eu le temps de l'intégrer complètement à nos travaux.
- Elle modifie actuellement le méta-modèle UML1.4 pour faire apparaître la notion d'exécution d'action (ce qu'un profil UML n'est pas autorisé à faire, en théorie).
- La définition du temps proposée entre en contradiction avec des réponses à d'autres RFP (par exemple avec « *Action Semantics* »).

Toutefois, une révision de cette réponse est en cours pour le futur UML 2.0 et ces lacunes devraient disparaître.

## I.2.4. Problèmes abordés sur les contraintes temporelles

### I.2.4.1. Problèmes liés à la modélisation des CT

La plus importante difficulté concernant la modélisation des CT réside dans la grande diversité d'expression des relations d'ordre. Remarquons qu'elles sont données sous différentes formes (pas forcément exprimées en termes de contraintes temporelles explicites) et de différentes manières par les clients. Par exemple, le freinage d'une voiture peut être défini en terme de synchronisation des roues, en terme d'une vitesse différentielle tolérable entre les roues... Pour le système informatique, cela peut correspondre à la définition d'une contrainte temporelle synchronisant la vitesse angulaire des deux roues.

En outre, les CT externes peuvent être exprimées dans différents types de temps, tandis que les CT internes sont exprimées dans un temps informatique (échantillonné ou logique). Cette diversité dans l'expression des CT externes provient de l'hétérogénéité de l'environnement. Ce dernier peut effectivement être composé de différents équipements mesurant et utilisant le temps différemment. Certains d'entre eux manipulent un temps continu, d'autres un temps échantillonné, d'autres encore des séquences d'événements. Par ailleurs, le système informatique peut lui-même être décomposé en sous-systèmes, répartis sur plusieurs nœuds ne fonctionnant pas à la même cadence ou ne partageant pas une même horloge globale.

Une autre difficulté est liée à la nécessité d'exprimer certains besoins dans différentes échelles de temps. Par exemple, dans un atelier, une procédure automatique de diagnostic peut être lancée toutes les semaines, tandis que la commande d'un bras de robot sera exprimée en millisecondes. Cette granularité des temps impose de définir des correspondances ou des intervalles de correspondance entre ces différentes échelles. Il faudrait donc au moins disposer d'un formalisme permettant l'expression d'intervalles temporels (voir [Corsetti-1991] pour une description détaillée de cette problématique).

Une première solution est de disposer d'un formalisme unifié, suffisamment riche, de modélisation de ces différents types de temps. Ce formalisme doit idéalement pouvoir exprimer toutes les catégories de CT, qu'elles soient implicites, explicites, déterministes, probabilistes, sous formes de durées ou d'intervalles.

### I.2.4.2. Problèmes liés à la vérification des CT

L'analyste-concepteur devra naturellement veiller à ce que les CT internes qu'il définit,

respectent les CT externes. Or, établir des correspondances entre les CT externes et les CT internes est délicat. Une même CT externe peut être en relation avec une multitude de CT internes. Ainsi, dans le cas simple d'un temps de réponse entre un stimulus et une réponse, cette CT externe va être en relation avec plusieurs CT internes. Nous aurons des CT sur la phase de capture du stimulus, sa ou ses transformations successives et sa génération sous forme de réponse. L'identification de ces dépendances n'est pas simple ou immédiate. Il sera toujours difficile pour le concepteur d'envisager les conséquences de la modification d'une CT sur le reste des CT ou tout simplement de vérifier leur cohérence.

De plus, l'analyste-concepteur doit pouvoir disposer de moyens de dérivation de CT à partir d'autres CT, en termes de composition ou de décomposition. Cela lui permet de raisonner successivement sur des vues globales puis détaillées.

Par ailleurs, on retrouve les problèmes classiques de la synchronisation : le partage de ressources et les possibilités de famines et de blocages qui en découlent [Chang-1993]. Toutefois, puisque nous ne nous sommes intéressés qu'aux premières étapes du développement, nous n'avons pas abordé les problèmes de conception détaillée rencontrés lors de l'ordonnancement des activités sur des processeurs.

Pour un système de grande taille, il est indispensable que cette activité de vérification soit automatisée et repose sur un modèle formel. Les modèles, aptes à répondre à l'ensemble des critères énumérés, sont maintenant présentés.

## I.2.5. Les modèles formels pour le temps réel

Afin de présenter les modèles formels adaptés au traitement temporel, nous avons privilégié une taxonomie basée sur deux critères : le premier est lié aux techniques d'analyse offertes, le second aux paradigmes de modélisation. Par conséquent, un rappel sur les techniques d'analyse est d'abord effectué, puis la taxonomie est donnée. Enfin, notre choix pour un modèle formel conclut ce chapitre.

### I.2.5.1. Rappels sur les techniques d'analyses formelles

Il existe essentiellement deux types de techniques d'analyse formelle : celles fondées sur la démonstration de théorèmes (*theorem-proving*) et celles basées sur les évaluations de modèles (*model-checking*). Elles ont été étudiées intensivement dans la littérature et de nombreux algorithmes et outils ont été développés (voir par exemple [L'Her-1997]).

#### Les démonstrations de théorème

Avec les démonstrations de théorèmes, le modèle du système et les propriétés désirées sont exprimés comme des formules d'une logique [Clarke-1996a]. A partir de l'ensemble des formules représentant le système, les propriétés doivent être retrouvées par des mécanismes de déduction et de substitution. Aux différentes étapes de cette preuve, les formules représentant le système sont réécrites en respectant un ensemble de règles jusqu'à l'obtention de la propriété.

Cette technique est puissante : elle permet de prouver des propriétés sur des systèmes de toute taille (à la limite infinie). Cependant, si elle permet de montrer qu'un système est correct, elle n'indique pas clairement les erreurs lorsqu'elles existent. C'est donc une méthode utilisée généralement en dernière phase de vérification.

Le plus gros problème pour son utilisation est qu'elle n'est pas automatisable (et ne le sera jamais[11]). C'est pourquoi la technique entièrement automatisée des évaluations de modèles est généralement préférée [Clarke-1996b].

[11] Le mathématicien Gödel a démontré qu'il est impossible de trouver un ensemble de règles de déductions et d'axiomes qui permettent de déduire automatiquement l'ensemble des vérités du calcul arithmétique.

### Les évaluations de modèles (*model-checking*)

Avec les évaluations de modèles, le système à vérifier est d'abord décrit dans un langage parallèle de haut niveau. Ensuite, cette description est traduite vers un modèle sous-jacent qui est généralement un système de transitions étiquetées. Ce graphe (ou automate) contient, éventuellement avec certaines abstractions, tous les comportements possibles du programme. Finalement, les propriétés de bon fonctionnement de l'application, exprimées dans un formalisme approprié (automates, logiques temporelles...) sont vérifiées sur le système de transitions étiquetées à l'aide d'outils appelés évaluateurs (*model-checkers*).

La technique de *model-checking* est automatique. Elle fournit des informations utiles même si le système n'est pas complètement spécifié. Elle donne un contre-exemple lorsqu'une propriété n'est pas vérifiée. Elle s'avère particulièrement utile dans les premières phases du processus de développement, quand les erreurs sont encore nombreuses.

Toutefois le *model-checking* est confronté à l'explosion du nombre d'états modélisant le système, car les algorithmes sont de l'ordre de la taille des systèmes. Les techniques de *model-checking* sont donc souvent restreintes à des systèmes ayant un nombre fini d'états.

## I.2.5.2. Taxonomie des modèles formels

#### I.2.5.2.a) Par rapport aux techniques d'analyse

Trois grands groupes peuvent se dégager[12] [Bucci-1995] :

- Les modèles déclaratifs, qui se fondent sur les notations mathématiques (axiomes, clauses,…) et qui donnent une vue abstraite de l'espace d'état à l'aide d'équations logiques et algébriques. Le système est décrit en spécifiant ses propriétés globales. Ce type d'approche privilégie l'analyse formelle de type démonstration de théorème. Dans cette approche se retrouvent : les logiques temporelles (RTL [Jahanian-1986], TLA [Lamport-1994], TRIO+ [Mandrioli-1992],…), les langages déclaratifs à flots de données comme LUSTRE [Halbwachs-1991] ou SIGNAL [Benveniste-1990], les méthodes algébriques de type VDM [Jones-1993], VDM++[Dürr-1995], Z [Spivey-1989]…
- Les modèles opérationnels (ou impératifs), qui se définissent en terme d'états et de transitions. Ils sont, par conséquent, intrinsèquement exécutables. La simulation est généralement possible et des analyses formelles (par *model-checking)* sont généralement proposées. Dans cette famille se retrouvent : les formalismes comme les RdP [Petri-1962], SDL [Union-1992], les modèles basés sur CCS de Milner [Milner-1980] et sur CSP de Hoare [Hoare-1978] comme les algèbres de processus telles que LOTOS [Eijk-1989] ou RT-LOTOS, les automates temporisés, les dérivés formels (comme Modecharts [Jahanian-1988]) des Statecharts [Harel-1987], les langages impératifs de type ESTEREL [Berry-1992]….
- Les modèles duaux, qui essaient d'intégrer à la fois les caractéristiques des modèles opérationnels et déclaratifs en les faisant cohabiter. ESM/RTTL [Ostroff-1987], qui couple des machines étendues à la logique temporelle d'intervalles, est représentative de ce type d'approche.

#### I.2.5.2.b) Par rapport aux paradigmes de modélisation

Afin de simplifier les tâches de vérification et de validation du système, diverses hypothèses simplificatrices de modélisation, concernant essentiellement les CT, peuvent être émises. Suivant le respect de ces hypothèses de modélisation, trois paradigmes de synchronisation peuvent être distingués [Carcagno-1995] :

- Paradigme synchrone fort : Dans cette approche, les actions et les communications sont considérées comme immédiates et de dynamiques négligeables par rapport à la dynamique du

[12] Précisons que cette classification n'en est qu'une parmi un grand nombre (cf. par exemple [Hoare-1987], [Ghezzi-1991], [Clarke-1996a] ou [Wirsing-1993]).

procédé. Le temps manipulé est défini par rapport aux arrivées successives des événements. C'est une vision multiforme du temps, car il y a autant de temps que d'événements externes indépendants. Le système évolue par l'arrivée d'événements. La phase de transition (ou transitoire) entre deux états est considérée comme immédiate. Par ailleurs, on suppose une communication synchrone par diffusion large (*broadcasting*) : toutes les communications sont diffusées à tous les composants. Le système est donc un système réflexe (les réactions sont supposées instantanées).

- Paradigme synchrone faible : Cette approche, par rapport au synchrone fort, permet de mieux maîtriser les temps de communication et d'exécution. Ceux-ci ne sont plus supposés nuls, mais sont uniformisés et fixés arbitrairement à une même constante. Le comportement du système peut donc se synthétiser par une suite temporelle d'instants équidistants, le système réagissant aux changements survenus depuis le dernier instant. Pour fonctionner, un tel système doit être cadencé par une horloge globale (ou un ensemble d'horloges locales synchronisées). Le temps de réaction correspond alors à une unité de temps de cette horloge. Une communication par *broadcasting* est là aussi supposée. Le système n'est plus réflexe mais périodique.
- Paradigme asynchrone : Avec ce paradigme, la durée des actions et les temps de communication sont considérés comme bornés (mais peuvent varier dans des intervalles). Les délais de transitions du système ne sont pas supposés constants (une transition pouvant prendre un nombre variable de cycles de calcul). Lorsque des synchronisations sont nécessaires entre nœuds, elles sont faites de manière explicite par des communications. La communication par diffusion n'est plus supposée par défaut.

Ainsi, dans les approches synchrones, nous retrouvons les logiques temporelles, les langages de type SIGNAL, ESTEREL, LUSTRE, les dérivés de Statecharts…

Avec le paradigme asynchrone, nous retrouvons les algèbres de processus (CSP, SDL), les langages déclaratifs de type ELECTRE [Cassez-1995] ou les RdP.

Notons que, dans un contexte de système distribué, il peut y avoir une utilisation de modèles synchrones, pour la description du comportement des nœuds et, de modèles asynchrones pour la description des communications entre les nœuds (voir par exemple [Carcagno-1995]).

### I.2.5.3. Notre choix : les Réseaux de Petri

Concernant les techniques d'analyse, nous préférons les modèles opérationnels pour plusieurs raisons :

- Ils permettent d'obtenir rapidement des modèles exécutables ; ils offrent donc des possibilités de simulation, de vérification et d'évaluation des performances très tôt dans le cycle de développement.
- Les techniques d'analyses (par *model-checking*) sont automatisées.
- Les STR étant déjà complexes par nature, nous voulons limiter l'utilisation de familles de modèles différentes, afin de ne pas y ajouter une complexité d'étude.

Une grande partie de notre problématique touchant le traitement du temps, nous préfèrerions utiliser des modèles formels ne faisant pas d'hypothèse simplificatrice et donc suivant un paradigme de modélisation asynchrone.

Cet ensemble de critères permet de restreindre notre choix des modèles formels à la famille des algèbres de processus, à des langages déclaratifs asynchrones ou à des RdP.

Parmi ces modèles formels asynchrones, nous avons choisi les RdP car :

- Ils disposent de nombreux outils d'aide, de vérification et de validation.
- Leur spectre d'utilisation est très large sur le cycle de vie.
- Ils offrent des facilités graphiques à exprimer et vérifier les principes de synchronisation

et de parallélisme.

- Leur utilisation et les nombreux travaux existants dans différents domaines[13] sont garants de la qualité du formalisme et de son adaptation à la grande majorité des problèmes des STR.
- Tous les modèles formels adaptés à l'expression du comportement dynamique reposent (directement ou indirectement) sur une description états/transitions du système. Or, les RdP sont l'un des formalismes états/transitions possédant des mécanismes puissants d'abstraction et de description.
- Un grand nombre de modèles de RdP manipulant le temps de manière explicite a été défini. De ce fait nous avons la garantie de disposer d'un modèle ayant une sémantique riche concernant l'expression des contraintes temporelles.
- Enfin, notre équipe de recherche dispose d'une longue expérience et de travaux antérieurs sur ce formalisme.

Néanmoins, ainsi qu'il a été annoncé en § I.2.1.3, l'utilisation conjointe d'un modèle formel avec la notation UML soulève de nombreuses interrogations. Celles-ci sont maintenant détaillées.

## I.2.6. Principes d'intégration d'UML et de modèles formels

L'idée de coupler des modèles semi-formels et formels n'est pas récente et a été fortement développée dans les années 90. Ces démarches, nommées approches mixtes, ont ainsi résolu de nombreuses difficultés [Andre-1995b] et déterminé un certain nombre de pratiques saines.

### I.2.6.1. Principes généraux des approches mixtes

#### I.2.6.1.a) A propos du processus de développement

Un grand nombre d'expérimentations a été mené sur le processus de développement et le moment où les modèles formels et semi-formels sont utilisés. En effet, bien que les approches mixtes permettent toutes de passer de l'informel au formel, en se servant de représentations intermédiaires semi-formelles, deux grandes stratégies de traduction peuvent être distinguées [Fraser-1994] :

- Le mode séquentiel, où le modèle semi-formel est entièrement établi avant de réaliser la modélisation formelle. Il n'y a pas de retour possible vers la représentation semi-formelle.
- Le mode parallèle, où les représentations formelles et semi-formelles sont établies simultanément et enrichies peu à peu par des affinements successifs.

Par exemple, dans les démarches transitionnelles avec mode séquentiel, nous trouvons les méthodes comme : SA-RT/PNO [Benzina-1997] et SOMT [Ek-1995]. Les spécifications SA-RT ou OMT déjà établies sont traduites en RdP ou SDL afin de vérifier la modélisation et d'automatiser en grande partie le passage à une conception.

Dans une démarche utilisant un mode parallèle et reposant sur une approche fonctionnelle, il y a, par exemple, SYS/PO et ESML [Bruyn-1988]. Avec une approche OO, l'offre est plus large [Andre-1995a]. Basées sur la notation OMT, il y a : OORT [Leblanc-1996], CO-OPN2/OMT [Biberstein-1997], OMT2/RTGOL [Sourrouille-1998]... Avec la méthode HOOD, signalons : HOOD/PNO [Paludetto-1991], Hrt-HOO/VDM++ [Lano-1997]…

Il semblerait que le mode parallèle soit le plus approprié pour la modélisation de grands systèmes dans lesquels les problèmes sont mal connus ou difficiles à appréhender [Kemmerer-1990] [France-1994]. R. Kemmerer a détaillé une démarche, appelé "mode intégré", qui est un

[13] Nous pouvons citer : l'analyse, la conception, les activités de test, l'évaluation de performance, l'étude de protocole, la sûreté de fonctionnement, l'ordonnancement, la supervision de procédé industriel…

raffinement du mode parallèle. Des allers et retours entre représentations semi-formelles et formelles sont effectués, chacun des formalismes permettant d'enrichir des aspects de l'autre. Du fait des caractéristiques des STR que nous étudions, nous avons choisi cette approche. Cette utilisation du mode intégré, nommée « d'intégration des modèles formels » [Ledang-2002] [Meyer-2001], peut se faire de plusieurs manières.

#### I.2.6.1.b) Intégration des modèles formels dans une modélisation Objet

Nous distinguons deux grandes approches d'intégration :

- L'intégration par adjonction : La description semi-formelle du modèle OO est complétée par une description formelle de certains aspects statiques ou dynamiques. Il n'y a pas, à proprement parler, de liens ou de relations explicites entre les modèles formel et semi-formel. L'analyste-concepteur doit alors être suffisamment compétent pour s'assurer que la modélisation formelle respecte la description semi-formelle. Cette approche apporte donc naturellement de la rigueur dans le développement, mais la cohérence de la modélisation et le respect des besoins reposent en grande partie sur la qualité de l'analyste-concepteur.
- L'intégration par dérivation : Elle consiste à générer, automatiquement ou non, les modèles formels à partir des représentations semi-formelles. Naturellement, des « techniques de passage » du semi-formel vers le formel doivent être déterminées. Il s'agit de techniques de « passage » et non de traduction[14], car les représentations formelles et semi-formelles ne sont pas, par définition, équivalentes mathématiquement. Cela implique, au minimum, la définition d'un ensemble de règles de translation expliquant comment les éléments de la modélisation semi-formelle sont reportés sur le modèle mathématique.

Dans les deux cas, la communauté scientifique a du adapter les langages formels existants afin qu'ils soient compatibles avec les concepts OO, soit en enrichissant leur syntaxe par des mécanismes objets (Object-Z [Duke-1994], Z++ [Lano-1991], VDM++ [Dürr-1995]), soit en définissant des règles d'utilisation et de représentation des principes à objets. De ce fait, de nombreux travaux peuvent être directement utilisés ou légèrement adaptés pour la notation UML. En § I.3.3, lorsque des approches basées sur les RdP seront détaillées, leurs différentes stratégies d'adaptation au paradigme OO seront aussi présentées.

Nous avons privilégié une approche d'intégration par dérivation, car elle est plus en accord avec notre souci de proposer une démarche utilisable par un ingénieur, en encapsulant les modèles formels derrière une représentation semi-formelle plus simple. Cette dérivation d'UML peut, elle aussi, se faire de plusieurs façons, détaillons-les.

### I.2.6.2. Quelles techniques de dérivation d'UML ?

#### I.2.6.2.a) Dérivation du modèle, du méta-modèle ou du méta-méta-modèle UML ?

En raison de la structuration en couches d'UML, les techniques de dérivation peuvent être définies à différents niveaux. En conséquence, trois techniques sont utilisables :

- Dérivation du modèle : une correspondance entre chaque élément du modèle UML et des constructions du langage formel est établie [Bruel-1998].
- Dérivation du méta-modèle : le langage formel est utilisé pour formaliser le méta-modèle UML [Reggio-2001] [Alvarez-2001] [Evans-1999] [Gogolla-2000].
- Dérivation du méta-méta-modèle : le langage formel est utilisé pour spécifier, à la fois, le méta-méta-modèle, mais aussi et surtout les règles d'instanciation du méta-modèle UML [Fernández-2001].

[14] Néanmoins, dans la reste de ce mémoire, nous utiliserons parfois le mot « traduction », qu'il faudra alors considérer avec le sens de « passage », de « translation » ou de « dérivation ».

Ces trois techniques de dérivation sont différentes et ne desservent pas les mêmes objectifs [Fernández-2000]. En effet, la dernière approche vise plutôt à formaliser et détecter des incohérences entre des évolutions d'UML ou des profils UML, tandis que les deux premières sont focalisées sur une version particulière d'UML et sont donc plus proches de nos préoccupations.

Pour notre part, et comme la plupart des chercheurs travaillant sur cet aspect, nous avons préféré la première approche, à savoir la dérivation du modèle. Elle nous apparaît effectivement plus pragmatique, plus simple à mettre en œuvre et à comprendre. Ainsi, l'identification des incohérences des spécifications semi-formelles est facilitée, car les éléments des modèles sont en relation directe. En outre, dans ce premier temps de la définition de notre approche, nous préférons nous focaliser sur l'identification des grandes règles de traduction, ainsi que sur des techniques de vérification, plutôt que sur la formalisation et la définition d'un langage de transformation[15]. Ce choix est possible en travaillant au niveau du modèle mais ne l'est plus si nous travaillons sur une dérivation du méta-modèle.

Enfin, un dernier argument concerne l'automatisation ou non du procédé de dérivation. Afin de mieux motiver notre raisonnement, nous détaillons cette question.

#### I.2.6.2.b) Dérivation automatique ou semi-automatique ?

A l'instar d'autres travaux [France-1994] [Baresi-2001a], nous pensons qu'une automatisation du procédé de dérivation est préjudiciable aux approches mixtes lorsqu'elle est appliquée très tôt dans le cycle de développement.

En effet, l'un des principaux avantages d'UML est de faciliter le difficile passage entre une description informelle des besoins vers une formalisation intermédiaire (spécification). L'analyste-concepteur doit pouvoir laisser en suspens un certain nombre de questions et de détails afin de se concentrer sur une capture globale des besoins. Les représentations semi-formelles d'UML sont donc toutes indiquées et favorisent un meilleur dialogue avec les clients.

En automatisant le procédé de dérivation, plusieurs risques apparaissent. Puisqu'une traduction automatique utilisera systématiquement une et une seule interprétation, se pose le problème du choix de l'interprétation. En outre, les interprétations varient en fonction du domaine étudié, de l'équipe de développement, de la culture d'entreprise, de la formation UML reçue… En sélectionnant toujours la même interprétation, il y a risque d'une mauvaise traduction des besoins. Cela réduit aussi le pouvoir d'expression d'UML ainsi que sa flexibilité.

De plus, tout passage d'une représentation semi-formelle à une modélisation formelle nécessitant des compléments d'information, comment seront-ils apportés dans une traduction automatique ? Il est à craindre qu'UML ne se différencie alors plus d'un langage formel que par son aspect graphique et qu'il y ait donc perte des bénéfices d'une approche mixte.

Pour ces raisons, nous plaidons, dans les premières phases du développement, pour un procédé de dérivation semi-automatique. Son rôle est de guider l'analyste-concepteur lors du passage à une modélisation formelle, en lui proposant des alternatives de traduction lorsqu'il y a ambiguïté ou en lui demandant des informations complémentaires.

C'est aussi pourquoi, nous avons préféré travailler sur des dérivations du modèle UML, plutôt que sur celles du méta-modèle. En effet, il nous a semblé que certaines ambiguïtés d'UML ne

[15] Cet argument peut être vu au contraire comme un inconvénient majeur, nous en convenons. La formalisation des règles de transformation proposée dans ce mémoire reste l'une des perspectives prioritaires des futurs travaux sur UML/PNO. Nous verrons cependant que cette formalisation est loin d'être triviale, car nous ne croyons pas en une traduction automatique ainsi que nous l'expliquons ci-après.

peuvent être résolues que par rapport au modèle et non au méta-modèle.

Nous venons de motiver nos choix de recherche. Nous allons maintenant présenter des travaux de recherche similaires au nôtre, portant sur la définition d'approches mixtes UML focalisées sur l'aspect comportemental et utilisant des modèles formels asynchrones.

# I.3. Les approches mixtes asynchrones UML pour le temps réel

## I.3.1. Introduction

Peu de temps après la normalisation d'UML, de nombreux chercheurs se sont intéressés à sa formalisation ou à son utilisation dans une approche mixte. Cet intérêt de la communauté scientifique pour UML n'a fait que croître. Pour cette raison, dresser un état de l'art exhaustif de toutes les approches existantes serait illusoire.

En effet, pour la quasi-totalité des modèles formels présentés précédemment, il existe des travaux de couplage avec UML. Ainsi, des propositions sont disponibles pour : les langages Z [Bruel-1998] [Evans-1998] [France-1997], B [Ledang-2002] [Meyer-2001], SIGNAL [Le Guernic-1997] [Beauvais-1999], VDM [Lano-1997], Larch [André-2000], des logiques temporelles [Lavazza-2001] [Bodeveix-2002] [Flake-2002b], des automates temporisés [Toetenel-2001] [Roubtsova-2001]...

Des plates-formes logicielles ont été développées et sont suffisamment matures. Parmi celles-ci, citons : Esterel Studio, UMLAUT (*Unified Modelling Language All pUrposes Transformer*) [Ho-1999] [Le Guennec-2001], MetaEnv [Baresi-2002], VIATRA [Csertán-2002]…

Des communautés de chercheurs se sont rassemblées, soit dans des projets internationaux (SACRES [Esprit-1999], NEPTUNE[Irit-2001], HIDE [Esprit-1998], ITEA DESS [ITEA-1999], WOODDES [Information Society Technologies-1999], DepAuDE [Information Society Technologies-2001]), soit dans des groupes de travail (« precise UML[16] », « 2Uconsortium[17] »)…

Nous avons donc préféré limiter notre étude à des démarches similaires à la nôtre, basées sur des modèles asynchrones et adaptés au traitement des STR. Il s'agit principalement d'approches utilisant des algèbres de processus ou des RdP. Nous détaillons plus particulièrement celles reposant sur les RdP.

## I.3.2. Les approches basées sur des algèbres de processus

En regard du grand nombre d'approches disponibles, nous avons choisi de ne détailler que les plus représentatives :

- Soit par leurs avancées dans le couplage UML et modèle formel pour les STR,
- Soit car elles sont supportées par des outils informatiques ou même des AGL.

Il s'agit de SDL/UML, de vUML et de Turtle.

### I.3.2.1. UML et SDL

L'approche UML/SDL est certainement la plus ancienne et la plus mature en terme d'utilisation industrielle. Elle est supportée par l'AGL Tau [Telelogic-1999]. Elle utilise le langage

[16] *Precise* UML (http://www.puml.org/). Ce groupe a fourni tout un travail de formalisation des aspects statiques d'UML.

[17] *2Uconsortium* (http://2uworks.org/). Ce groupe définit un méta-modèle UML formel [Evans-1999] en vue de soumettre une réponse au RFP UML2.0

formel SDL (*Specification Description Language*) normalisé par l'ITU[18].

UML/SDL est issue de la méthode SOMT (nommée aussi OORT [Leblanc-1996]) qui couplait OMT et SDL. L'ITU vient de définir deux normes expliquant l'utilisation d'UML et de SDL. La première (Z100) est la définition de SDL 2000 et précise le couplage avec UML. La seconde (Z109) spécifie comment un modèle UML est traduit en SDL.

SDL est basé sur une algèbre de processus beaucoup utilisée dans le domaine des télécommunications et la modélisation des protocoles. Les contraintes de temps ne peuvent être modélisées que par des « *time-out* » et les temps de communications que par l'utilisation de « *delayed channel* ». Ces derniers sont des canaux de communications entre processus retardant simplement la communication d'une durée indéterminée.

Beaucoup d'efforts ont été menés par l'ITU pour intégrer à SDL les notions de type et d'héritage, afin de le rendre compatible avec une approche OO. Néanmoins, l'origine fonctionnelle de SDL reste encore très présente. Son couplage avec UML reste encore difficile [Babau-1998] et seuls les diagrammes de classes et de séquence sont effectivement utilisés.

Le couplage réside essentiellement dans l'utilisation d'UML lors de la phase de spécification et dans la traduction (ITU Z109) ultérieure des diagrammes de classes UML vers SDL. Il reste alors à compléter, en phase de conception détaillée, le modèle décrit en SDL. A partir de ce dernier, des activités de vérification, de génération automatique de code et de test peuvent être menées.

Concernant les activités de validation, des diagrammes de séquence (ou plus exactement des « *Message Sequence Chart* ») peuvent être générés. Il n'est pas possible de revenir sur les diagrammes UML de manière automatique. Aucun lien avec les diagrammes d'états/transitions ou d'activités ne sont définis. La recherche automatique de la cohérence entre les diagrammes UML et SDL n'est pas non plus possible. UML/SDL s'apparente plus à une approche mixte à mode séquentiel qu'à mode intégré (ou parallèle).

Il reste que c'est la seule proposition industrielle basée sur un modèle formel asynchrone qui permette de mener des activités de vérification et de validation avec des contraintes temporelles qualitatives. Par contre, lorsqu'il y a besoin de manipuler des contraintes temporelles quantitatives ou mener des activités d'évaluation de performances, SDL reste très limité. Le passage au monde SDL se faisant relativement tard dans le cycle de développement, UML/SDL est donc plus une méthode de conception que de spécification.

#### I.3.2.2. vUML

vUML [Lilius-1999b] est un outil transformant certains diagrammes UML (diagrammes d'états/transitions et de collaboration) en une description dans le langage PROMELA, afin qu'ils soient analysés par l'évaluateur de modèle SPIN [Holzmann-1997]. PROMELA est une algèbre de processus, avec un paradigme de modélisation asynchrone, centrée sur la description des communications inter-processus. Spin est un évaluateur de modèles pour lequel les propriétés à vérifier sont exprimées avec la logique temporelle LTL (*Linear Temporal Logic*). Notons que les vérifications de ce *model-checker* ne permettent pas de manipuler des CT quantitatives telles que des deadlines ou des temps de réponses.

Dans le but d'automatiser la traduction des diagrammes UML, la sémantique des diagrammes d'états/transitions a été réduite [Lilius-1999a]. Elle reste proche du RTC (*Run to Completion*) d'UML (UML 1.3 pp2-149). Les gardes et les actions des transitions ainsi que les activités des états doivent directement être écrites en PROMELA. La notion d'historique et d'états orthogonaux n'est pas utilisable.

[18] *International Telecommunication Union* (www.itu.int)

En fait, à chaque diagramme d'états/transitions est associé un processus PROMELA. Afin de pouvoir faire une évaluation de modèles, vUML a besoin de connaître les instances des processus et leurs liens de communication (les canaux). Cette connaissance est apportée par un diagramme de collaboration. Cette description est statique et ne permet pas, pour l'instant, de modéliser la création dynamique de processus. Une évaluation du modèle peut ensuite être faite. Elle permet de détecter : étreintes mortelles, famines, violations de contraintes et dépassements de pile. Dans le cas d'une erreur, un ou plusieurs diagrammes de séquence peuvent être générés.

Quoique au stade de prototype de recherche, l'outil vUML est déjà utilisable sur des cas d'étude simple [Lilius-1998]. Signalons que d'autres équipes de recherches travaillent sur un formalisme équivalent avec les mêmes outils de vérification [Darvas-2002], HUGO [Schäfer-2001].

Néanmoins, les mêmes critiques que pour l'approche UML/SDL peuvent être adressées. Puisqu'elles sont toutes basées sur une algèbre de processus manipulant seulement des « *time-out* » et des « *delayed channel* », la manipulation de contraintes temporelles quantitatives reste difficile.

#### I.3.2.3. Turtle

TURTLE (*Timed Uml and RT-Lotos Environment*) [Saqui-Sannes-2000] [Saqui-Sannes-2001] est un profil UML définissant des extensions temps réel pouvant ensuite être traduites dans le langage RT-LOTOS [Courtiat-2000]. Ce dernier étend l'algèbre de processus LOTOS [Bolognesi-1987] par trois opérateurs temporels (attente limitée, délai déterministe et délai non déterministe). Il peut ainsi exprimer des intervalles temporels.

Le profil TURTLE spécialise les diagrammes de classes et les diagrammes d'activités UML afin de pouvoir traduire directement ces extensions en RT-LOTOS. Ainsi, une classe stéréotypée <<Tclass>> permet de modéliser les objets actifs et les communications de contrôle (les portes des processus LOTOS). Par ailleurs, le langage des diagrammes d'activités est étendu par ajout de connecteurs spécifiques à RT-LOTOS.

Il s'agit plus d'une démarche d'intégration par adjonction que par dérivation. En effet, l'extension TURTLE peut plutôt être vue comme une représentation graphique de RT-LOTOS à la mode UML. Peu de liens sont définis entre les éléments UML non TURTLE et RT-LOTOS. Par exemple, seules les communications modélisées par des portes sont prises en compte. Les communications classiques UML, par invocation d'opérations, sont simplement ignorées.

Les techniques de vérification présentées sont basées sur le calcul du graphe d'accessibilité des états. Ce graphe aide ainsi à identifier des situations de blocage et à raisonner sur les différents états accessibles du point de vue logique, mais aussi temporel.

D'autres travaux sont en cours sur LOTOS ou sur ses extensions temps réel : E-Lotos [Clark-2000] [Hernalsteen-1998] et les travaux [Carreira-2000] utilisant la boîte à outils CADP [Fernandez-1996]. Toutefois, leurs intégrations des contraintes temporelles ne sont pas aussi développées que dans TURTLE.

### I.3.3. Approches UML basées sur des réseaux de Petri

Avant de présenter les approches UML basées sur les RdP, nous faisons un rappel les différentes techniques de couplage des RdP avec une modélisation OO ainsi que sur les classes de RdP permettant la manipulation explicite de CT. Nous supposons que le lecteur possède une connaissance du formalisme des RdP.

#### I.3.3.1. Les RdP et la modélisation OO

##### I.3.3.1.a) Rappels sur la modélisation par RdP

Modéliser un système complexe, à l'aide de RdP, consiste à structurer ce système en deux parties [Valette-2000] : une partie « contrôle » et une partie « données ». La partie « contrôle » décrit essentiellement les enchaînements possibles des activités. La partie « données » correspond à l'ensemble des structures de données internes au système et des calculs effectués sur ces données.

Le RdP décrit la structure de contrôle du système. La modélisation des données est souvent réalisée par des jetons ayant une structure de données. Elle nécessite alors l'utilisation de RdP dits de haut niveau (RdP Colorés [Jensen-1981], Prédicat/Transition [Genrich-1986] ou Objet [Sibertin-Blanc-1991] [Lakos-1990 ]).

L'interprétation du RdP revient à spécifier les liens entre la partie « contrôle » et la partie « données ». On peut la traduire en associant :

- Des conditions aux transitions (ainsi que sur la partie structure de données du jeton).
- Des actions aux transitions (entraînant par exemple la modification des données du jeton).

Rappelons que le RdP n'est pas un outil facilitant la structuration de système. Il n'apporte qu'un seul concept dans l'organisation dudit système : celui de permettre une délimitation claire entre la structure de contrôle (décrite par le graphe lui-même) et la structure de données (décrite en complément du graphe et constituant l'interprétation du réseau de Petri). Une approche hiérarchisée est certes possible [Valette-1979], mais elle doit résulter d'une volonté du concepteur et s'appuyer sur des règles et des méthodes extérieures au RdP.

Pour cette raison, la structuration du système est souvent apportée par un autre mécanisme que les RdP. C'est pourquoi, différentes méthodes ont été définies afin de coupler des RdP à des approches OO.

#### I.3.3.1.b) Le couplage RdP et OO

Différentes études comparatives des couplages entre RdP et l'approche OO ont été effectuées [Zapf-1999] [Lakos-1997] [Valk-2000]. Trois grandes démarches sont identifiées [Bastide-1995] :

- Intégration des concepts à objets dans les RdP.
- Intégration des RdP dans les concepts à objets.
- Intégration mutuelle des concepts OO et des RdP.

Dans la première catégorie, les RdP modélisent toute la structure de contrôle du système. Les jetons modélisent les objets, ces derniers étant généralement représentatifs des propriétés statiques du système. Nous pouvons citer dans cette catégorie : LOOPN [Lakos-1990 ], MacroNet [Keller-1994], MOBY [Fleischhack-1993], OBJSA [Battiston-1988], THORN [Schöf-1995], SimCom [Verkoulen-1998]… Cependant, ce type d'approches est rapidement confronté à la complexité de modélisation d'un système, les RdP modélisant à plat toute la structure de contrôle du système.

Dans la seconde catégorie, les RdP sont utilisés afin de décrire le comportement des objets actifs (et parfois passifs) ainsi que les communications inter-objets. Les RdP sont « à l'intérieur » des objets, on parle alors d'approches « Objet à RdP »(*Petri Net Object*). Différentes classes de RdP (prédicat/transition, colorés, etc.) peuvent être utilisées suivant la classe de système modélisée. Un grand nombre de formalismes est proposé dans cette catégorie, citons HOOD Nets [Di Stefano-1991], HOOD/PNO [Paludetto-1991], OOBM [Hanisch-1997], OBM [Kappel-1991], OCPN [Maier-1997], PAM [Bachatène-1995], PN-TOX [Holvoet-1995]… La structuration est en général supportée par une méthode OO (HOOD, OMT, UML…).

Dans la dernière catégorie, une classe de RdP particulière est développée afin de directement prendre en compte les aspects OO (héritage, instanciation…). Cette classe de RdP, souvent appelée « RdP à Objet » (*Object Petri Net*)[Sibertin-Blanc-1985;Sibertin-Blanc-1991] connaît diverses déclinaisons. Nous retrouvons des formalismes comme CLOWN [Battiston-1995], CO-OPN/2 [Biberstein-1997], HOON [Löwe-1995], LOOPN++ [Lakos-1995b], OPN [Valk-1996]… Ces modèles peuvent être couplés à des méthodes OO ou définir leur propre formalisme.

### I.3.3.2. Les classes de RdP adaptées au traitement de CT explicites

Un RdP modélise, par sa structure, des relations d'ordre entre les transitions et dès lors, des relations d'ordre entre tirs de transition. Cela permet donc une manipulation de CT qualitatives et un RdP ordinaire peut modéliser toutes les relations d'Allen [Allen-1983] [Sénac-1996]. Mais pour manipuler des CT explicites, il faut l'enrichir de considérations quantitatives.

Un RdP étant un graphe biparti, il est naturel de placer ces considérations, soit sur les places, soit sur les transitions, mais on peut aussi les placer sur les arcs des transitions ou sur les jetons. En outre, les contraintes temporelles peuvent être représentées par des durées (classe de RdP temporisé) ou par des intervalles (classe de RdP temporel). Enfin, ces CT peuvent être soit déterministes, soit stochastiques. Cette liberté de choix dans l'affectation des CT permet d'obtenir une large variété de RdP ayant des pouvoirs d'expression temporelle plus ou moins grands et des capacités d'analyse plus ou moins réduites.

Nous présentons brièvement les classes de RdP les plus usuelles que sont les classes de RdP temporisé, t-temporel et stochastique. Nous conseillons au lecteur, intéressé par une comparaison détaillée de ces différentes classes, la consultation de la thèse de Marc Boyer [Boyer-2001].

#### I.3.3.2.a) Réseaux de Pétri Temporisés (RdPTé)

##### *I.3.3.2.a.i) Définitions*

Deux classes de RdP Temporisé (RdPTé) (*Timed Petri Net*) se distinguent :

- Les p-RdPTé [Sifakis-1977] : Une durée est associée à chaque place. La sémantique de cette durée correspond au temps de séjour minimum d'un jeton (son temps d'indisponibilité) dans une place.
- Les t-RdPTé [Ramchandani-1974] : A chaque transition est associée une durée dont la sémantique correspond à la durée de tir de la transition. Le jeton ayant sensibilisé la transition n'est plus disponible pendant toute cette durée.

Ces deux classes de RdP sont, en fait, équivalentes [Sifakis-1979] et proposent la même idée : dissocier le marquage en deux parties : les jetons libres et les jetons indisponibles. Un jeton indisponible ne peut plus sensibiliser une transition et donc participer à un franchissement.

##### *I.3.3.2.a.ii) Critiques*

La notion d'indisponibilité des jetons est l'une des lacunes des RdPTé car il devient extrêmement difficile de modéliser des mécanismes préemptifs de type "chiens de garde". En effet, les jetons sont absorbés par les transitions (ou les places). Ils ne sont alors plus disponibles pour le tir d'autres transitions. En outre, rien dans la définition de ces modèles ne permet de forcer le tir d'une transition franchissable : le ou les jetons participant à sa sensibilisation peuvent rester indéfiniment dans leur place et la transition ne sera jamais tirée. Il est toutefois possible de définir un fonctionnement au plus tôt où la transition est tirée dès sa sensibilisation. Dans ce cas, il est possible d'établir un graphe de marquage de tir au plus tôt et de quantifier les durées des chemins. Cette technique d'analyse a été utilisée pour l'optimisation de système de production [Gaubert-1999], mais elle doit être couplée avec des heuristiques et des techniques de calculs

extérieures. En effet, la politique de tir au plus tôt n'est pas suffisante, par exemple, pour déterminer un parcours minimum (en ordonnancement, il est connu qu'il faut parfois attendre pour gagner ensuite du temps).

Un dernier inconvénient des RdPTé réside dans leur incapacité à exprimer la notion d'indéterminisme temporel. En effet, dans de nombreux cas (connaissance incomplète du système, gigue temporelle, variabilité temporelle de la durée d'un traitement), la durée d'un traitement n'est pas exactement connue et peut être comprise dans un intervalle de temps min-max.

I.3.3.2.b) Réseau de Petri à transitions temporelles (t-RdPTl)

Comme pour un t-RdPTé, un RdP t-temporel peut comporter des informations temporelles sur ces transitions $t_i$, mais ici exprimées par des intervalles temporels $[a_i,b_i]$ (avec $0 \leq a_i \leq b_i$).

Pour être franchie, une transition $t_i$ doit être sensibilisée de manière continue pendant le délai minimum $a_i$ avant de pouvoir être franchie. Elle ne peut rester validée au-delà du délai maximum $b_i$. Grâce à ce mécanisme, des incertitudes de tir d'une transition peuvent être exprimées. De plus, le jeton sensibilisant une transition reste disponible pour d'autres transitions.

Différentes sémantiques de fonctionnement d'un t-RdPTl sont possibles concernant :

- Le traitement de la multi-sensibilisation d'une transition et,
- La sémantique temporelle de tir des transitions.

*I.3.3.2.b.i) Sémantique de la multi-sensibilisation*

Dans le cas d'une multi-sensibilisation d'une transition, il faut en premier lieu préciser sa politique de sensibilisation. Sur la Figure I-2, deux jetons arrivent au même instant t=0 dans la place P1.

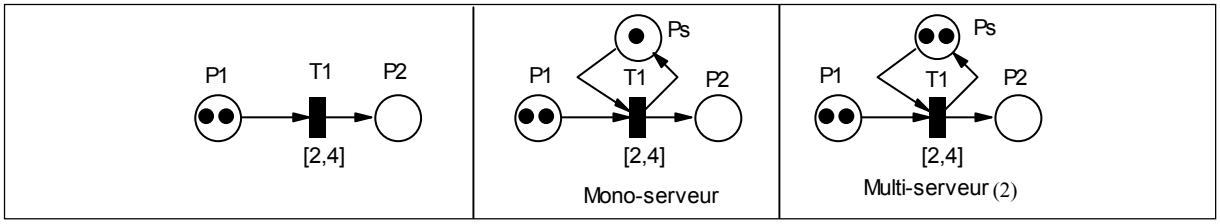


**Figure I-2 : Multi-sensibilisation et serveur**

Deux politiques peuvent alors être envisagées :

- La transition ne peut être multi-sensibilisée (politique mono-serveur) et seul un jeton pourra la franchir dans l'intervalle [2,4]. Une fois la transition tirée, elle sera alors de nouveau sensibilisée par le jeton resté dans la place P1,
- Les deux jetons pourront franchir la transition dans l'intervalle [2,4] (politique dite multi-serveur).

Ces politiques ont été explicitement représentées à l'aide de la place Ps sur la Figure I-2. Naturellement dans le cas de la politique multi-serveur, un nombre infini de jetons dans Ps peut être envisagé.

Une autre précision doit être apportée dans le cas d'une multi-sensibilisation. Nous supposons sur la Figure I-3 que deux jetons arrivent à la place P1 respectivement au temps t=0 et t=1. Sur le chronogramme donné, nous avons considéré une politique multi-serveur et indiqué les intervalles de tirs possibles de chaque transition. Il y a ambiguïté pour définir l'état des marquages futurs après le franchissement de la transition T2 au temps 4 (par exemple). Si l'on ne peut différencier les jetons, les deux jetons peuvent être alors choisis. Or, le choix de l'un ou de l'autre entraîne

alors une évolution différente du marquage. Cette ambiguïté doit être levée, le mécanisme le plus courant est d'associer (explicitement ou implicitement) à chaque jeton sa date d'arrivée dans la place. Dans ce cas, l'état du marquage peut être facilement défini.

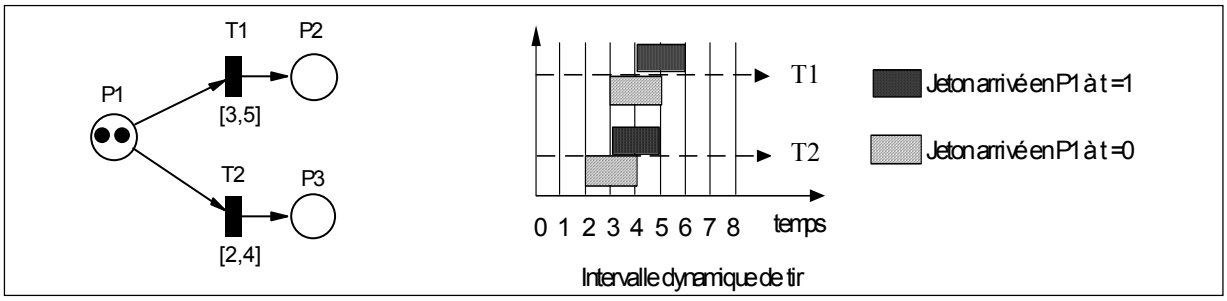


**Figure I-3 : Sensibilisation de transitions temporelles avec sémantique multi-serveur**

*I.3.3.2.b.ii) Sémantique de tir*

Concernant la sémantique de tir, deux interprétations sont possibles [Tsai-1995]:

- Une Sémantique de Tir Forte (STFO), où toute transition continûment sensibilisée doit être tirée avant sa borne temporelle supérieure $b_i$.
- Une Sémantique de Tir Faible (STFA), le tir d'une transition continûment sensibilisée n'est pas imposé.

La première sémantique est nécessaire pour la modélisation des chiens de garde (sinon, rien ne garantit le tir de la transition représentant l'expiration du délai d'attente). Mais cette politique STFO déroge à la "philosophie" des RdP. En effet, une telle politique de tir, implique une relation d'ordre entre les dates de tirs de toutes les transitions franchissables à un instant donné d'un RdP. Or, dans un RdP ordinaire, la décision de tirer ou non une transition est locale à chaque transition et est complètement indépendante des autres[19].

Ainsi, dans le cas de la STFO, l'ajout de ces relations d'ordres partiels entre transitions franchissables, peut laisser échapper des séquences de tir respectant les seules relations d'ordre imposées par le RdP (et non celles ajoutées par la STFO). Cet exemple est illustré par la Figure I-3 où la transition T2 sera systématiquement franchie avec une politique STFO si le jeton qui l'a sensibilisé, est resté pendant 4 unités de temps dans P1.

Toutefois, c'est toujours la STFO qui est utilisée dans la modélisation de STR ou réactif, en raison principalement de sa capacité à modéliser des mécanismes de type chien de garde.

*I.3.3.2.b.iii) Technique de vérification des t-RdPTl*

Il n'est pas possible de construire directement le graphe des états accessibles d'un RdP temporel. En effet, chaque transition temporelle franchissable est susceptible d'être tirée à chaque instant de son intervalle temporel. Dès lors, le graphe des états est infini.

Une stratégie alors possible afin d'éviter cette explosion est de choisir un instant de tir dans chaque intervalle (instant au plus tôt, au plus tard ou à un temps moyens[20]). Cependant, cette politique n'est pas complètement satisfaisante. Nous sommes certain de n'avoir qu'une partie des états accessibles du RdP et, plus grave encore (pour une analyse des contraintes temporelles), nous ne pourrons pas calculer les pires cas (ou seulement dans des cas très simples).

Par conséquent, la construction du graphe d'accessibilité doit passer par des regroupements d'états. Différentes stratégies de regroupement d'état ont été définies. La plus connue reste celle proposée par le graphe de classe des états [Menasche-1982]. Elle consiste à regrouper dans un même nœud tous les états accessibles suite au franchissement d'une même séquence de

[19] Sauf, bien évidemment, lorsque les transitions sont en conflit effectif.

[20] Ce qui est le principe des graphes d'état probabilisés

transitions mais à des dates différentes. On obtient ainsi un graphe dont les nœuds sont des classes d'états. Les arcs reliant les nœuds sont étiquetés par des transitions qui permettent de passer d'un nœud à l'autre. De ce fait, tous les états représentés dans une classe ont le même passé (en terme de transitions franchies, mais non d'instants de franchissement) mais pas forcément le même avenir. En effet, une attente plus longue dans le passé peut avoir rendu obligatoire le franchissement d'une transition en premier, alors que pour un autre état de la même classe, le choix reste ouvert avec d'autres transitions. L'algorithme a été amélioré et permet maintenant de traiter des RdP t-temporel non sauf [Berthomieu-2001] [Boucheneb-1999]. Cette technique permet donc d'obtenir une sorte de graphe d'accessibilité des états du RdP étudié. Elle reste néanmoins lourde en temps de calcul, rapidement confrontée à l'explosion combinatoire des états et ne peut porter que sur un graphe fini d'état.

Par ailleurs, la quantification temporelle des chemins entre deux marquages du graphe de classe ne permet pas de déterminer des durées exactes et réelles. En effet, en raison même des principes de construction du graphe de classe, il a été montré [Pradin-chézalviel-1999a] qu'en présence de parallélisme dans le RdP, le graphe de classe donne des intervalles temporels pessimistes, beaucoup plus grands que ceux réellement présents.

Cependant, la classe des RdP temporels ne permet pas de modéliser des informations probabilistes. Or, ce type d'informations est parfois nécessaire pour modéliser, par exemple, des défaillances ou des débits d'arrivée d'informations. Afin de pallier ce défaut, les RdP stochastiques ont été définis.

I.3.3.2.c) Les réseaux de Petri stochastiques

Les réseaux de Petri stochastiques peuvent associer une probabilité de tir à chacune de leurs transitions. Au lieu de considérer, comme dans les t-RdP temporels, que les instants de l'intervalle temporel de la transition ont une chance égale de se produire, on définit une fonction de répartition de probabilité de tir sur cet intervalle. En fonction de la classe de RdP stochastique (et des techniques d'analyses envisagées), différentes fonctions de probabilités peuvent être définies.

Par exemple, dans le cas des RdP stochastiques simples, la fonction utilisée est une distribution exponentielle appliquée à des intervalles temporel des transitions de type $[0,+\infty[$. L'utilisation de ce type de fonction (sans mémoire temporelle[21]) permet l'utilisation des techniques d'analyses Markovienne. On peut alors, dans le cas d'un graphe de marquage cyclique, déterminer les probabilités des états, le temps de séjour dans un marquage, la probabilité de tirer une transition dans un marquage donné. Dans le cas d'un graphe acyclique, on peut évaluer les temps moyens d'absorption…

Afin de permettre une expression de CT quantitatives, différentes classes de RdP stochastiques ont été proposées. Nous présentons plus particulièrement l'une d'elle, celle des RdP temporisés stochastiques [Gallon-1997]. Elle permet de définir, sur des intervalles temporels bornés, différentes fonctions de densité de tir : des fonctions exponentielles, mais aussi des Dirac, des fonctions uniformément continues ou une composition de ces deux dernières. Les auteurs proposent tout un ensemble de techniques d'analyses quantitatives, basées sur des graphes d'états probabilisés (tir au plus tôt, au plus tard, ou à des temps moyens). Un autre avantage de cette classe de RdP est qu'il est toujours possible de se ramener à un RdP t-temporel (en considérant les fonctions de densité de tir des transitions comme uniformes et égales) et d'appliquer des techniques d'analyse comme celle du graphe de classe.

[21] Si un événement produit un franchissement de transition t et transforme le marquage M1 en M2, l'évolution future des transitions sensibilisées en M1 avant le franchissement de t doit être identique à celle qu'elles subiraient si elles venaient juste d'être sensibilisées en M2 [Valette-2000].

### I.3.3.3. Taxonomie des approches UML/RdP

#### I.3.3.3.a) Critères et tableau comparatif

Au cours de cette thèse, l'offre en approches UML et RdP, au départ quasi-inexistante, a considérablement augmenté. Dans ce travail de recensement effectué en janvier 2003, nous en avons dénombré plus d'une dizaine. Bien qu'elles ne ciblent pas toutes la modélisation de STR, certaines travaillent sur des domaines proches (sûreté de fonctionnement, évaluation de performance de systèmes distribués). Elles offrent donc des concepts utilisables dans la modélisation de STR, notamment au niveau des techniques de dérivation ou de vérification des contraintes temporelles.

Par contre, nous n'avons retenu que des approches de dérivation de modèle et donc rejeté des approches par adjonction comme [Machado-2001] [Jørgensen-2002] [Gehrke-1998]. Nous avons aussi écarté les approches pour les workflow [Cardoso-2001] [Dumas-2001] [Eshuis-2001] [Wirtz-2000], ainsi que celles portant sur les interfaces de systèmes interactifs [Elkoutbi-2000].

Nous proposons une synthèse de ces différentes approches (cf. tableau 1) suivant la classe de RdP utilisée, les types de diagrammes UML traités, la présence d'outils de transformation et/ou de règles de transformation, la prise en compte de CT qualitatives (implicite) ou quantitative (explicite) et le domaine étudié.

| Université/projet | Classe de RdP utilisée | Diagrammes UML dérivés | Règles de dérivation | CT | Domaine D'étude |
|---|---|---|---|---|---|
| Université deChicago [Saldhana-2001;Saldhana-2000] | Coloré | Classe, états/transitions | textuelles | implicite | STR/réactif |
| COMET [Pettit-2000;Pettit-2001] | Coloré t-Temporisé | Collaboration, états/transitions | informelles | partielle | STR |
| Université d'Edingburg [Pooley-1998] | Stochastique | Séquence, états/transitions | informelles | explicite | Performance |
| Université de Floride [Dong-2001] [He-2000] | Prédicat Transition Hiérarchique | Classe, Collaboration, états/transitions | Textuelle | implicite | STR |
| HIDE [Huszerl-2001;Huszerl-2002] [Bondavalli-1999] [Majzik-1998] | Stochastique | Cas d'utilisation, classe, séquence états/transitions et déploiement | Nombreuses transformations (description textuelle) suivant l'analyse choisie | explicite | SdF (Sûreté de Fonctionnement) |
| Université de Milan [Baresi-2001a;Baresi-2001b] | RdP de haut niveau | Classe, séquence, états/transitions | MetaEnv grammaires graphiques | implicite | STR/réactif |
| Université de Zaragoza et de Turin [Merseguer-2002;Merseguer-2001] [Bernardi-2002] | Stochastique | Collaboration, séquence, états/transitions | En partie automatisée dans un prototype | explicite | Performance |

**Tableau 1 : Récapitulatif des approches UML/RDP d'intégration par dérivation**

#### I.3.3.3.b) Les classes de RdP utilisées

La plupart des approches utilise un couplage de type objet à RdP à l'exception de [Huszerl-

2002]. Cette dernière utilise un RdP global modélisant tout le système.

Pour les autres, les classes de RdP utilisées varient énormément. Bien que reposant sur des classes de RdP connues (coloré, prédicat/transition ou stochastique), elles proposent souvent des extensions différentes pour la prise en compte de certains aspects OO. Parmi ces extensions, citons : la communication (par des places ou des transitions les modélisant explicitement), la prise en compte (ou non) de l'instanciation des objets, la gestion (ou non) de l'héritage et du polymorphisme. De ce fait, il est difficile de les comparer. Leur point commun réside essentiellement dans le fait que le RdP est utilisé pour modéliser le comportement (cycle de vie) de l'objet ou de sa classe.

Ces approches ne traitent pas de l'héritage, à l'exception de [He-2000]. Cette dernière offre des mécanismes de gestion du polymorphisme et de l'héritage de la partie statique (opérations), mais pas de la partie dynamique.

I.3.3.3.c) Les diagrammes UML dérivés

Du fait des différents types de RdP et du domaine étudié, nous retrouvons différentes stratégies d'utilisation des diagrammes UML. Néanmoins, certaines particularités sont communes.

Toutes les approches proposent une dérivation basée sur les diagrammes d'états/transitions. La sémantique (et parfois même la syntaxe) de ces derniers est systématiquement adaptée. Ces modifications des diagrammes d'états/transitions sont faites pour :

- Les adapter au domaine d'étude (par exemple, par ajout d'informations probabilistes et temporelles sur les transitions),
- Leur donner une sémantique non ambiguë,
- Les simplifier afin de faciliter les procédés de dérivation et différentes simplifications sont proposées (mise à plat de la hiérarchie, pas de garde sur les transitions, pas d'états orthogonaux…)

Au sujet des diagrammes d'interaction, on constate deux grandes utilisations : soit pour décrire (généralement à l'aide d'un diagramme de séquence) un scénario qui doit être vérifié (approche SdF ou évaluation de performance), soit pour modéliser (diagramme de collaboration le plus souvent) toutes les communications et les liens possibles entre les objets.

Concernant le diagramme de classe, seules les informations sur la liste des opérations offertes par une classe sont utilisées. Il n'y a que l'approche de [He-2000] pour modéliser les associations entre classes sur les RdP.

Une seule approche [Bondavalli-1999] propose une dérivation des cas d'utilisation et des diagrammes de déploiement. Cependant, il s'agit d'une utilisation très particulière des cas d'utilisation adaptés pour représenter les défaillances possibles d'un système. Du fait de ces spécificités, la technique ne peut pas être réutilisée dans des approches pour des STR.

Concernant le diagramme de déploiement, il est lui aussi enrichi de considérations liées à la Sûreté de Fonctionnement (SdF) (définition des nœuds redondants, des arbitres et des liens de communication). Il est transformé en un RdP global utilisé pour décrire une vue globale simplifiée de l'architecture physique du système, afin de mener des analyses de probabilités de défaillance matérielle.

I.3.3.3.d) Les règles de transformation d'UML vers les RdP

La plupart des règles de dérivations sont données de manière informelle ou textuelle. Les grands principes sont présentés sur des cas simples et le processus de dérivation se veut généralement automatique. De ce fait, les ambiguïtés d'UML doivent être résolues avant la dérivation. Cela est fait en proposant des extensions UML afin que des informations

supplémentaires soient données et en réduisant la sémantique des diagrammes traités. Cette réduction de la sémantique a déjà été évoquée pour les diagrammes d'états/transitions.

Concernant les diagrammes de séquence, quand ils ne sont pas utilisés pour simplement contrôler la simulation du système (déjà modélisé par un RdP), leur sémantique est alors ramenée à celle des « *Message Sequence Chart* ». Typiquement, par exemple, les relations d'ordre entre les événements sont alors considérées comme définissant un ordre global, alors que, dans la définition d'UML, un ordre partiel est possible.

Une approche [Baresi-2001b] se démarque par ces choix. Ces auteurs ont focalisé leur recherche sur le développement d'un langage et d'un outil de transformation (MetaEnv) des modèles UML. Ils plaident, comme nous, pour un procédé de dérivation semi-automatique et adaptable par l'analyste-concepteur. Leur langage de transformation, basé sur une grammaire graphique (*graph grammar*), permet de spécifier différentes règles de transformation pour un même élément UML, le choix d'une traduction étant laissé à l'analyste-concepteur. Une fois ce paramétrage terminé, le procédé de traduction devient automatique.

I.3.3.3.e) Modélisation des contraintes temporelles et vérifications

Les approches ciblant les STR ou réactifs offrent peu de moyen d'expression (et par conséquent de vérification) des CT explicites. A l'exception de [Pettit-2000] qui permet de manipuler des durées (mais pas des intervalles temporels), les autres approches ne travaillent que sur les CT qualitatives modélisées par la structure du RdP.

Ainsi, la plupart des techniques de vérification se focalisent sur la recherche des bonnes propriétés du RdP. Elles sont basées sur le calcul du graphe de marquage. Ce graphe est obtenu à partir d'un RdP global obtenu après composition des RdP de chaque objet du système.

Concernant la vérification des durées dans l'approche [Pettit-2000], seule une simulation sous l'outil design/CPN est possible. L'approche de [Dong-2001] se base sur l'évaluateur de modèles STEP [Yu-2002] et permet de vérifier des propriétés exprimées avec une logique temporelle linéaire. Cependant, la manipulation de CT quantitatives n'est pas possible.

Il n'y a que dans les approches UML/RdP basées sur l'évaluation de performance de systèmes distribués ou la SdF qu'une meilleure expression des CT quantitatives est possible. Mais ces informations sont d'ordre probabiliste, ciblant plutôt la définition du débit des entrées/sorties du système, de taux de défaillance et de perte d'information… Elles utilisent des techniques basées sur des chaînes de Markov ou sur l'établissement de graphe d'état probabilisé. Dans les deux cas, seuls des temps moyens ou des taux de défaillance sont disponibles. Ces techniques ne permettent pas une couverture exhaustive quantitative du comportement du système [Gallon-1997]. Or, pour des STR stricts ou fermes, il est important de s'assurer, par exemple, qu'un temps de réponse est toujours respecté, garantie que ne peuvent apporter ces techniques d'analyses.

I.3.3.3.f) Bilan

Malgré des travaux similaires au nôtre, couplant UML et les RdP pour le temps réel, nous constatons que notre approche reste pertinente. En effet, aucune ne permet la prise en compte de CT explicites pour leur vérification dans les premières phases du cycle de développement (à l'exception des travaux de l'université de Floride [Dong-2001] basé sur la vérification de CT qualitatives).

Cependant, cette étude des autres approches UML/RdP ouvre des perspectives futures de travaux et de collaboration car nous constatons une grande complémentarité entre toutes ces approches. En effet, des principes semblables se retrouvent, que ce soit pour le couplage des RdP et de la modélisation OO ou pour les techniques de dérivation. Ces similarités permettent d'envisager une certaine compatibilité entre ces approches et donc une réutilisation de certains

travaux. Nous pourrions ainsi, en les adaptant à notre démarche, profiter de travaux plus avancés que les nôtres sur la définition de règles de transformation (en particulier pour le traitement des diagrammes d'états/transitions [Huszerl-2002]) ou sur le développement d'un langage de transformation [Baresi-2001b].

# I.4. Conclusion

Dans ce chapitre, nous avons présenté nos objectifs de recherche : proposer une méthode de spécification des STR stricts ou fermes, focalisée sur la gestion de la complexité et le traitement des contraintes temporelles.

Pour atteindre ces objectifs, nous avons motivé nos choix pour une approche mixte OO, basée sur la notation UML et l'utilisation d'un modèle formel.

En effet, face à la complexité des STR, l'apparition d'une notation standard comme UML nous a semblé être intéressante. Elle permet de disposer d'un langage commun entre les différentes disciplines impliquées dans le développement de STR, langage servant de support pour discuter des problématiques collectives et présenter de manière simplifiée l'état des divers travaux en cours.

Mais UML n'est pas satisfaisante pour représenter les aspects dynamiques d'un STR. En outre, elle ne propose aucun moyen de vérification et de validation de ses représentations. Or, des outils de vérification et de validation sont indispensables pour les STR étudiés. Il faut, par conséquent, coupler UML avec des langages formels, compatibles avec les concepts OO, adaptés à l'expression des aspects dynamiques et à la manipulation de contraintes de temps quantitatives.

Parmi les modèles formels répondant aux critères précédemment donnés, nous avons retenu les RdP. Face à la multiplicité déjà réelle des différents diagrammes UML, un modèle formel asynchrone unique pouvant s'adapter aux différentes représentations et aux différentes problématiques rencontrées dans le temps réel (parallélisme, synchronisation et temps) nous semblait préférable.

L'étude des approches mixtes UML pour le temps réel, basées sur les RdP, a montré qu'aucune n'offrait des moyens de modélisation et de vérification des contraintes temporelles quantitatives. Notre travail trouve donc sa place dans ce contexte.

# Partie II La structuration dans UML/PNO



L'objectif de cette partie II est de présenter les apports de l'approche UML/PNO concernant la gestion de la complexité de la modélisation. Cette problématique est en partie résolue par l'utilisation des principes Orientés Objets (OO) et la notation UML. L'approche OO offre en particulier des mécanismes puissants de structuration (hiérarchie de composition et d'héritage) de la modélisation.

Cependant, ces mécanismes doivent être adaptés au domaine du temps réel. En effet, il est intéressant d'adapter les concepts OO afin de prendre en compte directement le parallélisme. Toutefois, cette représentation intégrée nécessite certains aménagements face à des problèmes comme l'héritage du comportement. Nous présentons ces problèmes, puis nous proposons des adaptations afin de limiter les difficultés liées à l'utilisation de cette représentation intégrée. Ces propositions sont ensuite résumées.

# II.1. Les conflits du paradigme OO intégré

Comme annoncé au § I.2.2, nous avons choisi la représentation intégrée pour coupler les concepts à objets et les mécanismes du temps réel. Avec cette approche, la notion d'objet actif synchronisé réparti apparaît. Cette entité (que l'on nommera dorénavant, dans ce mémoire, objet actif) intègre les trois niveaux de la dynamique qui sont : le comportement interne de l'objet, les synchronisations inter-objets et la répartition sur un système distribué.

Toutefois, cette unification des mécanismes OO avec les mécanismes du parallélisme et de la répartition fait apparaître plusieurs conflits et limitations.

Ces problèmes sont principalement liés :

- A l'héritage du comportement,
- Aux protocoles de communication et mécanismes de factorisation traditionnels appliqués à un système d'objets répartis.

Nous allons détailler chacun de ces conflits, puis nous présenterons les réponses que nous proposons pour les résoudre.

## II.1.1. L'anomalie de l'héritage du comportement

### II.1.1.1. Rappels sur la notion d'héritage dans le cadre de la concurrence

Il faut, avant tout, dissocier la notion d'héritage de type (*sub-typing*) de celle d'héritage de classe (*sub-classing*) [America-1987] [Cook-1990]. Cette dissociation a longtemps fait débat au sein de la communauté scientifique, mais cela a permis de clarifier les concepts de l'héritage et de définir un consensus sur la terminologie.

Nous rappelons ce vocabulaire [Palsberg-1992] : un type peut être vu comme un ensemble de prédicats sur des objets. Dans ce cas, un objet est de type T lorsque cet objet satisfait les prédicats de T. Une classe décrit un ensemble d'objets ayant la même implémentation.

De ce fait, l'héritage de type concerne les aspects sémantiques et impose une conformité des comportements des objets en relation de sous-typage. Il définit une relation de « substitution » ou de « pré-ordre » entre type et sous-type.

Tandis que l'héritage de classe (nommé aussi héritage d'implémentation) est vu comme un mécanisme purement syntaxique destiné à la réutilisation de classes préalablement développées pour permettre une conception plus rapide de nouvelles classes. Il définit la notion d'enrichissement et d'extension des classes.

Par conséquent, le fait que deux classes soient en relation d'héritage n'implique pas forcément (en tout cas du point de vue théorique) que leurs types respectifs soient en relation de sous-typage (et vice-versa).

Ces deux notions d'héritage étant clairement définies, il reste encore certaines lacunes concernant leurs applications dans le domaine du temps réel.

### II.1.1.2. Les limites de l'héritage du comportement

#### II.1.1.2.a) Limites de l'héritage de type

La relation d'héritage de type est basée sur la notion de substitution d'un super-type par son sous-type et dès lors, requiert que ce sous-type se comporte comme le super-type [Wegner-1988].

C'est dans cette notion « pré-ordre » du sous-type que plusieurs relations d'héritage de type

sont définies et utilisées. Cela va de la relation dite « faible » [Gamma-1995] qui se borne à vérifier que les types de données et la signature des méthodes[22] sont compatibles entre le sous-type et son super-type à la relation dite « forte » [Liskov-1994] où le sous-type peut être substitué au super-type sans modifier le comportement global du système. Cette dernière relation a souvent été considérée comme trop restrictive [Wegner-1988] [Liskov-1999]. Diverses relations intermédiaires d'héritage de type pour le comportement (observable ou invocable [Ebert-1994]) d'un objet ont été proposées. On retrouve, dans ces diverses propositions, deux types de relations : soit par raffinement [Paech-1994], soit par extension [Schrefl-1995].

Cette compatibilité du comportement entre sous-type et type a été fortement étudiée ces dernières années. Des réponses ont été apportées pour la recherche de cette équivalence, en particulier lorsque le comportement de l'objet est spécifié par des machines à états [Mc Gregor-1993] ou des RdP [Kappel-1994] [Lakos-1995] [Van der Aalst-1997]. Dans ce cas, le comportement de l'objet est souvent appelé « cycle de vie de l'objet ». Des méthodes basées sur l'équivalence observationnelle des états ou des transitions ont été définies (cf. [Pomello-1992] pour une présentation de ces techniques) :

- Soit pour vérifier que le cycle de vie d'un objet est bien le sous-type d'un autre [Basten-1996] [Schrefl-1995],
- Soit pour définir des règles de bonne construction d'un cycle de vie hérité [Paech-1994].

Récemment, Van der Aalst a résumé ces travaux sur l'héritage du comportement (portant sur les RdP et les algèbres de processus) en les appliquant aux diagrammes UML [Van der Aalst-2002]. Il considère une sémantique des diagrammes UML compatible avec celle des RdP ou des algèbres de processus. Il peut alors vérifier si l'héritage de type entre des diagrammes d'états/transitions ou d'activités est compatible ou non.

Toutefois, nous n'avons pas trouvé de propositions lorsque des contraintes temporelles quantitatives sont portées par le cycle de vie de l'objet. Les travaux existants ne portent actuellement que sur des CT qualitatives et leurs extensions à des CT explicites restent complexes [Aksit-1994].

#### II.1.1.2.b) Limites de l'héritage de classe

Un mécanisme de synchronisation interfère avec l'héritage de classe si des modifications dans la hiérarchie de classes ou l'ajout d'une nouvelle classe exigent la redéfinition d'une ou plusieurs opérations définies dans des classes existantes. Cela impose la redéfinition de certaines méthodes héritées pour assurer l'intégrité des objets concurrents [Matsuoka-1990]. L'anomalie de l'héritage apparaît pour trois raisons [Matsuoka-1993] :

- Quand il y a un nouveau partitionnement de l'état d'acceptation[23] d'un objet.
- Quand il y a une sensibilité à l'historique.
- Lorsqu'il y a modification des états d'acceptation.

D'autres anomalies ont été découvertes plus spécifiques au traitement des contraintes temporelles (voir par exemple [Aksit-1994]).

---

[22] Il s'agit alors de décider quel mécanisme choisir entre la covariance et la contra-variance pour les arguments des méthodes de l'objet.

[23] Un objet est, à chaque instant, caractérisé par la valeur de ses variables d'état (qui sont un sous-ensemble de ses variables d'instance). Il possède donc un ensemble déterminé d'états possibles. Cet ensemble peut être partitionné en plusieurs sous-ensembles, en fonction des méthodes acceptables dans chaque état. Ainsi, chaque sous-ensemble de la partition déterminera un ensemble de méthodes acceptables. Un tel sous-ensemble est appelé état d'acceptation puisqu'il influence l'acceptation des méthodes.

Précisons que toutes ces anomalies sont liées au langage de programmation utilisé, mais aussi, selon certains [Abadi-1996] [Biberstein-1997], à la volonté de faire coïncider la hiérarchie de type avec la hiérarchie de classe.

Néanmoins, dès que le code de synchronisation est séparé du code destiné au traitement des requêtes et que chacune des ces deux parties peut alors être héritée séparément et indépendamment [Mitchell-1996] [Reitzner-1996] [Papathomas-1997], les anomalies sont évitées.

Il reste que ces solutions sont lourdes, complexes et dépendantes d'un langage de programmation bien spécifique. De plus, le développeur doit avoir une connaissance de l'implémentation d'une super-classe (ou tout au moins de ses mécanismes de synchronisation) lors de la définition d'une sous-classe. De ce fait, le principe d'encapsulation n'est plus respecté[24].

## II.1.2. Les limitations du paradigme OO intégré dans un système réparti

Le paradigme OO intégré ne fait pas une grande différence entre un ensemble d'objets sur un système centralisé ou distribué. La métaphore de transmission de messages et de communication client/serveur est simplement étendue à la communication à travers un réseau quelconque [Briot-1996]. Souvent, ce mécanisme d'invocation est considéré comme transparent et est principalement traité en phase d'implémentation. Toutefois, certains mécanismes du paradigme OO doivent être revus. Ceux-ci sont maintenant présentés. Cela exclut une présentation des difficultés ayant trait :

- A la mise en place de mécanismes spécifiques à la sûreté de fonctionnement (comme par exemple la résolution du problème de consensus [Guerraoui-1996] ou des généraux byzantins [Laprie-1995]).
- Aux techniques liées aux systèmes distribués qui se retrouvent dans toutes les approches (gestion de la redondance, mises à jour réparties des données …).
- A la migration des objets sur les nœuds, car cette technique est peu ou pas utilisée dans les systèmes temps réel que nous étudions.
- Aux techniques, provenant des bases de données OO, visant à optimiser et accélérer les accès concurrents à un même objet (protocole sur la concurrence transactionnelle au niveau des objets), d'une part, en raison des limitations qui peuvent en découler [Guerraoui-1995], d'autre part, car cela nous semble être des principes proches de l'ordonnancement de bas niveau et donc hors du cadre de notre travail.

### II.1.2.1. Concernant la transmission de messages

#### II.1.2.1.a) Rappels sur les communications inter-objets

La plupart des communications dans un système OO réparti se font sur le modèle client-serveur. Ces communications sont, en fait, déjà définies dans l'approche intégrée (au deuxième niveau d'intégration de l'objet actif, cf. § I.2.2) comme des synchronisations inter-objets.

Deux grandes politiques de synchronisation entre le demandeur et le fournisseur de service sont généralement distinguées[25] :

- Communication asynchrone : le client poursuit son activité aussitôt le message envoyé. Il n'attend pas la fin de l'exécution de l'opération invoquée, ni même la confirmation par le serveur de la réception de sa demande. Rien ne lui permet de savoir si la demande a bien été reçue par le serveur.

[24] Remarquons que ce constat n'est pas spécifique à l'héritage de classe du comportement. Dès lors que l'on permet la surcharge d'une méthode d'une super-classe, cela nécessite une connaissance du fonctionnement interne de cette super-classe [Mikhajlov-1997].

[25] Il existe ensuite un grand nombre de politiques de synchronisation qui sont différents raffinements de ces deux politiques [Burns-1994].

- Communication synchrone : le client fait sa demande et attend une réponse du serveur. On distingue deux cas :
  - La communication synchrone faible : le client attend simplement la confirmation de la réception de la demande de service (acquittement) par le serveur. Le serveur devra, s'il y a lieu, renvoyer par la suite les résultats de l'exécution de l'opération.
  - La communication synchrone forte : le client attend la réalisation de sa demande de service (et donc les résultats de l'opération s'il y a lieu).

Ces deux types de communications synchrones peuvent être couplés à des gardes temporelles précisant le temps d'attente de l'acquittement ou du service par le client (attente limitée). Si ce temps est dépassé, alors le client n'attend plus et peut entamer d'autres activités.

En considérant que ces mécanismes de synchronisation peuvent être étendus aux communications à travers un réseau, certaines précautions doivent être prises concernant les délais de communication et la communication par diffusion.

#### II.1.2.1.b) Prise en compte des délais de communication

Il faut prendre en compte les délais de communication, en particulier si les temps de communication sur ce réseau ne sont pas bornés. Cela pose la question de la représentation de ces temps, soit directement au niveau des synchronisations inter-objets, soit sur une couche dédiée à cet effet (parfois nommée couche protocole ou méta-protocole ).

Dans le premier cas, une nouvelle répartition des objets sur les nœuds du système impose alors de revoir une grande partie des communications inter-objets. Dans le second cas, il y a risque d'incohérence entre la description des synchronisations au niveau des objets et la couche protocole (par exemple, une synchronisation forte avec garde temporelle échouant toujours du fait de temps de communication dépassant systématiquement le délai de l'attente limitée).

#### II.1.2.1.c) La communication par diffusion

Autant une communication par diffusion reste relativement simple à établir dans un système centralisé, autant de nombreux problèmes se posent dans un système réparti. La plupart de ces problèmes ne sont pas spécifiques à l'approche OO. Se retrouvent, par exemple, la difficulté de garder des relations d'ordre entre les communications venant de nœuds différents [Lamport-1978] et l'augmentation des communications afin de synchroniser les différents nœuds [Dolev-1992].

Par ailleurs, ce type de communication augmente l'indéterminisme du système (ou accroît les difficultés à le diminuer). De plus, il est souvent reproché à la communication par diffusion d'augmenter la difficulté à établir toutes les relations de dépendances entre objets (car elle n'identifie pas explicitement tous les destinataires des messages). De ce fait, la traçabilité s'en trouve diminuée.

### II.1.2.2. Concernant la duplication d'objets

Outre les techniques et difficultés supplémentaires pour la mise à jour des objets dupliqués (non spécifiques à l'approche OO), l'application du protocole client-serveur aux mêmes objets peut poser le problème de la duplication des invocations.

En effet, un même objet peut être à la fois client et serveur. Autrement dit, un objet serveur peut à son tour invoquer d'autres objets (en tant que client). Imaginons qu'un objet (à la fois client et serveur) soit dupliqué : toutes ses copies vont alors invoquer plusieurs fois les mêmes objets. Il y a, par conséquent, duplication des mêmes requêtes, ce qui risque d'amener des incohérences (par exemple, une opération d'incrémentation invoquée plusieurs fois, au lieu d'une).

#### II.1.2.3. Concernant les mécanismes traditionnels de factorisation

Les mécanismes de factorisation OO (classes et héritages) ont souvent été établis sur des hypothèses fortes d'informatique traditionnelle (séquentialité de l'exécution des opérations et mémoire centralisée). Ces hypothèses peuvent atteindre leur limite si elles sont directement transposées dans un système concurrent réparti [Briot-1996].

Ainsi, l'utilisation des variables de classes (et de manière générale de toute variable globale) pose un problème, car il semble difficile de garantir que la mise à jour d'une variable de classes sera immédiatement reflétée sur toutes les instances de la classe lorsque celles-ci sont situées sur plusieurs nœuds du système.

# II.2. Propositions pour limiter les conflits du paradigme OO intégré

## II.2.1. Motivations et choix

#### II.2.1.1. Pas d'héritage pour les objets actifs

Nous pensons que le principe de l'encapsulation est l'un des plus importants de l'approche OO. L'objet ne peut être consulté ou modifié que par l'invocation de ses opérations. Ce principe définit une frontière entre la partie interne de l'objet, protégée de toute modification inopportune et la partie externe, correspondant au comportement attendu de l'objet.

Différents noms ont été donnés à cette partie externe de l'objet : interface, spécification de l'objet, contrat de l'objet… Pour notre part, nous utiliserons le terme de contrat afin de le distinguer de la notion d'interface définie par UML.

Ce contrat de l'objet sert donc d'interface entre la partie interne de l'objet (et qui doit remplir les besoins exprimés par le contrat) et les utilisateurs de l'objet qui peuvent faire confiance à ce contrat sans connaître la partie interne. Notre point de vue est que ce mécanisme est l'essence même de l'approche OO et qu'il doit toujours être respecté.

C'est principalement pour cette dernière raison que nous préférons ne pas utiliser la relation d'héritage de classe. En effet, cette dernière, outre ses anomalies, brise le principe d'encapsulation (cf. § II.1.1.2) en forçant le concepteur, lors du développement d'une sous-classe, à avoir une certaine connaissance de la partie interne de la super-classe. Par ailleurs, cette relation est principalement utilisée dans les phases de conception détaillée ou de codage, phases non abordées dans ce travail de recherche.

Au sujet de l'héritage de type, on peut regretter le peu de solutions concernant la recherche d'équivalences de comportement comportant des contraintes de temps quantitatives. Nous n'utilisons donc pas l'héritage de type pour les objets actifs. Par contre, pour les objets passifs, en particulier les structures de données, il peut être utilisé.

A l'instar de certaines approches [De Hondt-1997] [Weck-1997], nous préférons utiliser des mécanismes alternatifs liés à la composition d'objets, à la notion de contrat et de délégation de services.

#### II.2.1.2. Une sémantique restreinte de la communication entre objets

Nous avons choisi, dans ce premier temps de définition d'UML/PNO, de restreindre les protocoles de communication. Nous n'autorisons donc que les communications de type client/serveur (par échange de messages) dans lesquelles les destinataires sont explicitement identifiés. Elles seront toujours décomposées en communications « une à une ».

Par conséquent, nous n'autorisons pas les communications par diffusion (*broadcast*) ou par

levée d'exception. Celles-ci devront donc être traduites en communications de type client-serveur.

Les temps de communication ne seront pas ajoutés aux descriptions des synchronisations des objets, mais portés sur un objet spécial : l'objet « médium ».

#### II.2.1.3. Une duplication explicite des objets

Pour la classe de systèmes temps réel que nous étudions, nous pensons que le besoin parfois problématique de duplication d'objets sur plusieurs nœuds reste secondaire[26]. Lorsque le cas se produit malgré tout, nous préférons alors considérer ces objets dupliqués comme des objets différents mais respectant un même contrat. L'intégrité et la mise à jour des données entre ces objets sont alors explicitement représentées.

Les limitations des mécanismes traditionnels de factorisation, posées par l'utilisation de variables de classes ou de variables globales, peuvent être évitées par une modélisation explicite à l'aide d'objets dédiés. Par conséquent, nous ne permettons pas l'utilisation de variables de classe ou de variables globales dans UML/PNO.

#### II.2.1.4. Vers une approche orientée composant

Du fait de ces choix, UML/PNO est basée sur une hiérarchie de composition dans laquelle les relations d'héritage (pour les objets actifs) ne sont pas utilisées. A leurs places, nous utilisons la relation de composition et avons étendu la notion de contrat (ou interface) d'un objet. Cette vision est en fait similaire à l'approche dite « orientée composant » (*component approach*) [Meyer-1999] [Szyperski-1998].

Afin de bien différencier ces objets et de montrer leur lien avec l'approche orientée composant, nous les appelons : objets composés ou CO (*Compound*[27] *Object*) [Delatour-1998]. Nous conservons la notion de classe et d'objets définis comme des instances de classe. Nous avons donc la notion de classe d'objets composés ou CC (*Compound* Class). Détaillons ces notions de CC et de CO et montrons comment elles nous permettent d'atteindre les objectifs décrits précédemment.

### II.2.2. Les imprécisions du stéréotype UML <<subsystem>>

D'une façon générale, dans UML/PNO, un CO constitue l'unité principale de structuration du système et en modélise les parties et/ou les sous-parties. Souvent, il correspond à une partie physique du système réel que l'on modélise. Le CO contient toute l'information nécessaire à la spécification, à la conception et au développement de cette partie du système.

Pour représenter les CO et les CC avec UML, nous nous sommes appuyés sur la notion de paquetage UML stéréotypé <<subsystem>>. Nous présentons ce stéréotype et montrons comment nous avons dû l'aménager pour répondre à nos besoins, en créant les stéréotypes « CO » et « CC ».

#### II.2.2.1. Rappels sur la notion de « subsystem » UML

La notion de <<subsystem>> UML est très proche de la notion de CC, c'est pour cette raison que nous l'avons reprise. Un <<subsystem>> est une entité instanciable (c'est un paquetage et un classifieur[28]), destinée à représenter les parties et sous-parties d'un système.

Dans UML 1.3, un <<subsystem>> (que nous nommons aussi sous-système) comporte deux parties (cf. Figure II-1) :

- Une partie spécification qui permet de rassembler tous les diagrammes UML expliquant le

[26] Rappelons que nous ne traitons pas des difficultés liées à la sûreté de fonctionnement.

[27] Le mot « compound » a été préféré a celui de « component » car UML définit déjà une entité « component » dont la signification diffère de celle que nous donnons à « compound ».

[28] Et dès lors, peut être utilisé comme une classe dans les diagrammes UML.

rôle du sous-système (une abstraction du comportement du sous-système). Une liste des opérations (services) publiques offertes peut y être jointe.

- Une partie réalisation qui contient tous les diagrammes UML expliquant comment les spécifications sont réalisées.

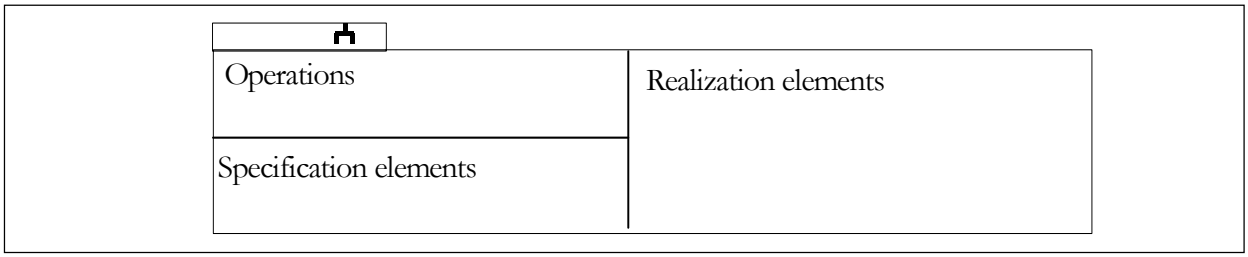


**Figure II-1 : Représentation d'un <<subsystem>> UML**

Cette représentation est intéressante car l'implémentation d'un sous-système est découplée de la façon dont il est perçu par son environnement. De plus, il est possible de représenter les liens entre la partie spécification et réalisation et d'indiquer les éléments réalisant telle partie de la spécification (lien « realize », [OMG-1999b] pp2-174).

UML permet deux types d'utilisations des sous-systèmes :

- Comme sous-système ouvert, dont les composants sont directement accessibles par l'environnement.
- Comme sous-système encapsulé, c'est alors une boîte noire dont les éléments internes ne sont accessibles que par les opérations publiques du sous-système.

La première utilisation est permise grâce à une relation d'importation (« import ») ou d'héritage entre deux sous-systèmes et grâce à des règles de visibilité des éléments du sous-système (élément public, protégé ou privé). Dans ce cas, le sous-système important l'autre sous-système (ou le spécialisant) a accès à l'espace de nommage de ce dernier. Dès lors, il peut directement référencer les éléments publics (et les éléments protégés si c'est une relation d'héritage) contenus dans le sous-système.

Dans le cas des sous-systèmes encapsulés, il est possible d'utiliser la dépendance <<use>> ([OMG-1999b] pp3-84). Seules les opérations publiques du sous-système (modélisées par l'interface du sous-système) sont alors accessibles.

C'est cette seconde utilisation qui nous intéresse pour la traduction en UML de la notion de CC, car le principe d'encapsulation est respecté. Toutefois, plusieurs points dans la définition UML des sous-systèmes encapsulés restent non-définis et limitatifs concernant la modélisation d'entités temps réel.

### II.2.2.2. Imprécisions de la définition des sous-systèmes encapsulés

#### II.2.2.2.a) Le mécanisme de transmission des opérations

La norme UML précise que les éléments internes d'un sous-système encapsulé ne sont accessibles que par invocation des opérations offertes par celui-ci. Elle précise aussi qu'un sous-système n'a pas de comportement propre. Ce dernier est en fait délégué aux entités qui le composent. Ce n'est donc qu'une « enveloppe » chargée de transmettre les demandes de services aux entités qu'elle contient ([OMG-1999b] pp2-180 et 2-181). Cependant, ce mécanisme de transmission des invocations n'est pas décrit. Différentes politiques de transmission des invocations peuvent être envisagées, nous en évoquons quelques-unes :

- Toutes les invocations des opérations sont dirigées sur une entité interne spécifique du CO, qui se charge alors de la redirection vers les éléments internes.
- Suivant le nom des invocations, elles sont envoyées vers des instances prédéfinies.
- Les valeurs des arguments de l'invocation sont utilisées afin d'identifier directement l'entité interne. Remarquons qu'un argument ne peut pointer directement sur une instance du sous-système puisque l'espace de nommage de cette instance n'est pas connu à l'extérieur du sous-système.

Dans tous les cas, il faut prévoir un moyen de modifier les paramètres de l'invocation. L'exemple donné en Figure II-2 détaille ce dernier point. Une invocation d'opération m1 est effectuée par un objet A du sous-système S1[29] au sous-système S2. Selon UML, cette invocation est traduite par l'envoi d'un stimulus où les entités envoyant et recevant l'invocation doivent être identifiées et nommées.

Lorsque A effectue son invocation, quelle entité « envoyeur » déclare-t-il : S1 ou l'objet A ? Autrement dit, cela se traduit-il par un stimulus S(S1,S2,m1)[30] ou S(A,S2,m1) ?

Supposons que le stimulus envoyé soit du type S(S1,S2,m1). En effet, le sous-système S2 n'est pas sensé partager l'espace de nommage de S1 et ne connaît donc pas l'existence de A.

Que se passe-t-il ensuite, au moment du traitement par S2 du stimulus ? Est-il modifié de sorte que le receveur soit maintenant l'entité interne de S2 (dans notre cas, l'objet B) ? Ou bien un nouveau message est-il créé avec les bonnes valeurs ?

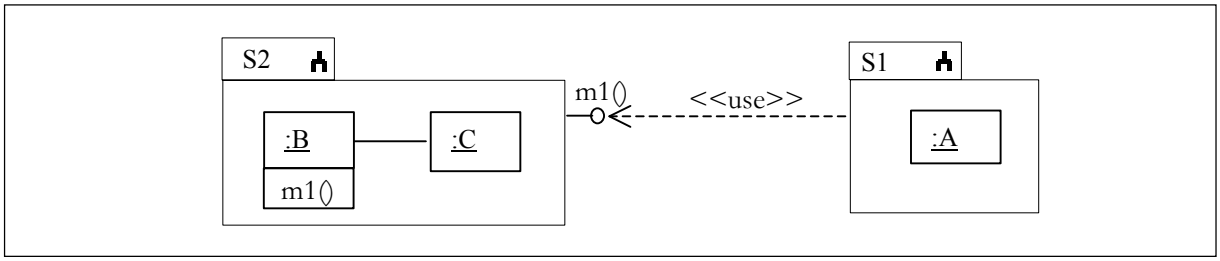


**Figure II-2 : Invocation d'opération entre deux « subsystems »**

A toutes ces questions, concernant la modification des invocations (aussi bien à l'envoi qu'à la réception) ainsi qu'au choix du mécanisme de transmission (ou redirection), UML 1.3 n'apporte pas de réponse. Il laisse l'analyste-concepteur libre de ses propres interprétations.

II.2.2.2.b) Les relations entre sous-systèmes et visibilité des composants

L'exemple de la Figure II-2 incite aussi à mieux comprendre comment se passent les relations de visibilité des composants des sous-systèmes. Puisque S1 entretient une relation de dépendance avec S2, cela implique que tous ses éléments ont accès aux opérations publiques de S2. Il n'y a aucun moyen de limiter cette visibilité.

Mais qu'en est-il des éléments de S2 ? Ont-ils connaissance de S1 ? Cela va dépendre de la relation entre les deux sous-systèmes. Si cette relation est une association bidirectionnelle, la réponse est positive : les objets de S2 peuvent faire référence à ceux de S1. Si c'est une dépendance <<use>>, UML ne répond pas explicitement à cette question ([OMG-1999b] pp3-49 et 3-84) et il semble difficile de statuer. En effet, pour une communication asynchrone, il n'y a pas d'ambiguïté possible et la relation <<use>> peut être considérée comme uni-directionnelle. En revanche, pour un appel synchrone faible, se pose le problème du retour de service de l'invocation. Si ce retour est implicite, la relation <<use>> est alors bi-directionnelle. Les objets de S2 peuvent faire référence à S1. Si ce retour est explicite, il faut alors l'indiquer par une autre relation <<use>> allant de S2 à S1.

[29] Rappelons qu'un sous-système n'a pas de comportement interne.

[30] En considérant qu'un stimulus peut être représenté par la notation S(envoyeur, receveur, opération).

Ce problème de visibilité montre qu'il est difficile de représenter les relations de dépendances qu'entretiennent les éléments d'un sous-système avec l'environnement de ce sous-système et qu'il est impossible de différencier la visibilité des éléments sur l'environnement.

#### II.2.2.2.c) L'instanciation d'un sous-système

Selon UML 1.3, un sous-système est une entité instanciable. Lorsqu'il est instancié, il doit alors être considéré comme un objet composite. Mais aucune explication ni aucun exemple n'est fourni sur la façon de créer une instance de sous-système : un sous-système ne définit pas de constructeur. Par conséquent, l'instanciation d'un sous-système ne permet pas non plus de le paramétrer, ni de définir ses relations de dépendances avec son environnement. Nous sommes alors directement confrontés au problème évoqué au § II.1.2.2.

### II.2.2.3. Limites des sous-systèmes encapsulés pour le temps réel

Les informations proposées par un sous-système encapsulé pour son utilisation consistent simplement en une liste d'opérations. Certes, la partie spécification du sous-système doit permettre de décrire plus en détail l'utilisation de ces opérations par des diagrammes de séquence ou de collaboration. Néanmoins, cette description n'est pas utilisée lors de l'usage d'un sous-système par invocation de ses services. Cela nous semble insuffisant dans le domaine de la modélisation temps réel.

En effet, il a été montré que l'interface d'un objet (réduite à une liste des opérations offertes) n'est pas suffisante pour en comprendre son utilisation [Meyer-1988]. Des pré et post conditions sur les opérations ainsi que des invariants sur le comportement global doivent être définis[31]. L'interface d'un objet et cet ensemble de prédicats définissent, dans la terminologie employée par Meyer, le contrat d'un objet. Il permet de préciser le comportement attendu d'un objet. Avec UML 1.3, ces prédicats peuvent être exprimés en OCL. Des mots-clefs ont d'ailleurs été réservés à cet effet [OMG-1999b](pp 7-4) et des extensions ont été définies (voir par exemple la méthode Catalysis/UML [D'Souza-1998]).

Meyer a précisé que cette notion de contrat d'un objet peut être reprise pour définir le comportement externe d'un composant, mais il conseille de l'enrichir par la liste des services que peut invoquer le composant [Meyer-2000]. Cela permet ainsi de décrire toutes les relations de dépendance du composant.

En outre, comme cela se pratique dans la méthode HOOD [HRM-1989], nous pensons que cette notion de contrat doit aussi être complétée par des explications sur les mécanismes de synchronisations inter-objets. Par exemple, afin d'utiliser un service, il faut savoir si l'appel à celui-ci est bloquant (ie. synchrone) ou non bloquant (asynchrone). Il faut aussi pouvoir décrire les effets observables d'une invocation d'opération, en particulier si cette invocation déclenche une requête vers un autre sous-système.

Enfin, afin de s'adapter aux spécificités du temps réel, nous pensons qu'il est nécessaire d'enrichir les indications sur les mécanismes de synchronisation par des informations temporelles. Lorsqu'un service est demandé à un sous-système, au bout de combien de temps ce service est-il rendu ? Toutefois, ce type d'information, par essence dynamique, figurera sur les diagrammes de comportement présentés par la suite.

Cette présentation du stéréotype <<subsystem>> UML, ainsi que de ses imprécisions et

[31] Notre recherche se focalisant sur la partie dynamique, nous n'avons pas travaillé sur la formalisation de ces aspects statiques. Aspects qui portent, par exemple, sur la vérification des types de données, la recherche de la compatibilité des opérations, la vérification des pré et post conditions des contrats des CO ou de leurs invariants.

limites, nous a permis de préciser nos objectifs concernant la définition des CC et CO.

## II.2.3. Le stéréotype <<CO>>

Pour des raisons didactiques, nous allons d'abord présenté la notion de CO puis celle de CC. Il faut considérer un CO comme une entité instanciant un CC donné. Un CO est constitué de deux parties (cf. Figure II-3) :

- Une partie publique, nommée vue externe du CO, qui représente en quelque sorte le guide d'utilisation du CO.
- Une partie privée, nommée vue interne du CO, qui décrit le fonctionnement interne du CO. Cette partie peut elle-même être constituée de CO (alors nommés sous-CO).

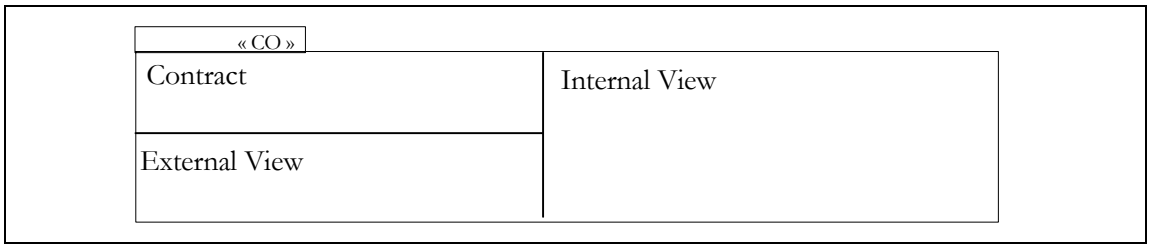


**Figure II-3 : Représentation d'un « CO » UML/PNO**

### II.2.3.1. La vue externe d'un CO

Comme dans la partie spécification d'un sous-système, un grand nombre de diagrammes UML peut être utilisé pour décrire le comportement attendu du CO. Dans UML/PNO, ce sont des diagrammes de séquence, de collaboration et de cas d'utilisation. Ils décrivent le fonctionnement du CO par rapport à son environnement. Ces diagrammes sont nommés "diagrammes externes du CO".

Par ailleurs, cette description doit être enrichie d'une entité plus complète qu'une simple liste d'opérations (cf. § II.2.2.3). Nous avons nommé cette entité <<contract>>[32]. C'est une instance d'une extension au stéréotype « interface » d'UML. Comme un objet <<interface>>, un <<contract>> ne fait que déléguer les services qu'il offre aux sous-CO chargés de réaliser le service. C'est cet objet qui effectue les modifications nécessaires sur les stimuli et messages lors de la transmission des invocations (cf. § II.2.2.2.a).

Un objet <<contract>> est constitué :

- D'un constructeur et d'une fonction d'initialisation des liens entre CO (*InitCO*). Leurs rôles seront détaillés lorsque nous présenterons le stéréotype CC et la relation d'utilisation des CO.
- D'un compartiment rassemblant les invariants du CO, décrits en OCL.
- D'un compartiment, dit « *CO_interfaceIN* », donnant la liste des opérations offertes par le CO. Pour chaque opération, des pré et post conditions peuvent être décrites en OCL.
- D'un compartiment, nommé « *CO_interfaceOUT* », donnant la liste des opérations invoquées par le CO
- D'un compartiment, nommé «*CO_dependency*», donnant la liste des entités extérieures au CO dont les opérations sont invoquées par celui-ci.

Lorsque le CO est un objet actif, à son contrat est ajouté un diagramme à réseau de Petri

[32] En référence à Meyer.

(RdP), nommé *external Behavioural Petri Net* (eBPN). Ce diagramme à RdP de comportement externe sera présenté en partie III. Son rôle est de modéliser les politiques de communication des opérations offertes et requises, ainsi que de donner des indications sur le comportement observable du CO.

### II.2.3.2. La vue interne d'un CO

La vue interne du CO décrit la décomposition du CO : le CO n'est plus considéré comme une boîte noire. Cette vue comprend la liste des sous-CO qui le constituent (nommée *List_sub_CO*) et les diagrammes de séquence, de cas d'utilisation, de collaboration et d'activités. Ces diagrammes UML, dits "diagrammes internes du CO", décrivent le fonctionnement interne du CO. Ils précisent les collaborations entre les sous-CO et les fonctionnalités internes du CO.

Un diagramme à RdP, nommé *internal Behavioural Petri Net* (iBPN), complète la description de la vue interne lorsque ce CO est un objet actif. Son rôle est détaillé en partie III.

Lorsqu'un CO est composé d'autres CO, nous parlons de CO composite. Si un CO est non décomposable, c'est un CO terminal. Dans ce dernier cas, sa partie interne est un objet au sens UML.

### II.2.3.3. Illustration sur l'exemple du simulateur de vol

Pour concrétiser la notion de CO, nous allons présenter plus en détail l'exemple qui nous servira de cas d'étude tout au long de ce mémoire. Il s'agit du simulateur de vol SIMONA (cf. introduction générale), nommé *Simona Research Simulator* (*SRS*).

#### II.2.3.3.a) Présentation du SRS

Ce simulateur expérimental se présente sous la forme d'une cabine d'avion en carbone ultra léger et en fibres d'aramide montée sur une plate-forme hydraulique à six degrés de liberté. Afin de parvenir à des simulations de haute qualité, le pilote doit être incapable de percevoir des délais entre l'occurrence des signaux d'entrée qu'il provoque et la réponse de la cabine. De telles contraintes temporelles demandent une vitesse de rafraîchissement élevée des signaux de commande de la plate-forme. Une précision relativement bonne dans l'élaboration des différentes temporisations est de même nécessaire à la simulation. L'infrastructure du système informatique temps réel de commande du simulateur est constituée de plusieurs calculateurs connectés à un réseau local haute vitesse.

#### II.2.3.3.b) Architecture du premier niveau du SRS

La décomposition de ce simulateur, représentée par un diagramme de collaboration en Figure II-4, reste très proche de la distribution physique imposée par l'environnement.

Les CO S_*Cockpit, S_Motion* et *S_Visual* concernent la gestion des périphériques matériels. Le premier encapsule toute la gestion des périphériques matériels contenus dans la cabine de pilotage. Le second s'occupe du déplacement de la plate-forme hydraulique du simulateur. Le troisième gère l'affichage des images sur l'écran parabolique placé devant le cockpit afin de simuler le monde extérieur que devrait voir le pilote.

Le CO S_*Vehicle* contient tous les modèles mathématiques décrivant les caractéristiques de l'avion. Le CO S_*Environnement* contient tous les modèles relatifs à l'environnement de l'avion (conditions météorologiques, description géographique...).

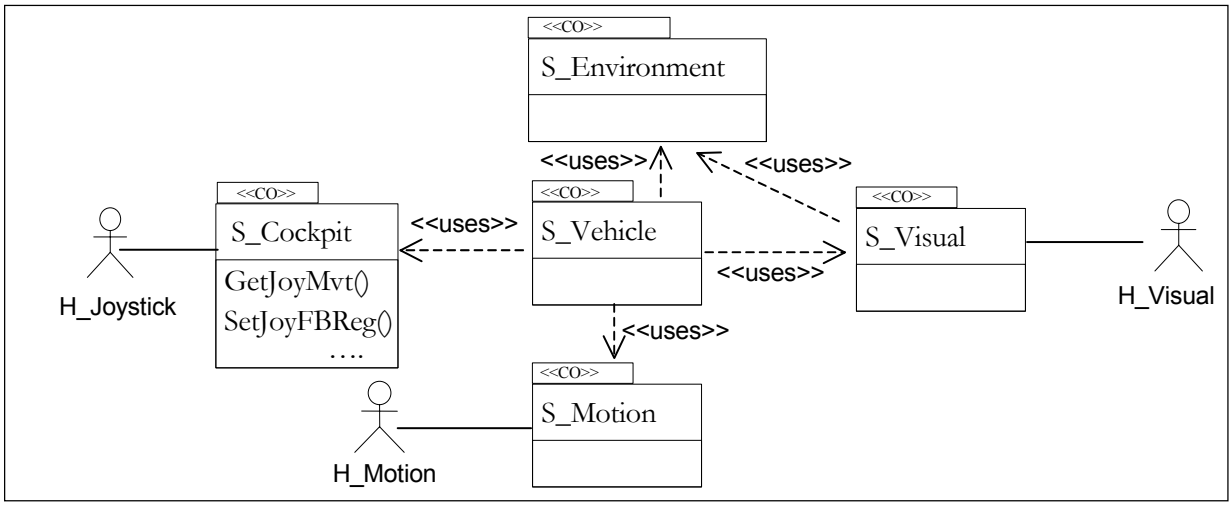


**Figure II-4 : Décomposition partielle du 1er niveau du SRS**

Des relations de dépendances, stéréotypées « uses », entre CO dénotent les appels de méthodes. Ce type de relation est détaillé en § II.2.5.1.

Concernant les acteurs (*H_Joystick*, *H_Motion* et *H_Visual*) modélisés sur le diagramme de collaboration, il s'agit des périphériques physiques réels que doit contrôler le simulateur. Notons que, dans la modélisation temps réel, nous nous écartons de la représentation classique UML qui veut qu'un acteur représente un rôle spécifique et non les périphériques matériels utilisés par cet acteur. Or, dans le contexte temps réel, il y a un besoin d'identifier rapidement les stimuli et les réponses du système (les CT sont généralement exprimées sur ces entrées et sorties) et par conséquent les périphériques matériels qui y sont liés. La notion d'acteur en tant que rôle peut toujours être présente. Toutefois, dans ce cas, elle est utilisée dans les cas d'utilisation et non dans un diagramme de collaboration. La différenciation entre rôle et périphérique se fait par le préfixe H_ (pour Hardware) dans le nom de l'acteur. Ces deux entités restent des acteurs car elles font partie de l'environnement du système.

Dans cette partie II, nous limiterons notre étude au CO *S_Joystick*, placé à l'intérieur du CO *S_Cockpit*. Ce CO est chargé de contrôler le manche à balai de la cabine de pilotage. Il s'agit là d'un joystick à retour de forces, dont les mouvements doivent être régulièrement capturés et auquel il faut appliquer des commandes (vibration, rigidification des mouvements possibles du manche du joystick…).

Au niveau du CO *S_Cockpit*, nous avons représenté deux opérations invocables[33] (*GetJoyMvt* et *SetJoyFBReg*) par le CO *S_Vehicle*. Ces opérations sont en fait traitées par le CO *S_Joystick*. Elles permettent d'obtenir le mouvement du joystick physique et de paramétrer la consigne de régulation de son retour de force. Figure II-5, le mécanisme de transmission de ces deux opérations est représenté. Cette redirection est modélisée par une dépendance stéréotypée <<in>>. Cette relation est détaillée en § II.2.5.2.

Le CO *S_Cockpit* est à considérer comme composite, bien que nous n'ayons dessiné ici qu'un seul sous-CO le composant. Une note donne la liste des sous-CO (initialisée par le constructeur).

[33] Naturellement, ce CO offre de nombreuses autres opérations, mais dans un souci de clarté, nous ne représentons que celles-ci.

A travers cet exemple, les grands principes des CO ainsi que leurs représentations en UML ont été donnés. Présentons maintenant l'instanciation d'un CO à partir d'une classe composée (CC).

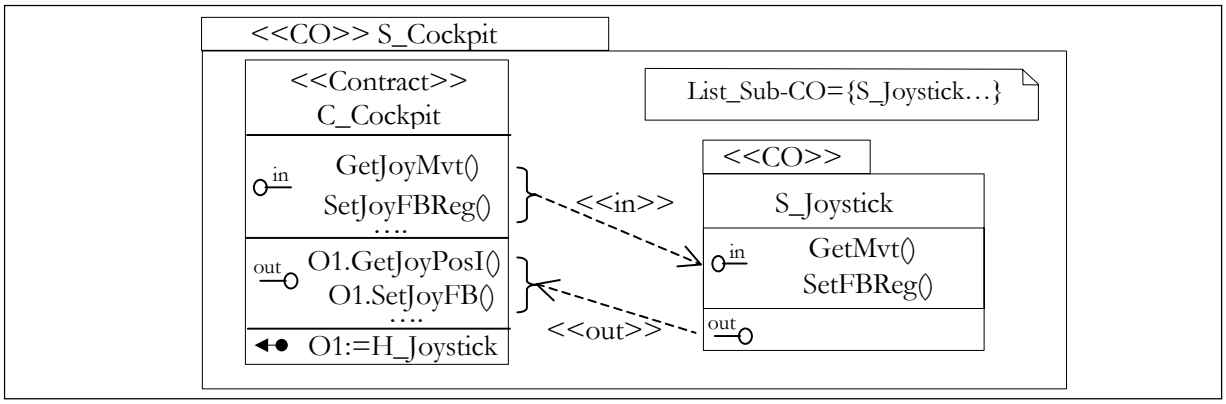


**Figure II-5 : Vue partielle du CO S_Cockpit**

## II.2.4. Le stéréotype <<CC>>

La différence essentielle entre un CO et un CC est le fait que le CC ne sait pas, par définition, avec quelles autres instances il communiquera. Ce n'est que lors de l'instanciation du CC que nous pourrons préciser avec quelles autres entités cette instance devra communiquer. Naturellement, un CC dispose d'un constructeur afin de permettre le paramétrage de ses instances. L'ajout des informations sur les communications se fait grâce à la fonction d'initialisation *InitCO* de l'objet <<contract>> du CC (cf. § II.2.3.1). Nous nous plaçons donc dans un contexte de définitions de classes dépourvues d'appels nominatifs.

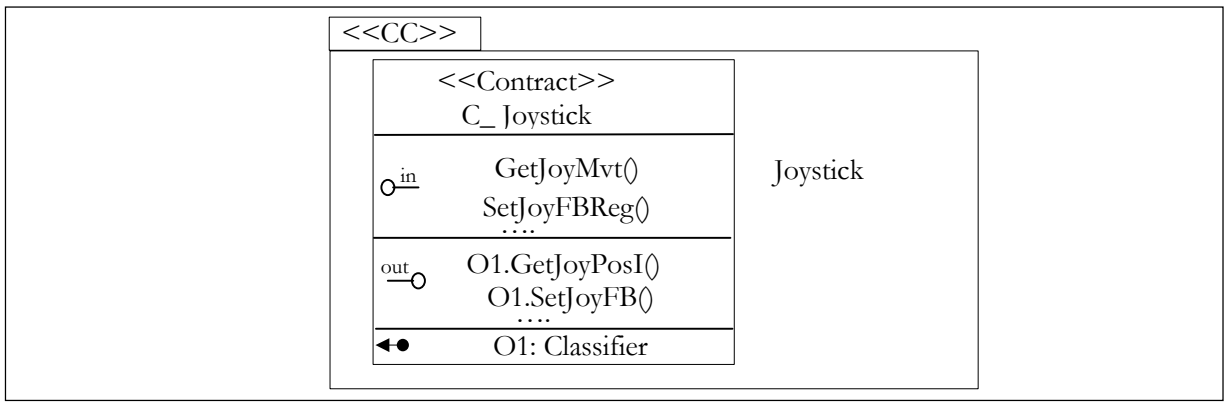


**Figure II-6 : Vue du CC Joystick**

C'est pour cette raison qu'un objet <<contract>> distingue l'interface OUT et le compartiment de dépendance. L'interface OUT permet d'identifier les opérations (dont leurs signatures) invoquées par le CC. Le compartiment de dépendance d'un CC n'est pas complet, il donne seulement une indication sur le nombre d'entités appelées par le CC (cf. Figure II-6).

Ce n'est que lors de l'établissement des liens d'utilisation entre une instance de CC et d'autres CO que le compartiment de dépendance est complété. Détaillons cette relation d'utilisation.

## II.2.5. Les relations entre CO et sous-CO

### II.2.5.1. La relation d'utilisation entre CO

L'appel de services entre CO (communication) est régi par un certain nombre de règles dites

d'utilisation des services. Cette relation d'utilisation est représentée par le lien stéréotypé <<uses>>. C'est une extension de la dépendance UML <<use>> qui a globalement le même rôle mais qui, en plus, respecte les règles d'utilisation de services entre CO et va compléter le compartiment de dépendance du contrat associé au CO.

Ces règles, dite d'utilisation <<uses>> des services entre CO, sont :
Un CO x peut demander un service S à un CO y ssi :

- Les CO sont "frères", c'est-à-dire CO x et CO y sont de même niveau,
- L'interface OUT du CO x est compatible avec l'interface IN du CO y. Cette compatibilité portant uniquement sur une équivalence du nom du service S et de sa signature dans les deux interfaces.

Par conséquent, si un CO doit utiliser les services d'un sous-CO contenu dans un CO frère, ce service doit être présent dans le contrat de ce dernier.

Ces règles garantissent la production de CO réutilisables, développables de manière indépendante et pouvant toujours s'interfacer. Les CO sont des boîtes noires. Si nous autorisions des accès directs entre contenus de CO, nous ne pourrions plus garantir l'indépendance spatiale de ces CO et le principe d'encapsulation ne serait plus respecté.

Une relation <<uses>> peut représenter plusieurs invocations d'opérations. Dans le cas d'une communication synchrone forte, le retour est implicite. Pour une communication synchrone faible, l'acquittement est implicite, mais le retour des résultats de l'appel doit être explicitement représenté par une relation <<uses>>.

Nous avons précisé que cette dépendance <<uses>> mettait à jour le compartiment de dépendance du CO. Cette mise à jour se fait en utilisant la fonction *InitCO*(*NuméroClassifieurOut, instance*), en précisant donc le numéro de l'objet invocable et en donnant l'instance qui lui correspond. Cette fonction vérifie alors la compatibilité des interfaces.

Un dernier mécanisme est mis en place par la dépendance <<uses>> : il s'agit de la modification des paramètres « émetteur » et « envoyeur » des messages (ou stimuli) UML représentant les invocations (cf. II.2.2.2.a). Afin de bien comprendre son principe, la relation de transmission qui est en relation avec ce mécanisme est maintenant détaillée.

### II.2.5.2. La transmission de services dans un CO

Le mécanisme de transmission[34] de services indique quel sous-CO est chargé de réaliser les services offerts par un CO. Si un CO composite offre un service, cela implique que l'objet <<contract>> de ce CO contient une référence sur le sous-CO chargé de réaliser cette opération. Un lien <<in>> (cf. par exemple Figure II-5) modélise cette relation de transmission. Ce lien paramètre le mécanisme de modification des attributs des messages ou stimuli (émetteur, récepteur et nom de l'opération).
Nous avons défini une autre relation, symétrique à <<in>>, nommée <<out>>, qui indique les opérations externes au CO et requises par les sous-CO (cf. par exemple Figure II-5).

Ces deux relations permettent de séparer les composants internes du CO des communications avec l'extérieur du CO. Autrement dit, les sous-CO n'ont aucune connaissance de l'environnement du CO. C'est le contrat du CO qui est chargé de faire le lien. Ainsi, tant que le

[34] Nous avons préféré utiliser le terme de transmission (« forwarding ») plutôt que celui de délégation, car ce dernier est déjà utilisé dans la communauté scientifique avec un sens légèrement différent [Stein-1987].

contrat d'un CO est respecté par la vue interne, cette vue interne peut être modifiée sans remettre en cause les collaborations de ce CO avec d'autres CO frères.

Naturellement, ces deux relations <<in>> et <<out>> vérifient la compatibilité des opérations (signatures et pré et post conditions doivent être compatibles).

### II.2.5.3. Exemple de collaboration entre CO

Figure II-7, nous donnons un exemple d'invocations d'opérations entre CO. Le sous-CO A du CO X effectue une invocation d'opération m1 au CO Y. En retour, le CO Y invoque l'opération m2 du CO X. Décrivons le cheminement et les transformations du stimulus, noté S(émetteur, récepteur, opération) correspondant à l'invocation de m1.

Le sous-CO A effectue la demande au contrat du CO X. le stimulus S(A, C_X, O1.m1) est donc émis. C'est l'objet contrat C_X qui va ensuite rediriger cette invocation et transformer le stimulus en S(X ,Y , m1). Concernant l'invocation de l'opération m2 du CO X par le CO Y, le stimulus arrivant sur le CO X est de la forme S(Y,X,m2). C'est le contrat C_X qui le traite et le transforme en S(C_X,A,mA). Notons qu'il est possible de renommer les opérations (cf. opération m1). C'est l'un des rôles du lien <<in>>.

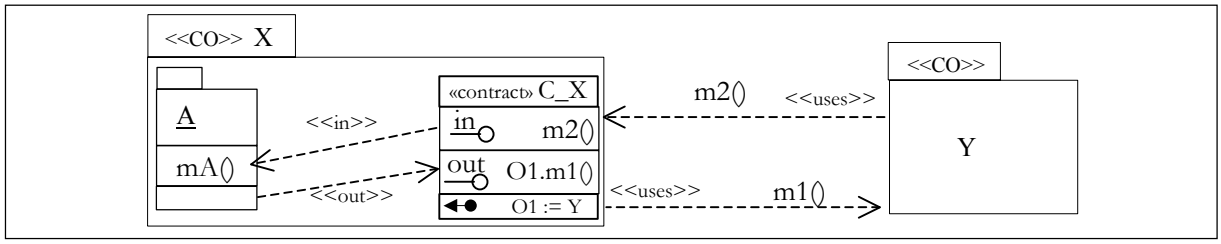


**Figure II-7 : Invocation d'opérations entre deux CO**

# II.3. Résumé sur la structuration dans UML/PNO

UML/PNO est basée sur une structuration Orientée Objet dans laquelle la hiérarchie de composition et la notion de contrat d'un objet sont utilisées. Ces deux notions représentent une alternative à la relation d'héritage (de type et de classe). Afin de différencier cette notion de celle d'« objet », nous appelons ces entités dans UML/PNO des objets composés (Compound Object ou CO). Un CO est une instance d'un CC (Compound Class), un stéréotype basé sur le stéréotype UML <<subsystem>>). Un CC a été défini afin de lever les ambiguïtés et les vides sémantiques de la notion de sous-système UML.

De ce fait, un CO est composé de deux vues :

- Une vue externe, principalement symbolisée par le contrat du CO, qui représente en quelque sorte son guide d'utilisation. Lorsque le CO est actif, son comportement externe est formalisé par un diagramme à RdP de comportement externe (eBPN).
- Une vue interne qui modélise l'intérieur d'un CO, sa composition en sous-CO, sa réalisation… Dans le cas où ce CO est actif et terminal (non décomposé en sous-CO), son comportement interne est modélisé par un diagramme à RdP de comportement interne (iBPN).

Différents diagrammes UML (cas d'utilisation, de séquence, de collaboration et d'activités) peuvent être utilisés afin de documenter ces deux parties du CO. Les relations d'utilisation entre CO et les transmissions d'invocation ont été définies afin de séparer les deux vues et de les rendre indépendantes. De plus, les possibilités de communications et de visibilité des sous-CO ont été restreintes. L'utilisation d'un CO se fait toujours par invocation des opérations qu'il offre. Grâce à ces restrictions, le principe d'encapsulation est toujours respecté : des CO peuvent utiliser un CO dont seule la partie externe est définie ; une vue interne peut implémenter une ou plusieurs vues externes et vice versa. Cela permet, par exemple, à différentes équipes de travailler

indépendamment, certaines modélisant la vue externe, d'autres développant la vue interne.

Notre principal objectif, qui était de disposer d'un mécanisme puissant de structuration, reposant sur les principes OO et compatible avec le domaine du temps réel, est atteint. Très tôt dans le cycle de développement, l'analyste-concepteur dispose de vues (dont il peut ajuster le détail) des parties du système en cours de construction. Chaque partie (CO et sous-CO) est indépendante et deux niveaux de documentation, à l'aide des diagrammes UML, sont possibles (vue interne ou externe). Cette structuration apporte une réponse supplémentaire à la gestion de la complexité des informations de modélisation.

Les diagrammes à RdP, utilisés pour formaliser le comportement des CO actifs, vont maintenant être présentés en partie III.

# Partie III Formalisation du comportement



Ce chapitre présente nos propositions concernant la formalisation des aspects dynamiques du système. Cette formalisation est apportée par l'utilisation des RdP. Pour ce faire, nous avons défini un nouveau type de diagramme, que nous avons appelé diagramme à RdP. Il existe différentes utilisations de ces derniers, mais tous sont basés sur le même formalisme. Dans cette partie III du mémoire, nous présentons ce formalisme et son utilisation pour la modélisation des RdP de comportement (*Behavioural Petri Net*, BPN). Ils modélisent de manière formelle le comportement des CC et les communications entre CC.

Nous distinguons pour chaque CC :

- Le RdP de comportement externe (*external* BPN), qui ne s'intéresse qu'aux relations inter-objets et décrit une partie du comportement attendu du CC (son contrat), sans détailler son comportement interne.
- Le RdP de comportement interne (*internal* BPN), qui décrit le comportement interne du CC.

La notation et la sémantique générale utilisées pour les diagrammes à RdP sont présentées. Les principes concernant la représentation du comportement externe et la représentation du comportement interne des CC et CO sont détaillés. Ces propositions ainsi que celles données en partie II sont ensuite résumées.

# III.1. Les diagrammes à réseaux de Petri

## III.1.1. Principes des diagrammes à RdP

Ainsi qu'il est rappelé au § I.3.3.2.b, le couplage entre la notation UML et le modèle formel que sont les RdP peut se faire de trois façons différentes. Parmi celles-ci, l'approche dite « objets à RdP » a été choisie pour plusieurs raisons :

- Un grand choix de classes de RdP peut être envisagé et ce, dans une même modélisation. Cela permet d'utiliser la famille de RdP la plus adaptée à une représentation donnée. Par exemple, si le comportement d'un objet est très simple, un RdP ordinaire est suffisant et il n'est alors pas nécessaire d'utiliser un RdP de haut niveau.
- De plus, n'utilisant pas l'héritage et nous focalisant sur la partie dynamique de la modélisation, nous n'avions pas l'utilité d'une classe de RdP à objet[35].
- Enfin, puisque nous reprenons une grande partie des concepts de HOOD/PNO déjà développés au sein de notre équipe de recherche, il était préférable de garder une même approche.

Comme toute approche de type « objet à RdP », des extensions aux RdP doivent être définies pour prendre en compte, par exemple, les mécanismes de communication et l'invocation d'opération entre objets. Afin de faciliter la compréhension, ces extensions ont été regroupées en définissant un formalisme nommé diagramme à RdP. Les RdP sont utilisés pour modéliser le comportement des entités de la modélisation, la structuration reposant sur la notation UML.

Les entités dans UML/PNO sont naturellement les CO et les CC mais aussi, comme nous le verrons en partie IV, des acteurs de l'environnement ou des objets modélisant le réseau de communication (objet médium, cf. § II.2.1.2). De ce fait, nous retrouvons différentes utilisations des diagrammes à RdP, toutes basées sur ce même formalisme, suivant le comportement de l'entité modélisé. Ainsi tout au long de cette partie III, sauf dans le cas d'une mention explicite contraire, le terme d'objet désigne de manière générique ces différentes entités possibles de la modélisation, à savoir un CO ou un CC ou un acteur ou un objet médium.

## III.1.2. L'interprétation UML/PNO donnée aux diagrammes à RdP

Un RdP est un graphe constitué de places, de transitions et de jetons. Nous précisons l'interprétation que nous donnons à ces différents éléments.

### III.1.2.1. Les transitions

Les transitions sont associées aux événements d'un objet. Elles représentent des contraintes de changement d'état et des contraintes d'exécution des opérations qui leur sont éventuellement associées. Une transition peut être contrainte par une condition. Cette condition est, soit une expression algébrique booléenne, soit l'attente d'un événement. Suivant la classe de RdP utilisée, l'expression algébrique peut alors faire référence à des attributs portés par les jetons sur les places en aval de la transition (cf. Figure III-1 a).

Il y a deux types de transitions :

- Des transitions internes (représentées en noir), dont le franchissement est conditionné par des événements internes à l'objet.

[35] Notre objectif n'est pas d'utiliser les RdP pour modéliser l'aspect statique du système.

- Des transitions externes (représentées en blanc), dont le franchissement est conditionné par des événements externes à l'objet (appel ou réception de méthodes ou d'événements de l'environnement).

#### III.1.2.2. Les places

Les places sont associées aux activités et aux états d'attente d'un objet. Lorsqu'elle est associée à une place, une opération est activée si cette place comporte au moins un jeton. La place représente le fait que l'opération est en cours, sa transition d'entrée étant associée à l'événement "début d'opération" et sa transition de sortie à l'événement "fin d'opération" (cf. Figure III-1.a).

Il y a deux types de places :

- Des places internes modélisant des comportements internes de l'objet,
- Des places externes (représentées par un double cercle concentrique) modélisant des communications asynchrones entre objets et partagées par deux diagrammes à RdP.

#### III.1.2.3. Les jetons

Le placement des jetons dans les places du RdP (plus précisément le marquage du RdP) représente l'état d'un objet à l'instant considéré. Ainsi, des séquences de places et de transitions correspondent à des structures de processus séquentiels instanciées chaque fois qu'elles sont traversées par un jeton. De cette manière, la concurrence intra-objet est permise.

A un jeton peut être associé un type de données (on utilise dans ce cas un RdP de haut niveau). Les attributs de ce jeton peuvent être évalués lors du franchissement d'une transition et peuvent représenter des variables de l'objet modélisé[36].

#### III.1.2.4. Notation agrégée

Une méthode (ou une opération) est décrite par une séquence formée d'une transition, d'une place et d'une transition (cf. Figure III-1.b). Dans les cas où la durée estimée de l'opération n'a aucune conséquence au niveau du contrôle, la séquence peut être agrégée en une seule transition.

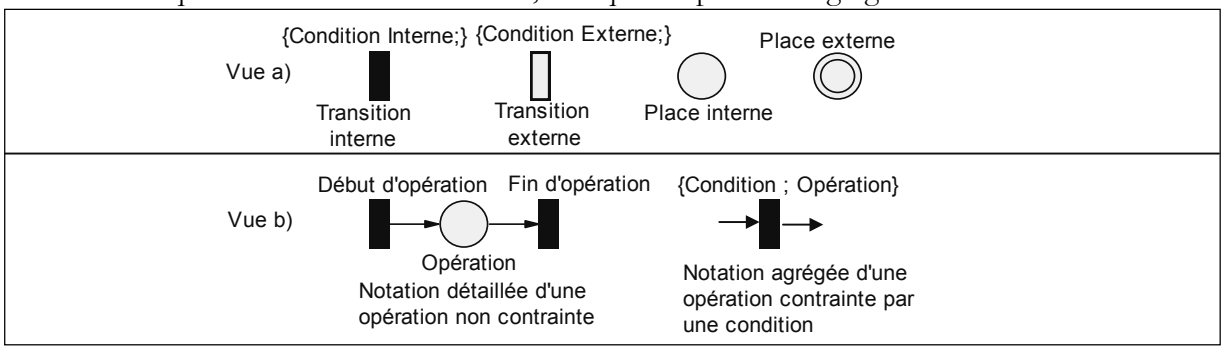


**Figure III-1 : Notations et symboles primaires pour les diagrammes à RdP**

## III.1.3. Représentation des contraintes temporelles

#### III.1.3.1. Quelles contraintes temporelles ?

Au § I.2.2.2, nous avons présenté différents types de Contraintes Temporelles (CT) et précisé qu'elles pouvaient être exprimées avec différents types de temps (physique, échantillonné, global, local, relatif, absolu).

Puisque nous nous plaçons dans un paradigme de modélisation asynchrone (cf. § I.2.5.3.),

[36] Suivant la classe de RdP de haut niveau, le jeton contient directement les attributs de l'objet (RdP colorés, prédicat/transition) ou une référence à un attribut de l'objet (RdP à objet). Nous ne détaillerons pas les mérites de chaque approche qui portent sur des aspects statiques.

nous ne pouvons pas faire l'hypothèse d'un temps global (une horloge partagée par tous les composants), ni même d'un temps absolu (possédant une même référence temporelle initiale pour les horloges des composants du système). Par conséquent, nos contraintes temporelles seront exprimées dans un temps local relatif. Chaque composant possédera sa propre horloge et les références initiales de ces horloges ne pourront être supposées égales.

Naturellement, lorsque les besoins exprimeront des contraintes temporelles globales (ou absolues), ils devront être traduits par des synchronisations explicites entre les composants et par des contraintes temporelles locales relatives.

La nécessité de manipuler des CT relatives évite d'établir un ordre temporel total entre les activités du système. Ainsi en conservant les relations d'ordres partiels entre les activités, nous pouvons plus facilement manipuler les CT et les modifier[37]. Les relations d'ordre restent explicites. Par contre, on s'expose à des risques d'incohérences entre les schémas de synchronisations. La vérification de ces schémas est plus difficile que dans une approche ayant un axe temporel unique où il suffit de s'assurer de la cohérence des CT sur l'axe temporel. Par conséquent, nous devrons offrir des techniques de vérification de la cohérence de ces CT. Ces techniques seront présentées en partie V.

En § I.2.4, nous avons identifié les grandes classes de difficultés que posaient les contraintes temporelles. Parmi celles-ci, nous avons évoqué la grande diversité d'expression des CT. Afin d'apporter une première réponse à ce problème, nous proposions une représentation unifiée des CT, présentée ci-après.

### III.1.3.2. Expression d'une contrainte temporelle dans UML/PNO

Par définition, une CT est une relation d'ordre entre événements temporels. Nous nous sommes volontairement limités à une expression des relations d'ordres élémentaires [Allen-1983] [Schmiedel-1998]. Nous verrons, en effet, que grâce à l'utilisation des RdP, nous pourrons étendre leur pouvoir d'expression.
Nous définissons une CT par la formulation donnée ci-après.

Définition 1 : Expression d'une contrainte temporelle

Nous définissons une contrainte temporelle entre deux événements par un quadruplet $<T_{deb}, T_{fin}, Intervalle, F_{dt}>$ où

- $T_{deb}$ et $T_{fin}$, sont les événements associés à la contrainte temporelle,
- Intervalle permet d'exprimer un intervalle temporel entre les deux événements de la forme [x,y] ou [x,y[ ou ]x,y] ou ]x,y[ et $\forall x,y \in Q^+$ tel que $x \geq y$
- $F_{dt}$ est une fonction de densité de probabilité sur l'intervalle temporel

Nous supposons que $T_{deb}$ survient toujours avant ou simultanément à $T_{fin}$ (dans le cas contraire, il faut inverser les deux événements). Nous constatons ainsi qu'avec cette expression, nous pouvons définir des CT qualitatives ou quantitatives. De plus, nous pouvons donner des informations probabilistes, comme des durées moyennes d'action, par l'utilisation d'une fonction de densité spécifiant les plages les plus probables de l'intervalle temporel. Dans un souci de

[37] En effet, en associant les CT à un axe temporel unique, les relations de synchronisation sur les activités sont alors exprimées par rapport à cet axe temporel et non plus par rapport aux activités. Il y a donc risque de perdre les informations concernant des ensembles de CT entretenant des relations de dépendance (des schémas de synchronisation). Dès lors, il devient difficile de modifier des relations de synchronisation tout en préservant la cohérence de ces schémas [Allen-1983].

simplification, nous nous limiterons à des fonctions probabilistes simples telles que des exponentielles, des distributions de Dirac ou des fonctions uniformes dont la densité de probabilité de tir sur l'intervalle est toujours égale à 1. Plus précisément, les fonctions de probabilités autorisées sont celles définies par [Gallon-1997] qui sont utilisées dans la classe de RdP temporisé stochastique déjà présentée au §I.3.3.3.c.

Le tableau 1 représente quelques exemples de relations d'ordre élémentaires qui peuvent être exprimées pour une CT avec une fonction de densité uniforme.

| Relation d'ordres partiels | Intervalle | Expression |
|---|---|---|
| $T_{deb}$ avant $T_{fin}$ | ]0,∞[ | $\infty > T_{fin} - T_{deb} > 0$ |
| $T_{deb}$ simultanément à $T_{fin}$ | [0,0] | $T_{fin} - T_{deb} = 0$ |
| $T_{deb}$ avant ou simultanément à $T_{fin}$ | [0,∞[ | $\infty > T_{fin} - T_{deb} => 0$ |
| $T_{deb}$ sans relation avec $T_{fin}$ | ]-∞,∞[ | Aucune |
| $T_{deb}$ exactement x unités de temps avant $T_{fin}$ | [x,x] | $T_{fin} - T_{deb} = x$ |
| $T_{deb}$ au moins ou exactement x unités de temps avant $T_{fin}$ | [x,∞[ | $\infty > T_{fin} - T_{deb} >= x$ |
| $T_{deb}$ au plus x unités de temps avant $T_{fin}$ | [0,x[ | $x > T_{fin} - T_{deb} > 0$ |
| $T_{deb}$ au plus ou exactement x unités de temps avant $T_{fin}$ | [0,x] | $x >= T_{fin} - T_{deb} > 0$ |
| $T_{deb}$ entre x et y unités de temps avant $T_{fin}$ | ]x,y[ | $y > T_{fin} - T_{deb} > x$ |

**Tableau 2 : Expression de relations d'ordre entre deux évènements temporels**

### III.1.3.3. La notion de contraintes temporelles attendues et posées

Lors de notre travail, nous avons constaté une différence fondamentale entre les contraintes temporelles concernant :

- Les CT qui sont des CT que l'on cherche à obtenir et dont le respect doit être vérifié tout au long du développement.
- Les CT qui sont, en quelques sorte, imposées à l'analyste-concepteur ou sur lesquelles il lui est difficile de renégocier.

Nous nommons les premières, les Contraintes Temporelles Attendues (CTA) et les secondes, les Contraintes Temporelles Posées (CTP).

Les raisons de cette différenciation ont été présentées au § I.2.3.3. En effet, nous avons vu que les analystes-concepteurs devaient s'assurer, durant toute la phase de développement, que leur modélisation respecte les contraintes temporelles demandées par les clients. Ces CT seront donc considérées comme des CTA. Elles représentent un besoin attendu par les clients. Elles peuvent être assimilées à des hypothèses temporelles à vérifier. Ce sont, par exemple, des CT externes de temps de réponse ou des CT internes de validité temporelle d'information.

La notion de CTA est indissociable de la notion de CTP, car pour pouvoir vérifier le respect de ces CTA, il nous faudra des CT exprimant des temps qui ne sont pas des hypothèses. Par analogie avec une formule mathématique, nous pouvons considérer les CTA comme des inconnues et les CTP comme les paramètres de cette formule.

Les CTP sont souvent des CT représentant des durées opératoires de traitement. Elles sont généralement fixées ou supposées par les analystes-concepteurs. Le plus souvent, elles sont

considérées comme vraies et n'ont pas à être vérifiées. Plus exactement, nous pouvons les considérer à un certain niveau d'abstraction (ou pendant une phase du développement) comme supposées et vraies. Puis, lorsque nous descendons dans les niveaux d'abstraction, nous pouvons reconsidérer ces CTP et ne plus les supposer vraies. Elles deviennent alors des CTA. Ce sera, par exemple, des CT internes d'action (temps d'exécution estimé d'une opération), des CT internes de relation entre actions (un délai pour une communication) ou des CT externes comme le comportement temporel d'un dispositif physique externe qui ne peut être remis en cause par l'analyste-concepteur...

En faisant une analogie avec les concepts utilisés en ordonnancement, nous pourrions dire qu'une CTA représente une fenêtre temporelle dans laquelle l'exécution d'une opération est souhaitable (contrainte au plus tôt et au plus tard), tandis qu'une CTP représente le temps d'exécution d'une opération (minimum et maximum).

Comme nous allons le voir, cette différenciation entre CTA et CTP est essentielle dans notre approche. En effet, toutes les aides, concernant les aspects temporels que nous proposons, reposent sur cette distinction.

Nous allons maintenant présenter la classe de RdP que nous utilisons dans les diagrammes à RdP. La définition de cette classe de RdP a été effectuée afin de pouvoir prendre en considération la représentation des CTA et des CTP.

## III.1.4. La classe de RdP temporel utilisée dans UML/PNO

### III.1.4.1. Motivations de représentation

Parmi tous les RdP dédiés à la représentation des CT, il nous fallait une famille permettant d'exprimer l'ensemble de nos CT qui soit, d'une part, compatible avec la représentation précédemment donnée (définition 1) et qui, d'autre part, permette d'utiliser des outils de vérification et de validation existants.

Utilisant des intervalles temporels pour l'expression des CT quantitatives, nous devions faire appel à des RdP temporels. Par ailleurs, voulant différencier de manière simple (et donc graphiquement) les CTA et les CTP, il nous fallait une classe de réseau qui permette une représentation distincte de ces CT. Pour ces raisons, notre choix s'est porté sur une classe de RdP où les considérations temporelles peuvent être placées sur les places (pour représenter les CTA) et sur les transitions (pour représenter les CTP).

Enfin, nous voulions pouvoir exprimer des considérations stochastiques sur l'intervalle temporel des CT. Plus précisément, ces informations probabilistes ne concernaient que les CTP[38].

Par conséquent, nous utilisons une classe de RdP à places temporelles et transitions stochastiques temporelles pour les diagrammes à RdP. Cette classe de RdP peut être considérée comme la réunion de deux classes :

- Celle des RdP à places temporelles de [Khansa-1997] (RdP p-temporel)
- Celle des RdP à transitions temporelles stochastiques[39] de [Atamna-1994] et [Gallon-1997] (RdP t-temporel stochastique).

Donnons les définitions formelles de cette classe de RdP.

[38] En effet, il y a peu d'intérêt de vouloir introduire des considérations probabilistes sur les CTA. Quelle serait l'interprétation à donner à une telle expression ?

[39] Les concepteurs de cette classe de RdP la nomment, en fait, classe des RdP temporisés stochastiques, bien que des intervalles de temps sur les transitions soient manipulés.

## III.1.4.2. Les RdP à places temporelles et transition temporelles stochastiques

### III.1.4.2.a) Définition formelle du RdP pt-temporel stochastique

Un RdP pt-temporel stochastique est un n-uplet <R, ISP, IST, FT> où :

- R est un RdP ordinaire marqué.
- ISP est une application de P vers $(Q^+ \cup \infty)x(Q^+ \cup \infty)$ ($Q^+$ ensemble des rationnels positifs ou nul) telle que : $p_i \rightarrow ISP_i=[a_i,b_i]$ avec $0 \leq a_i \leq b_i$. $ISP_i$ est "l'Intervalle Statique de Place" qui définit le temps de séjour d'une marque dans la place $p_i$. Un jeton dans la place $p_i$ ne participe à la validation de ses transitions de sortie que s'il a séjourné au moins la durée $a_i$ dans la place. Après la durée $b_i$, le jeton sera dit "mort" et ne participera plus à la validation des transitions.
- IST est une application de T vers $(Q^+ \cup \infty)x(Q^+ \cup \infty)$ ($Q^+$ ensemble des rationnels positifs ou nul) telle que : $t_i \rightarrow IST_i= [\theta_{mi},\theta_{Mi}]$ avec $0 \leq \theta_{mi} \leq \theta_{Mi}$. $IST_i$ définit un "Intervalle Statique de Transition" exprimant le fait qu'une transition doive être sensibilisée pendant le délai minimum $\theta_{mi}$ avant de pouvoir être franchie et ne peut rester validée au-delà du délai maximum $\theta_{Mi}$.
- FT est une fonction de densité de probabilité : A chaque transition $t_i$ est associée une densité de probabilité $f_i(x)$ telle que : $\int_{\theta mi}^{\theta Mi} f_i(x)dx = 1$. Nous autorisons uniquement l'utilisation des fonctions $f_i$ de type exponentiel, Dirac, ou uniforme, définies dans [Gallon-1997].

Il s'agit donc d'une classe de réseau où des considérations temporelles qualitatives, exprimées par des intervalles temporels de type [Min, Max], peuvent être attachées aux places (intervalle temporel $[a_i,b_i]$) et aux transitions (intervalle temporel $[\theta_{mi}, \theta_{Mi}]$) du RdP. En outre, à l'intervalle temporel de la transition, peut être ajoutée une fonction de densité de tir.

### III.1.4.2.b) Règles de fonctionnement

Les définitions des états et des règles de franchissement des transitions pour une telle classe de RdP sont maintenant données.

L'état d'un RdP pt-temporel est défini par un ensemble E=(M, IDT, IDP), avec :

- M, une application de marquage, affectant à chaque place du réseau un certain nombre de jetons.
- IDT, une application "intervalle de tir", associant à chaque transition sensibilisée du réseau l'intervalle de temps, dit "Intervalle Dynamique de Transition" dans lequel elle peut être tirée. Cet intervalle peut être différent de "l'Intervalle Statique de Transition", car la référence temporelle n'est pas l'instant de sensibilisation de la transition, mais l'instant d'arrivée dans l'état E.
- IDP, une application "intervalle potentiel", associant à chaque jeton k dans une place $p_i$ un intervalle de temps, dit "Intervalle Dynamique Potentiel", dans lequel le jeton k peut être utilisé pour le tir d'une transition. Supposons que le jeton k, arrive dans la place $p_i$ (ayant un intervalle statique $[a_i,b_i]$) à l'instant c, alors à l'instant c+d, l'intervalle dynamique de k est $[\max(a_i-d,0),b_i-d]$.

Une transition t est dite potentiellement franchissable à l'instant $\theta$ depuis un état E(M,IDP,IDT) ssi :

- $\forall p \in P, M(p) \geq Pré(p,t)$ (condition des RdP ordinaires).
- L'intersection des IDP des jetons participant au tir est non vide et il n'existe pas de jetons (ne participant pas au tir) dans les places dont les bornes supérieures de leur IDP soient strictement inférieures à $\theta$. Sinon il y a mort de jetons (condition des RdP p-temporels).

- θ est compris (bornes incluses) dans l'intervalle dynamique (IDT) associé à la transition dans E (condition des RdP t-temporels).
- La probabilité de tir de la transition t à l'instant considéré est non nulle (condition des RdP t-temporels stochastiques).

**La sémantique choisie est la sémantique forte avec une politique de sensibilisation multi-serveur** (cf. § I.3.3.3.b). Afin de permettre l'utilisation de RdP non-saufs (et traiter le cas de la multi-sensibilisation), nous associons explicitement à chaque jeton sa date de production (*ProductionTime)* et de consommation (*ConsumptionTime )* dans une place. Dans ce cas, les règles de franchissement sont simplifiées puisque nous pouvons alors associer à chaque jeton une application "temps de séjour" dans une place permettant, en fonction de ces dates, de connaître son âge. Nous verrons en partie V une autre raison à la définition de ces attributs.

**Enfin, précisons que, par convention, toute transition dont l'intervalle temporel n'est pas spécifié, est considérée comme immédiate.**

### III.1.4.3. Interprétation des places et transitions temporelles

Rappelons, sans plus tarder, que cette classe pour les diagrammes à RdP ne sera utilisée que pour une description des aspects dynamiques du système. L'utilisation d'une telle classe de RdP est faite afin de différencier la représentation des CTA et des CTP. Les CTA peuvent être associées à des places temporelles du RdP, tandis que les CTP à des transitions temporelles du RdP.

En effet, lorsque des analyses formelles ou des simulations seront faites sur cette classe de RdP, nous nous ramènerons :

- A des RdP t-temporel [Merlin-1974] [Merlin-1976] pour l'analyse,
- A des RdP t-temporels stochastiques [Atamna-1994] [Gallon-1997] ou simplement à des RdP t-temporels pour des simulations ou des évaluations des performances.

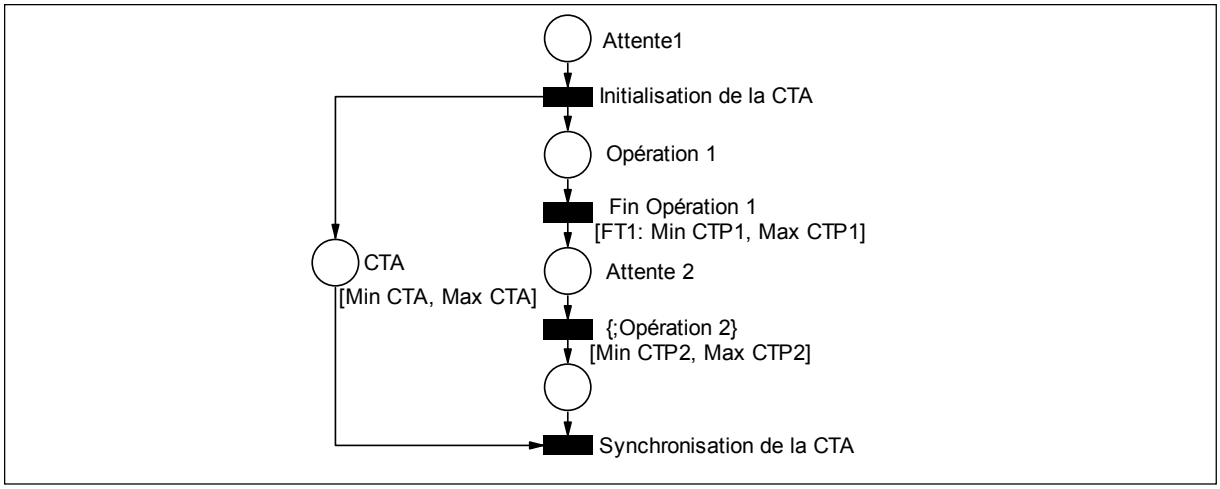


**Figure III-2 : Représentation des contraintes temporelles sur un diagramme à RdP**

La Figure III-2 récapitule la notation utilisée pour représenter les différentes CT. Nous avons modélisé des CTP concernant la durée des opérations 1 et 2. Ces CTP modélisent le fait que les opérations prennent entre Min CTP et Max CTP unités de temps pour s'exécuter.

A l'opération 1, nous avons associé une densité de probabilité FT1. La représentation de l'opération 1 est une représentation détaillée. L'opération 1 est associée à une place. La CTP de l'opération est associée à la transition de sortie de la place. La représentation de l'opération 2 est

une vue agrégée. L'opération et sa CTP sont alors associées à la transition. La CTA représente le fait qu'il est souhaité que l'exécution des opérations 1 et 2 soit comprise dans l'intervalle de temps [MinCTA, MaxCTA]. Par convention de notation, la transition en entrée de la CTA, est appelée "transition d'initialisation de la CTA". La transition de sortie de la CTA est nommée "transition de synchronisation de la CTA". Ces deux transitions sont importantes car elles définissent explicitement et de manière précise le début et la fin du traitement auquel est associée la CTA (dans notre cas l'exécution séquentielle des opérations 1 et 2).

Dans l'exemple de la Figure III-2, il semble donc que cette CTA soit respectée si la somme des intervalles des CTP des opérations 1 et 2, c'est-à-dire [Min CTP1 + Min CTP2, Max CTP1 + Max CTP2], est comprise dans l'intervalle [Min CTA, Max CTA]. Cet exemple permet de mieux comprendre les avantages de notre différenciation entre CTA et CTP et leur affectation distincte sur le RdP. Il devient alors simple d'identifier rapidement les CT à vérifier (les CTA).

Signalons que le principe de vérification que nous venons de présenter (somme des intervalles des CTP) n'est plus une condition suffisante (elle est seulement nécessaire), dès lors que l'opération 2 nécessite des ressources partagées. Dans ce cas, le délai d'attente ne pourra plus être nul, il dépendra de la disponibilité de la ressource.

Par ailleurs, il est important de bien comprendre le rôle donné à la place CTA : celui de ne représenter qu'une hypothèse temporelle et non de modéliser une partie du contrôle du système. Ainsi, dans le cas du non-respect d'une CTA (entraînant la mort d'un jeton dans la place), aucun mécanisme implicite n'est défini pour modéliser des alternatives.

Si des alternatives doivent être modélisées, elles ne doivent pas être représentées par des CTA mais par des CTP. En effet, la détection et la mise en place d'une alternative nécessitent, au moins, une modélisation par chien de garde (représentant un chronomètre et donc le lancement d'une activité ayant une durée opératoire). Par conséquent, s'il doit y avoir mise en place d'une alternative, elle devra être explicitement modélisée par des transitions temporelles.

Le formalisme et la sémantique du diagramme à RdP viennent d'être présentés. Leur utilisation pour la représentation du comportement externe et interne d'un CC et de ses CO est maintenant donnée.

# III.2. Le diagramme de comportement externe d'un CC

Le rôle d'un eBPN (external Behavioural Petri Net) est de compléter la description du contrat d'un CC. Notre objectif est de pouvoir avoir une description simplifiée de la dynamique du contrat d'un CC et d'utiliser cette description lors de la définition du système et des collaborations entre CC sans avoir recours à leur vue interne.

Cela comprend des indications sur :

- Les mécanismes de communication (synchrone ou asynchrone) des opérations offertes par le CC.
- Les effets observables de l'invocation d'une opération d'un CC si cette invocation déclenche une requête ou des requêtes vers d'autres CC.
- Les conditions temporelles externes du CC entre opérations offertes et requises.

Avant de donner la représentation de chacun de ces trois points, précisons qu'il n'y a pas de différence de représentation (au niveau du RdP de comportement) entre un CC et un CO. Naturellement les CO à un instant t auront des marquages différents (chaque CO possède son propre marquage) mais ils utilisent la même description du RdP de comportement du CC dont ils sont l'instance.

## III.2.1. Représentation des mécanismes de communication

### III.2.1.1. Motivations

Concernant la représentation des mécanismes de communication, le but de l'eBPN est simplement d'indiquer si les opérations offertes par le CC sont synchrones ou asynchrones. Il n'a pas à indiquer quel sous-CC est chargé de rendre le service (cela se fait par la relation <<in>> dans la partie interne du CC).

### III.2.1.2. Représentation d'une communication en UML/PNO

Pour illustrer les mécanismes de communication, nous supposons un système composé de quatre objets. Un sous ensemble de leurs relations de communication est représenté sous deux vues différentes : par un diagramme de collaboration (Figure III-3.a) et un diagramme de séquence (Figure III-3.b).

L'objet "Objet1" effectue d'abord une requête asynchrone à l'objet "Objet2" , puis une requête faiblement synchrone à l'objet "Objet3" et enfin une requête fortement synchrone à l'objet "Objet4". Notons qu'au niveau des diagrammes, nous représentons ces différents protocoles de synchronisation. Ce sont des extensions à la représentation des communications entre les objets afin de différencier les différents protocoles de communication. Ces extensions sont inspirées des propositions faites par Douglass dans son livre sur UML et le temps réel [Douglass-1998].

Détaillons ces extensions présentées en Figure III-4. Toutes les communications par invocation d'opération sont représentées par des flèches pleines. Les communications ne faisant pas appel à une invocation d'opération ou n'ayant pas de protocole particulier (flux de données en provenance des acteurs de l'environnement, signal ou message simple) sont représentées avec une flèche non pleine.

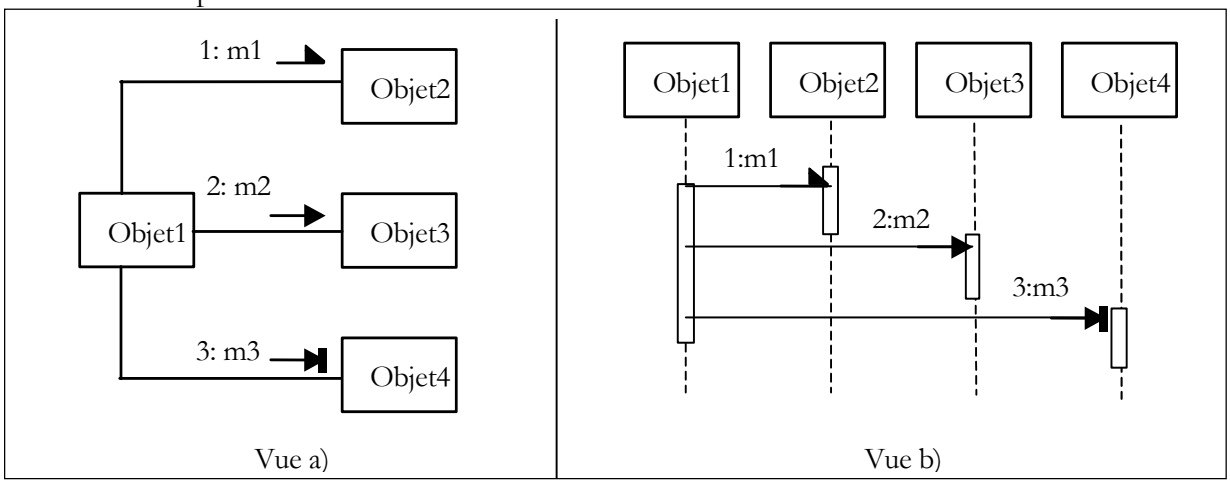


**Figure III-3 : Diagrammes de collaboration et de séquence de relations inter-objets**

Puisque dans UML/PNO, ces types de communication ne sont pas supportés (cf. § II.2.1.2), il faudra que l'analyste-concepteur les traduise.

Néanmoins, au cours du développement (en particulier en phase d'analyse), nous permettons l'utilisation de message simple, soit car le protocole n'est pas encore déterminé, soit car la prise en compte des données venant des acteurs du système n'est pas encore définie.

Notons que pour chaque communication synchrone faible le retour des résultats est spécifié par une flèche pointillée (afin d'indiquer qu'il s'agit d'un retour) et nous indiquons si ce retour est une communication synchrone faible ou asynchrone.

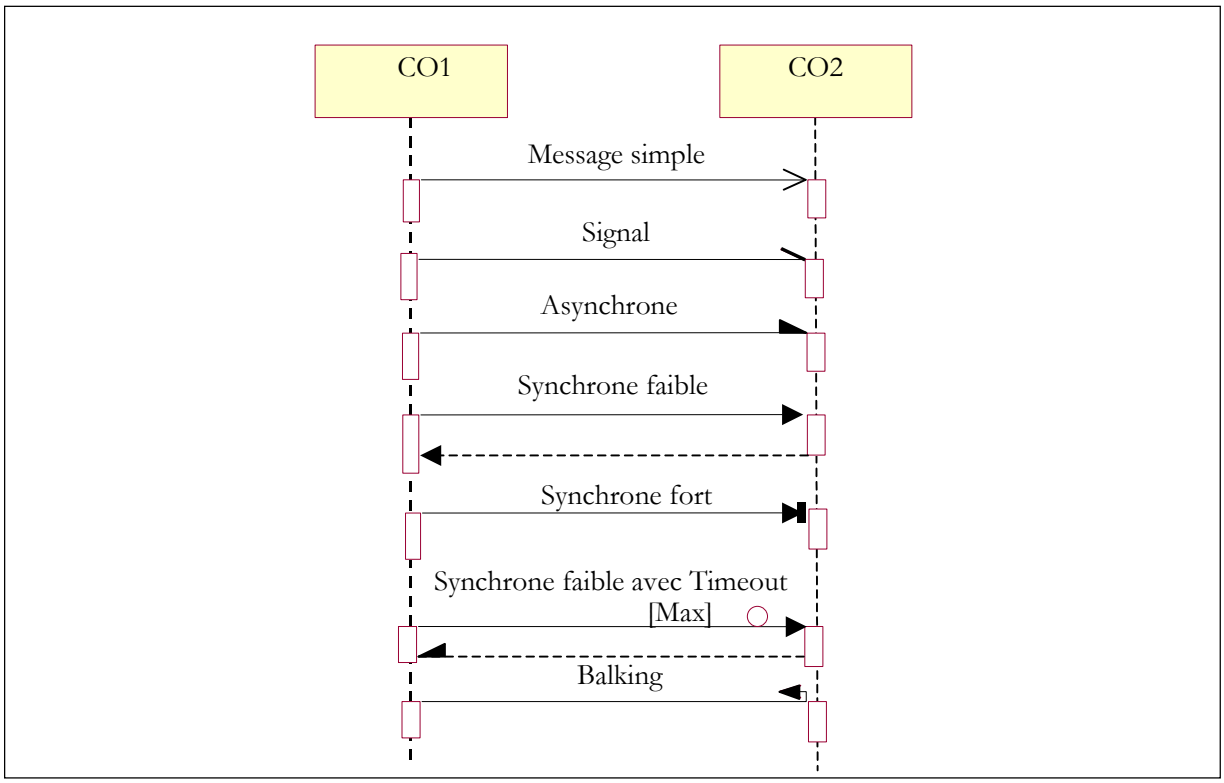


**Figure III-4 : Représentation des protocoles de communication UML/PNO**

Les attentes limitées sont utilisées en reprenant la notation de G. Booch. L'attente est précisée par un intervalle temporel. La communication de type *balking* est une communication avec attente limitée d'une durée infinitésimale.

### III.2.1.3. Représentation détaillée des communications par RdP

Dans un diagramme à RdP, les spécifications des communications de l'exemple de la Figure III-3.b sont données par la Figure III-5.a.

La communication asynchrone est modélisée à l'aide d'une place partagée (le médium de communication) et d'un jeton (modélisant l'occurrence du message, c'est-à-dire l'occurrence de l'événement de type *call event*) que le client pose à destination du serveur lorsqu'il établit sa requête. Rappelons qu'un message UML d'appel d'opération est défini par l'événement invoquant l'opération (*call event*). De même, toute communication par message simple est considérée comme une communication asynchrone.

Les communications synchrones sont modélisées à l'aide de deux places partagées. Dans le cas d'un synchronisme faible (message "m2"), la première place véhicule la requête et la seconde fait office d'acquittement. Figure III-5.a, une telle communication est représentée entre "l'objet1" et "l'objet3".

Dans le cas du synchronisme fort (message "m3"), les deux places ont la même signification globale que dans le cas précédent sauf que l'acquittement n'est pas donné tant que l'opération "op4" n'est pas exécutée par le serveur "objet4".

Il s'agit ici du paradigme plus connu sous le nom d'appel de procédure distante (ou *Remote Procedure Call*). Le client "objet1" attend la fin de l'exécution et éventuellement les résultats (jeton dans la place "m'3") avant de poursuivre son fil de contrôle. Dans ces descriptions, les jetons peuvent être anonymes ou porter des données (ce sont alors les paramètres associés au message) suivant les besoins de l'application.

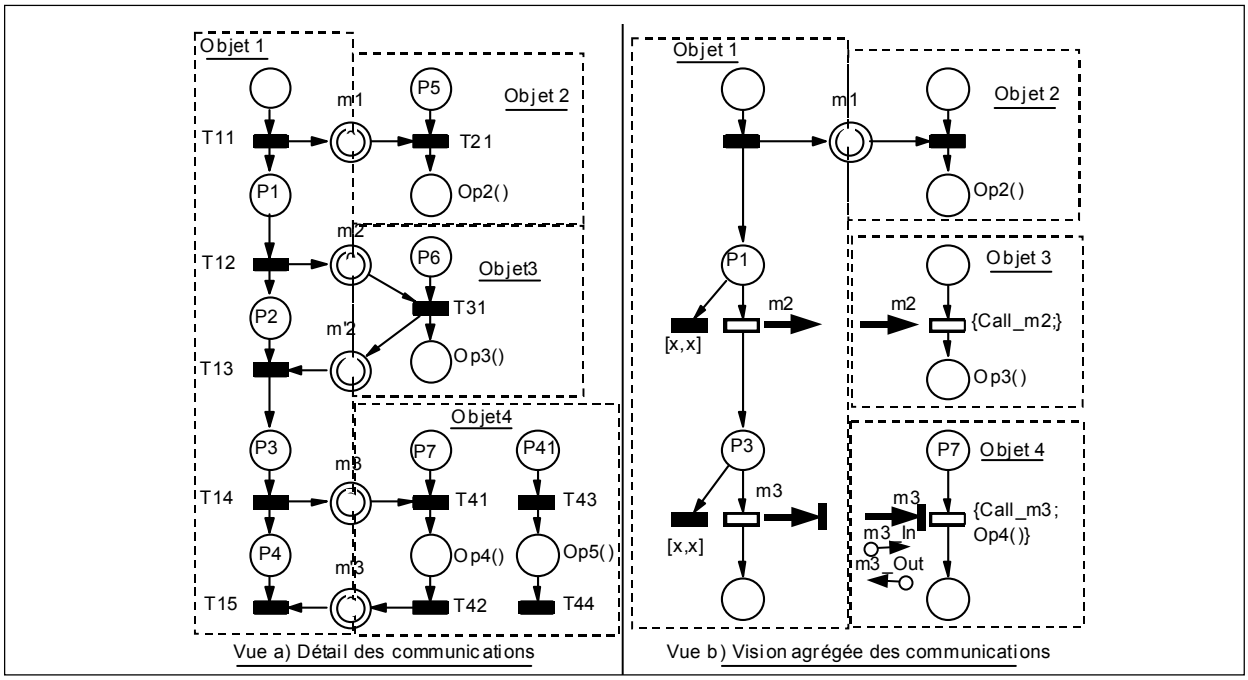


**Figure III-5 : Spécification par RdP des messages de l'exemple de la Figure III-3**

Les aspects dynamiques apparaissent ici de façon simple. Les attentes éventuelles de synchronisation sont modélisées par des places (P2, P4… pour "objet1", P5 pour "objet2", P6 pour "objet3" et P7 pour "objet4").

### III.2.1.4. Représentation agrégée des communications synchrones par RdP

Il est possible de simplifier la représentation des communications synchrones entre objets en agrégeant les places partagées et en les représentant alors par une transition partagée.

Après une telle agrégation, les communications présentées par la Figure III-5 a), peuvent être représentées par les RdP de la Figure III-5 b). Cette sémantique rend plus indépendantes les représentations des comportements des objets et va permettre une exploitation des réseaux associés aux objets par décomposition ou par composition de places et/ou de transitions (suivant l'objectif poursuivi par le concepteur). Notons qu'elle facilite également certaines représentations comme les chiens de garde. Deux exemples de requête avec attente limitée ont été rajoutés sur l'objet 1 de la Figure III-5 b) au niveau des communications synchrones. Si l'objet 3 n'accepte pas la requête m2 dans un délai égal à x, alors l'objet 1 renonce et son fil d'exécution est dérouté. Le même raisonnement peut être conduit sur le message m3. Précisons que cette représentation agrégée n'est pas toujours utilisée, en particulier lorsque des CTA et des CTP sont en relation avec la communication. Dans cette dernière situation, nous utilisons la représentation détaillée.

L'équivalence des représentations des communications entre places et transitions partagées a été traitée dans la thèse de Mario Paludetto [Paludetto-1991].

### III.2.1.5. Exemple de représentation des communications d'un eBPN

Les deux diagrammes de séquence, donnés en Figure III-6, représentent les communications entre le CO *S_Vehicle* et le CO *S_Joystick*.

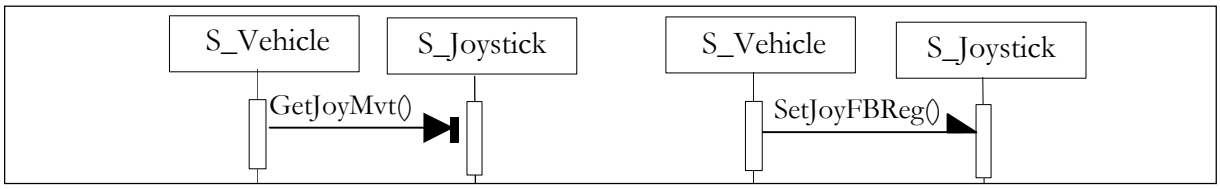


**Figure III-6 : Exemple de diagrammes de séquence externes pour le CO *S_Joystick***

Afin de simplifier notre exemple, nous n'avons pas représenté la transmission des messages effectuée par le CO *S_Cockpit*. Deux communications sont représentées : un appel synchrone fort (*GetMvt*) et une invocation asynchrone (*SetJoyFBreg*). La Figure III-7 donne leur représentation sur l'eBPN du CO *S_Joystick* en donnant la vue détaillée (vue a) et agrégée (vue b).

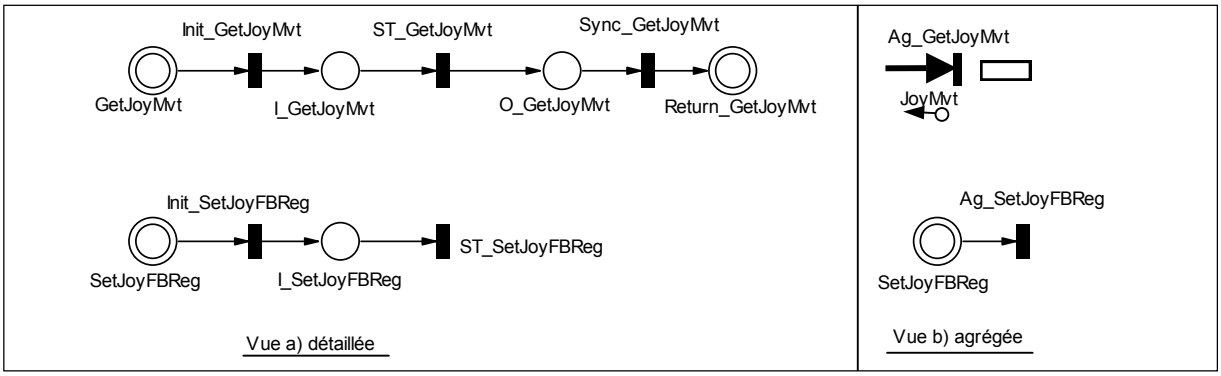


**Figure III-7 : eBPN partiel du CO *S_Joystick***

Pour l'invocation synchrone de « *GetJoyMvt* », cette communication, dans la vue détaillée, est décomposée en 3 transitions internes, 2 places externes et 2 places internes. Une telle décomposition (la communication pourrait être simplement décomposée par une transition interne et deux places externes) est définie afin de modéliser les effets observables d'une opération mais aussi permettre une représentation aisée des contraintes temporelles. Les explications concernant cette décomposition sont données dans les prochains paragraphes.

## III.2.2. Représentation des effets observables des invocations

### III.2.2.1. Motivations

La représentation des mécanismes de communication n'est pas suffisante, il faut aussi modéliser les effets observables d'une invocation d'opération. Par effet observable, nous entendons le comportement du CC qui se traduit par des appels à d'autres opérations externes à ce CC (les opérations requises par le CC).

En revenant sur l'exemple de la Figure III-7 a), nous constatons que nous avons déjà appliqué ce principe. Pour la communication synchrone (*GetJoyMvt*), nous avons représenté la génération de la réponse. Pour la communication asynchrone (*SetJoyFBreg*), nous avons modélisé la consommation du jeton à l'aide d'une transition puits.

### III.2.2.2. Représentation des effets observables des invocations

La représentation de ces effets se fait par un ensemble de places et de transitions préfixées des lettres *ST_* (pour *Supposed Treatment*). Elles représentent une abstraction du comportement de l'eBPN et peuvent être considérées comme des suppositions faites sur le comportement interne du CC.

De par le principe de représentations des opérations en vue détaillée (cf. Figure III-8), il y a systématiquement des places représentant l'invocation ou la requête d'une opération. Ces places

sont identifiées par leur nom[40], elles commencent respectivement par les lettres *I_* (pour *IN*) et *O_* (pour *OUT*), suivi du nom de l'opération. Ces places jouent un rôle particulier car elles sont l'interface entre le BPN externe et le BPN interne d'un CC. Pour ces raisons, nous les nommons places internes de communication. Leur rôle sera détaillé en §III.3.

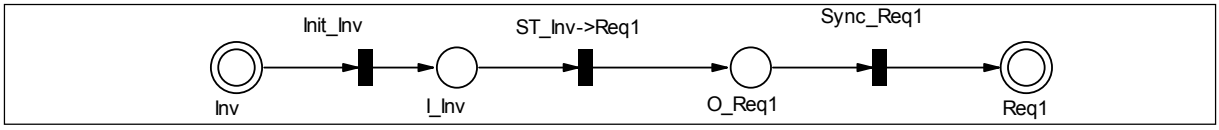


**Figure III-8: Principe pour la représentation des effets observables sur un eBPN**

Entre ces places internes de communications, les traitements fictifs (transitions et places préfixés des lettres *ST_*) sont représentés. Figure III-8, nous avons modélisé le fait que l'invocation de l'opération *inv* entraîne alors l'appel de l'opération *Req1*.

### III.2.2.3. Représentation des alternatives sur un eBPN

Dans le cas d'une alternative où une invocation d'une opération entraîne l'appel d'une opération ou d'une autre opération, la résolution du choix peut être effectuée par le comportement interne du CC.

Dans ce cas, le contrat externe du CC (et donc l'eBPN) n'est pas en mesure de lever l'indéterminisme. Nous représentons alors les deux alternatives possibles par deux traitements supposés (cf. Figure III-9).

Dans d'autres cas, l'indéterminisme peut être levé en fonction des paramètres des jetons et exprimé par une condition sur le franchissement d'une transition.

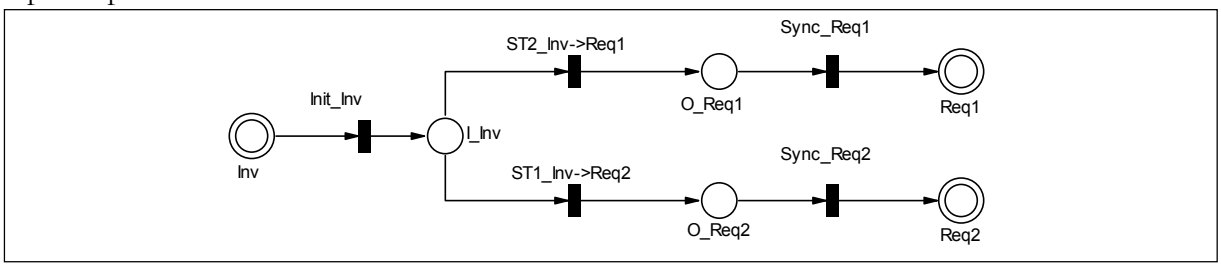


**Figure III-9: Représentation des alternatives sur un eBPN**

### III.2.2.4. Représentation des appels multiples sur un eBPN

Une autre situation peut survenir, proche de l'alternative, quand une invocation provoque l'appel de plusieurs opérations. Dans ce cas, nous considérons que le traitement et la génération des requêtes se font de manière parallèle (cf. Figure III-10).

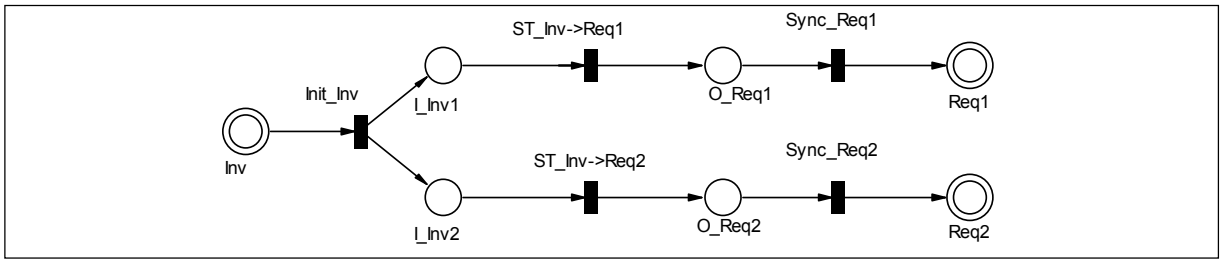


**Figure III-10: Représentation d'appels multiples sur un eBPN**

[40] Précisons que cette convention de nommage est interne à la description des BPN, l'analyste-concepteur peut utiliser les noms qu'il veut. Nous utilisons ici cette convention pas seulement dans un souci pédagogique. Nous verrons en partie IV que cette convention est utilisée par les techniques de dérivation des diagrammes UML vers les RdP.

Cette représentation est possible car le comportement externe d'un CC n'a pas à modéliser les mécanismes internes du CC concernant la gestion du parallélisme ou l'accès exclusif à des ressources. Le comportement externe d'un CC est donc vu comme une entité disposant de ressources infinies. Dans l'exemple donné à la Figure III-10, nous n'avons pas introduit de relation d'ordre entre les appels de Req1 et Req2. Nous ne savons pas quelle opération sera appelée en premier. L'ajout de ce type de relations d'ordre est possible et sera décrit au §III.2.3.3.

#### III.2.2.5. Représentation des temps d'exécution sur un eBPN

Concernant les transitions de traitement supposé, elles peuvent être enrichies par des CTP, indiquant des temps d'exécution supposés. Par exemple, à la Figure III-8, nous pourrions ajouter un intervalle temporel à la transition *ST_Inv1->Req1* précisant le temps mis par le CC à appeler l'opération *Req1* suite à l'invocation de *Inv*. Détaillons cette représentation des contraintes temporelles.

## III.2.3. Les contraintes temporelles externes d'un CC

#### III.2.3.1. Motivations

Nous voulons représenter sur les eBPN toutes les contraintes temporelles que l'utilisateur d'un CC peut émettre sans en connaître le fonctionnement interne. Pour désigner ce type de CT, nous parlons de CT externe du CC. Ce sont des contraintes temporelles entre les opérations offertes et invoquées par le CC. Cela peut être, par exemple, le temps mis par un CC à remplir un service. Cela peut être aussi un intervalle de temps entre la réception d'une invocation et la réaction du CC par invocation d'une opération sur un autre CC. Nous pensons que ce type d'information doit être joint à la description du contrat d'un CC (cf. § II.2.3.1).

#### III.2.3.2. Représentation des contraintes temporelles en UML/PNO

Les CT externes d'un CC sont représentées sur un diagramme de séquence à l'aide de notes UML. Elles peuvent être complétées par des indications graphiques de la notation UML. Mais ces informations graphiques peuvent être ambiguës et avoir plusieurs interprétations. Pour ces raisons, nous ne prenons en considération que les CT exprimées sous forme de notes.

La définition d'une CT par une note se fait suivant la notation <<Type_CT>> :{Nom_CT ; Expression temporelle}". Le champ "Nom_CT" permet de donner un nom à une CT. Le stéréotype <<type_CT>> indique s'il s'agit d'une CTA ou d'une CTP. L'expression temporelle définit de manière quantitative la contrainte. Elle suit la définition donnée en § III.1.3.2 (proposition 2). Elle peut utiliser les primitives UML *Event.sendTime* et *Event.receiveTime*, mais aussi celles proposées dans la réponse [OMG-2002d] au RFP « UML *Profile for Schedulability, Performance and Time* », portant sur des actions ou des stimuli.

Donnons quelques exemples de ces CT exprimées sous forme de notes :

1. <<CTA>> {Temps de réponse, *Stimulus.receiveTime*, *Reponse.sendTime*, [0,20], 'ms'} exprime un temps de réponse (une CTA de validité). Entre la réception d'un stimulus et la génération de la réponse, il ne doit pas s'écouler plus de 20 millisecondes.
2. <<CTA>> {Temps entre plusieurs occurrences et une réponse, *Stimulus([1-5],i=1).receiveTime*, *Reponse.sendTime()*, [10,50], 'ms'} exprime une CTA entre la première de cinq occurrences successives d'un stimulus et une réponse.
3. <<CTP>> {Temps d'activité; *Message1.receiveTime*, *Message2.sendTime*, [0,20], 'ms', cont} exprime un temps de traitement entre la réception d'un message et la génération d'un second message. La probabilité est uniformément continue (« cont »).
4. <<CTP>> {Temps de communication, *Message.sendTime*, *Message.receiveTime*, [0,10], 'ms', cont} exprime un délai de communication (une CTP de relation entre actions). L'envoi d'un message (en fait l'événement associé à ce message, dans l'esprit UML) prend entre 0 et 10

millisecondes. La probabilité est uniformément continue.

Notons que ce sont des CT exprimées sous forme de temps relatif (et non absolu). Ainsi que nous l'avons dit au § III.1.3.1, toute CT absolue devra être transformée en CT relatives. Cette transformation est de la responsabilité de l'analyste-concepteur.

### III.2.3.3. Représentation des CT sur un eBPN

Les CT externes s'appliquent entre des invocations et des requêtes d'opérations du CC.
En ce qui concerne les CTP, nous avons déjà évoqué leur représentation en §III.2.2.5. Elles sont portées sur les transitions modélisant les traitements supposés de l'eBPN (transitions et places préfixées des lettres *ST_*). Sur la Figure III-11, nous avons représenté 2 CTP.

Cependant, nous n'avons pas détaillé la représentation des CTA. Figure III-11, nous avons représenté un eBPN ayant deux opérations offertes (*inv1* et *inv2*) et deux opérations appelées (*Req1* et *Req2*).

En amont des places *I_*, une transition (commençant par *Init_*) permet de représenter les CTA portant sur l'opération offerte. Par symétrie, nous avons une transition (commençant par *Sync_*), située en aval des places *O_*, destinée à modéliser les CTA portant sur l'opération appelée par le CC.

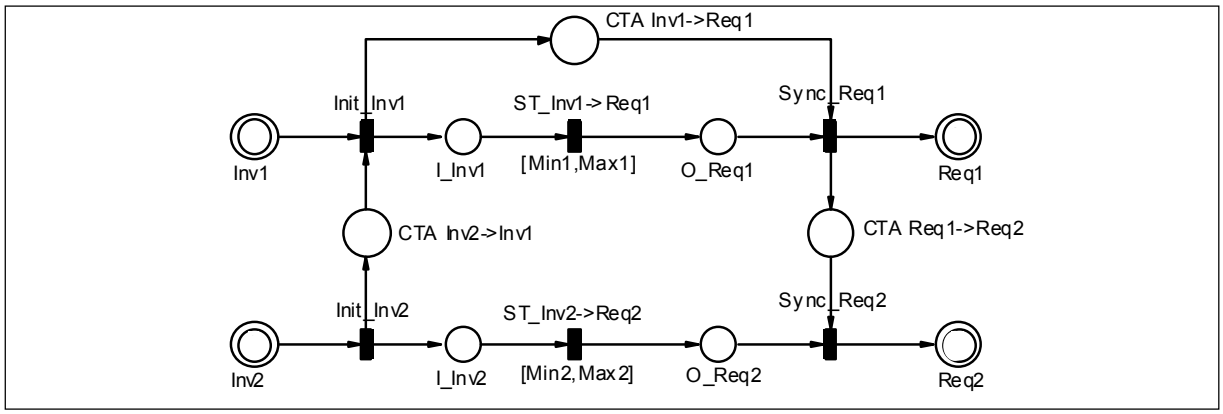


**Figure III-11: Principe de représentation des CT externes sur un eBPN**

Trois CTA ont été représentées sur la Figure III-11. La première (CTA *Inv2->Inv1*) indique une CT entre l'invocation de l'opération *Inv2* et *Inv1*. La seconde (CTA *Inv1->Req1*) précise un temps de réponse attendu entre l'appel de l'opération *Inv1* et la réaction du CO par appel de l'opération *Req2*. La dernière CTA (CTA *Req1->Req2*) modélise une relation d'ordre entre l'appel des opérations *Req1* et *Req2*.

### III.2.3.4. Exemple de représentation des CT d'un eBPN

Pour illustrer la représentation des CT externes sur les eBPN, nous continuons à détailler les fonctionnalités du CO *S_Joystick*.

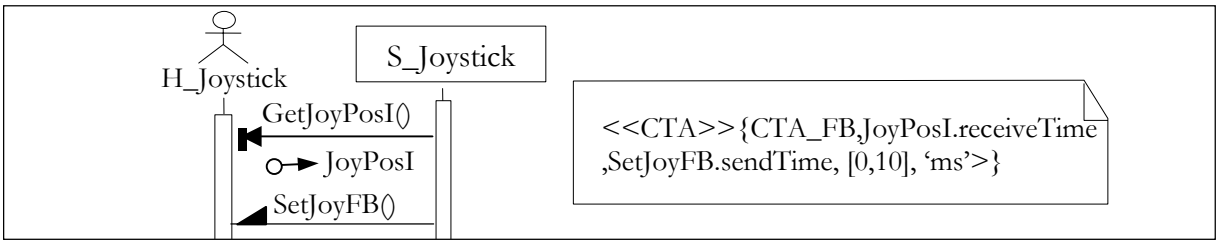


**Figure III-12 : CT externe pour le CO *S_Joystick***

Le retour de forces à appliquer au joystick doit être effectué moins de 20 ms après la réception de la position du joystick. Nous donnons la représentation de cette contrainte sur un diagramme de séquence en Figure III-12.

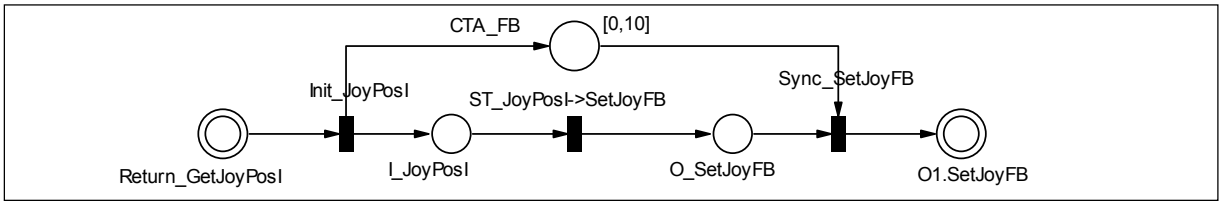


**Figure III-13: eBPN partiel du CO *S_Joystick* avec une CTA**

Figure III-13, nous complétons la représentation de l'eBPN *S_Joystick* déjà vue en Figure III-7. Nous n'avons représenté que les parties de l'eBPN *S_Joystick* où s'appliquent la CTA, entre le retour de l'appel synchrone fort de l'opération *GetJoyPosI* et l'envoi du retour de force.

## III.2.4. L'eBPN du CO S_Joystick

Afin de résumer nos explications concernant les eBPN, nous complétons une partie de l'eBPN du CO *S_Joystick* (cf. Figure III-15). Nous rappelons ses dépendances (cf. Figure III-14).

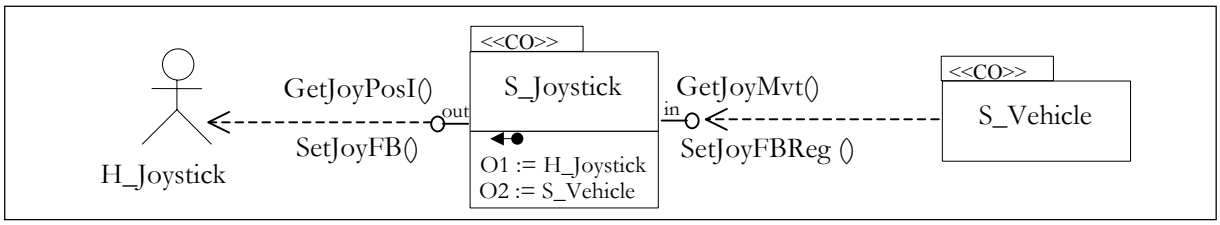


**Figure III-14 : Dépendances du CO *S_Joystick***

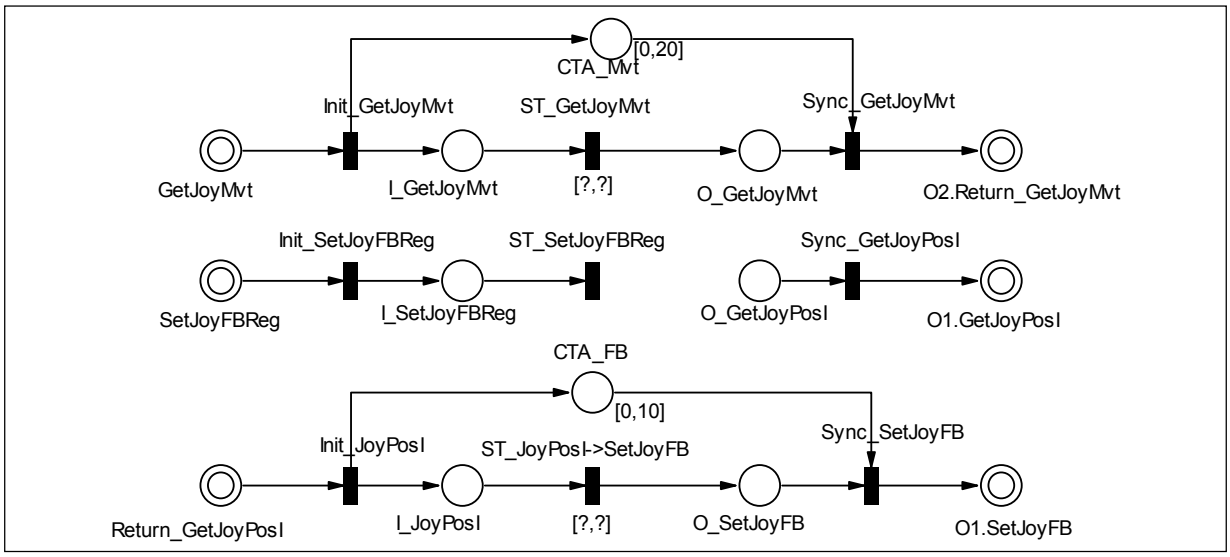


**Figure III-15 : Vue détaillée partielle de l'eBPN du CO *S_Joystick***

Sur la Figure III-15, nous retrouvons une partie des places et transitions déjà expliquées dans les paragraphes précédents. Une CTA (place *CTA_Mvt*) a été ajoutée sur la demande du mouvement du joystick (opération synchrone forte *GetJoyMvt*).

Afin de simplifier nos explications sur l'eBPN, nous donnions, jusqu'à présent, le nom des opérations requises sans préciser quel CO était appelé. Nous voyons maintenant que l'eBPN utilise la liste des CO définis dans le contrat (compartiment *CO_dependency*). Par exemple, sur la Figure III-15 l'opération *SetJoyFB* est invoquée sur l'objet O1.

De par son principe de construction, l'eBPN est toujours en mesure d'accepter les invocations de ses opérations offertes. En outre le traitement des invocations est immédiat, il n'y a pas prise en compte des temps d'attente d'une requête (en considérant, par exemple, que le CC est occupé par un autre traitement). Par cet aspect, en considérant que le CC est doté de ressources infinies, l'eBPN est une représentation simpliste du comportement du CC. Lorsque l'on doit mieux définir le CC, en terme de réalisation, ces hypothèses de ressources infinies sont peu à peu abandonnées. C'est justement le rôle du diagramme à RdP de comportement interne (iBPN) de modéliser de manière réaliste le comportement du CC. Détaillons ce diagramme.

# III.3. Le diagramme de comportement interne d'un CC

Un iBPN (*internal Behavioural Petri Net*) décrit le comportement interne d'un CC ou CO actif et explique comment les besoins décrits au niveau de l'eBPN sont réalisés. En ce sens, un iBPN est toujours dépendant de l'eBPN qu'il réalise.

A partir des opérations offertes et requises d'un eBPN, mais aussi d'opérations internes au CC, l'iBPN décrit la structure de contrôle du CC en établissant toutes les relations d'ordre entre ces opérations.

Par rapport à l'eBPN, l'iBPN remplace les traitements supposés (places et transitions préfixées des lettres « *ST_* ») de l'eBPN. Les problèmes de concurrences, de ressources partagées, de synchronisations… sont alors pris en compte par l'iBPN.

Une grande partie des principes des diagrammes à RdP et donc des iBPN a déjà été présentée dans les paragraphes concernant les eBPN. Par exemple, un iBPN peut porter des CTA ou des CTP comme l'eBPN. Néanmoins certains points doivent être plus détaillés. Cela concerne :

- Le couplage entre eBPN et iBPN,
- La représentation des comportements périodiques,
- L'initialisation et l'arrêt du contrôle,
- La représentation des données.

Nous détaillons chacun de ces points en nous appuyant principalement sur l'exemple simplifié du joystick.

## III.3.1. Le couplage entre iBPN et eBPN

### III.3.1.1. Principes du couplage entre iBPN et eBPN

Comme nous l'avons déjà expliqué, il faut imaginer l'iBPN se substituant aux transitions supposées de l'eBPN. Par conséquent, la modélisation de l'iBPN se fait à partir des places internes de communication de l'eBPN. L'iBPN n'a pas à modéliser les protocoles de communication, ni les CTA et CTP externes du CC. La description exhaustive du comportement d'un CO (BPN complet) n'est donc faite que lorsque nous composons l'eBPN et l'iBPN du CO. Cette composition se fait par fusion des places de communication interne de l'eBPN et de l'iBPN et par suppression des traitements supposés de l'eBPN. L'identification des places internes de communication est établie grâce aux relations <<in>> et <<out>> vues en § II.2.5.

Figure III-16, nous donnons une représentation du couplage entre l'iBPN et l'eBPN d'un CO terminal (ces diagrammes à RdP sont représentés en gris sur la figure). La partie interne d'un CO terminal est réalisée par un objet (au sens UML), nommé par convention *« Imp_NomDuCO »*. Cette représentation en Figure III-16 n'est donnée qu'à titre explicatif et n'est naturellement pas une représentation UML. Nous représentons seulement ici les places de communications externes et internes des diagrammes à RdP du CO1.

Figure III-17, la représentation de la décomposition d'un CO en deux sous-CO est donnée. Étant donc non terminal, ce CO n'a pas d'iBPN à proprement parler. La représentation du

contrôle interne est déléguée aux deux sous-CO. Néanmoins, une représentation abstraite du comportement interne de ce CO peut être effectuée par composition des eBPN des deux sous-CO.

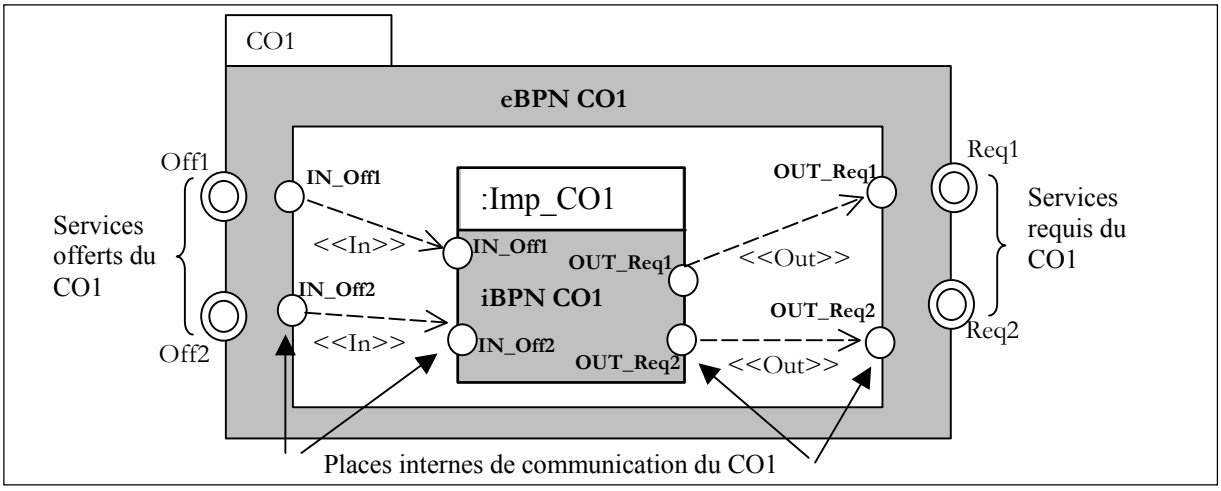


**Figure III-16 : Représentation de la décomposition d'un CO terminal**

Nous reviendrons sur ce principe de composition (et de décomposition) des diagrammes à RdP de comportement en partie V de ce mémoire.

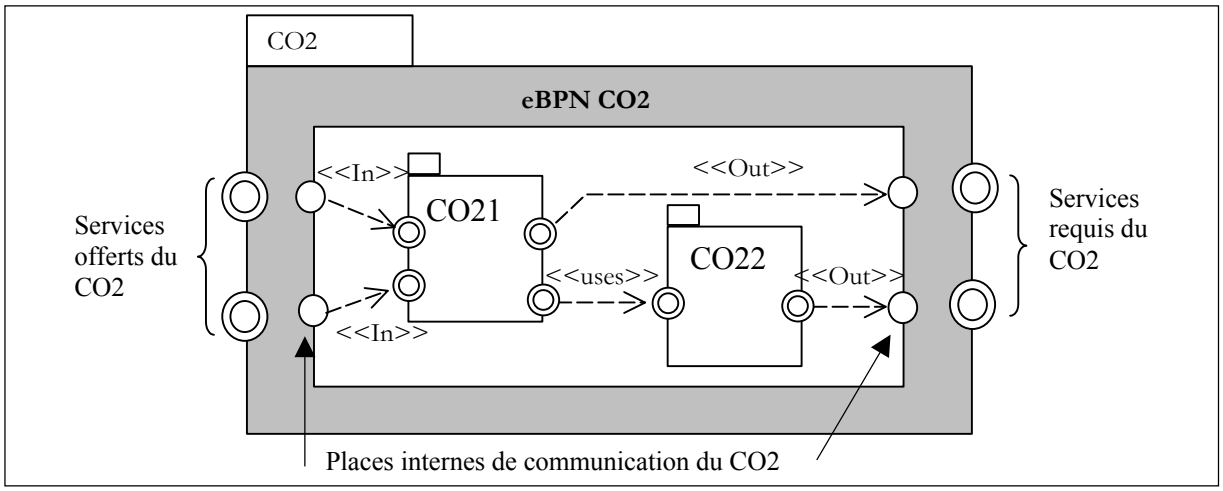


**Figure III-17 : Représentation de la décomposition d'un CO en 2 sous-CO**

### III.3.1.2. L'iBPN du CO *S_Joystick*

Le CO *S_Joystick* est considéré comme terminal, sa partie interne est alors modélisée par un objet actif UML.

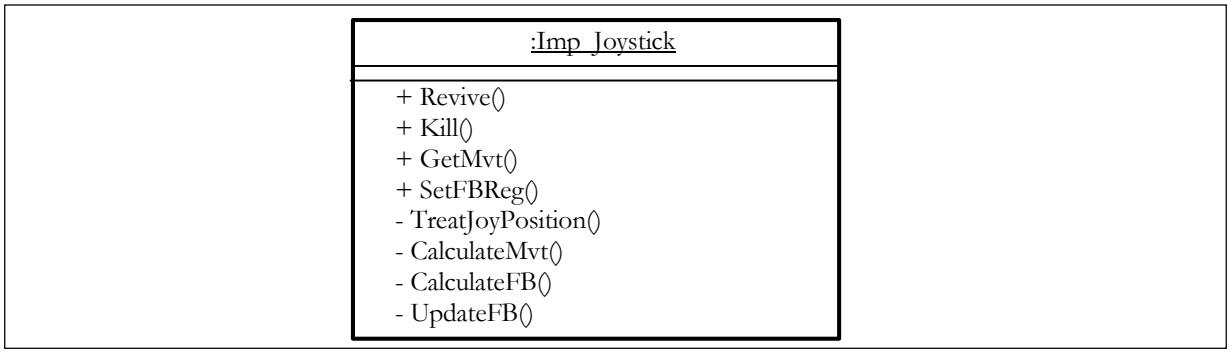


**Figure III-18 : Représentation UML de l'objet *Imp_Joystick***

La Figure III-18 fournit une représentation UML des opérations de cet objet (*Imp_Joystick*). Les opérations précédées du signe + sont des opérations publiques et faisant partie du contrat de *S_Joystick*. Les opérations précédées du signe – sont des opérations privées, internes au CO.

Cet objet *Imp_Joystick* doit naturellement réaliser tous les services offerts par le CO *S_Joystick*. La Figure III-19 est la représentation schématique d'une partie du comportement interne (iBPN) de cet objet. En se reportant à l'eBPN de la Figure III-15, les mêmes places IN et OUT (par exemple *I_JoyPosI*, *O_GetJoyPosI*) sont retrouvées.

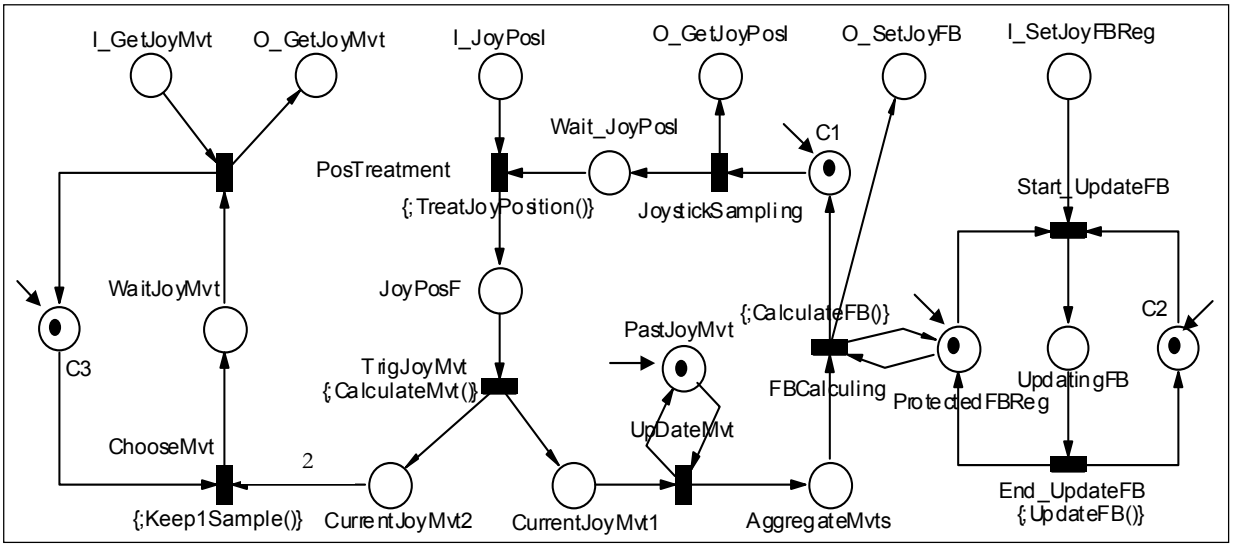


**Figure III-19 : iBPN simplifié de *S_Joystick*, partie nominale incomplète**

Dans cette Figure III-19, nous supposons que l'objet joystick vient d'être initialisé. Les principaux traitements de cet iBPN concernent l'échantillonnage de la position du joystick et l'élaboration de son retour de force.

L'échantillonnage est composé de trois étapes. La première consiste à capturer la position physique du joystick (transition *JoystickSampling*) en faisant la demande de la position du joystick à un dispositif externe (place *O_GetJoyPosI*). La seconde étape convertit cette mesure en une mesure manipulable par l'objet joystick (transitions *PosTreatment*). Notons que l'iBPN indique alors quelle opération interne de l'objet est appelée (ici l'opération *TreatJoyPosition*). Enfin, la troisième étape calcule en fonction de cette position le mouvement du Joystick (opération *CalculateMvt*).

Le calcul du retour de force (*UpDateMvt*, *AggregMvt*, *FBCalculing*) est détaillé dans les paragraphes suivants.

Afin d'augmenter la lisibilité du BPN, il est possible de ne représenter que des vues de l'iBPN. Nous parlons alors de « parties du BPN ». Ce principe de partie permet une représentation plus lisible du BPN. Un BPN peut donc avoir plusieurs parties, mais nous conseillons de ne pas en multiplier le nombre. Une partie « d'initialisation et d'arrêt » et une partie « nominale » sont généralement suffisantes. Dans ce cas, il faut bien sûr indiquer que nous ne représentons qu'un sous-ensemble de places et de transitions. Cela se fait en utilisant de petites flèches noires entrantes ou sortantes des places ou des transitions qui indiquent qu'il y a d'autres transitions (ou arcs de transition) liées à ces places (ou transitions) (cf. Figure III-19 par exemple sur la place *C1).*

## III.3.2. Représentation des comportements périodiques

L'échantillonnage de la position du joystick doit être effectué toutes les 10 ms (avec une gigue de tolérance de 1 ms). Ce comportement périodique est déjà représenté par un ensemble de places et de transitions formant un cycle. Néanmoins, rien ne permet d'affirmer que ce cycle est

bien effectué toutes les 10 ms car il dépend uniquement du temps de traitement des différentes opérations de cet échantillonnage.

L'analyste-concepteur doit donc pouvoir avoir le choix de représenter explicitement cet aspect de la dynamique de l'objet. Plusieurs modélisations des comportements périodiques sont possibles avec le formalisme des RdP (cf. Figure III-20).

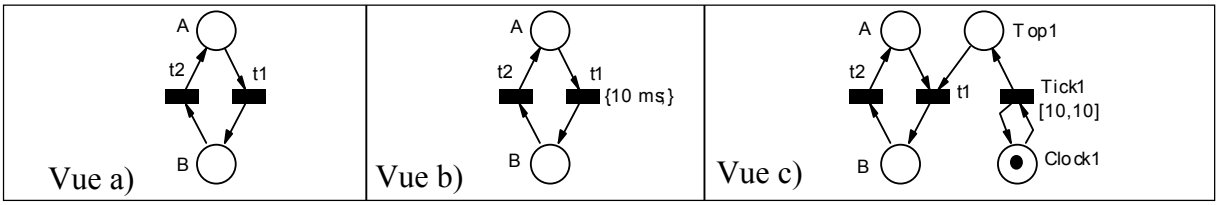


**Figure III-20 : Différentes représentations d'un comportement périodique**

Dans le cas a) de la Figure III-20, la périodicité est implicite (nous sommes dans la même situation que celle du Joystick donné en Figure III-19). En vue b), la synchronisation est modélisée par un événement externe (un top horloge arrivant ici toutes les 10 ms). Cependant, cette supposition d'une horloge physique externe à l'objet est peu adaptée à des modélisations en phase de spécification ou de conception préliminaire : elle est trop proche des détails de l'implémentation. Pour cette raison, c'est la troisième représentation (Figure III-20.c) que nous utilisons dans UML/PNO. Pour désigner l'ensemble des places (*Clock1*, *Top1*) et de la transition *Tick1*, nous parlons des places et transitions horloges.

En Figure III-21, la représentation du traitement périodique du Joystick a été ajoutée à son iBPN.

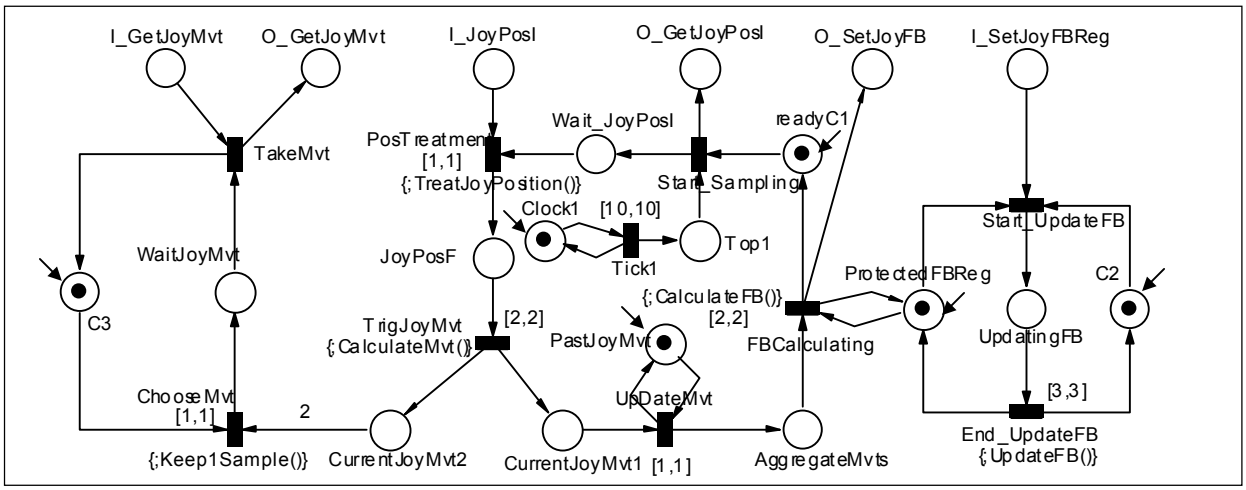


**Figure III-21 : iBPN *S_Joystick*, partie nominale**

Cette représentation du comportement périodique peut aussi être utilisée dans les diagrammes de comportements externes.

## III.3.3. Initialisation et arrêt du contrôle

Dans UML/PNO, nous distinguons deux étapes lors de la phase de création d'un CO :

- La première, dite d'instanciation, correspond à la mise en mémoire des entités. Cette étape se retrouve dans toute application informatique et n'est pas spécifique au temps réel.
- La seconde, dite d'initialisation, correspond à la mise en place de la structure de contrôle. Cette étape est spécifique à des applications multitâches et temps réels. Du fait du parallélisme, un ordre doit être respecté dans l'initialisation de la structure de contrôle[41] afin de s'assurer

[41] Qui correspond, par exemple, au niveau de la conception détaillée, à la mise en place des mécanismes de synchronisation entre objets de bas niveaux (tâche, sémaphore, boite aux lettres,…).

qu'une partie de l'application ne démarre pas et ne fasse appel à des entités qui ne sont pas encore créées ou prêtes[42].

Nous considérons généralement la première étape d'instanciation comme implicite et n'ayant pas à être modélisée. En effet, dans un système temps réel, l'allocation dynamique est généralement proscrite du fait de l'indéterminisme temporel qu'elle peut introduire. Toutes les allocations de CO sont considérées comme statiques dans UML/PNO.

Par contre et bien qu'UML/PNO ne couvre pas complètement la phase de conception détaillée où la problématique de l'initialisation de la structure du contrôle doit être complètement traitée, nous avons choisi de permettre sa représentation. En effet, cette description est parfois abordée en phase de spécification lorsque par exemple, des dispositifs physiques nécessitant des procédures précises de démarrage et d'arrêt doivent être modélisés.

De manière symétrique à la création, nous distinguons deux étapes de destruction d'un objet, l'étape d'arrêt des fils de contrôle de l'objet et l'étape de destruction de l'objet de la mémoire du système.

Pour ces raisons, l'iBPN possède une place nommée « *dead* » modélisant le fait que l'objet est en mémoire, mais dont sa structure de contrôle n'est pas encore active. L'étape d'initialisation est représentée par une opération publique nommée « *revive* ». Symétriquement, une opération « *kill* » modélise l'étape d'arrêt de la structure de contrôle du CO. De ce fait, dans un iBPN, l'état avant l'initialisation et avant la destruction de l'objet est représenté par la même place « *dead* ».

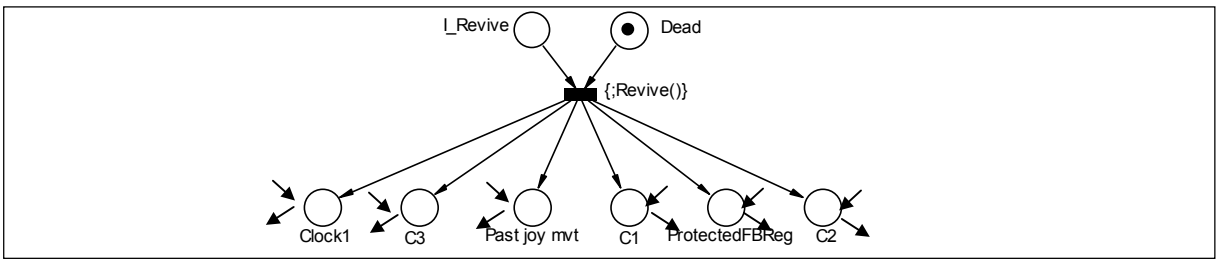


**Figure III-22 : iBPN de *S_Joystick*, partie initialisation**

La Figure III-22 représente la partie « initialisation » de l'iBPN de l'objet Joystick. Dans l'état *dead,* la structure de contrôle de l'objet n'a pas d'existence. L'opération *revive()* modélise la création de cette structure de contrôle. Dans l'exemple étudié, l'initialisation consiste à démarrer les cycles d'activités (rôle des places *C1, C2* et *C3*), l'horloge et à initialiser les données. Bien entendu, les places *C1*, *C2, C3, PastJoyMvt*, *ProtectedFBreg, Clock1* sont les mêmes que celles de la Figure III-19.

## III.3.4. Représentation des données

Nous ne représentons sur l'iBPN que les états des données ayant une influence sur le comportement de l'objet, c'est-à-dire sur les fils de contrôle de l'objet et le séquencement des opérations. Ces états sont associés à des places et à des jetons du RdP. La transformation de l'état d'une donnée est modélisée par la consommation d'un jeton et la production d'un jeton ou par le changement d'un attribut du jeton si celui-ci contient une structure de données. Pour chaque utilisation différente d'une donnée, il y a une place associée signifiant que la donnée est prête pour cette utilisation.

Nous nous appuierons sur l'iBPN de la Figure III-21 pour illustrer quatre situations fréquemment rencontrées lors de la modélisation faisant intervenir les données (Stockage des

[42] En adaptant un paradigme de modélisation asynchrone, nous ne pouvons considérer comme instantanées ces activités d'initialisation et d'arrêt de la structure de contrôle.

données, échantillonnage et accès exclusif, aspect contrôle porté sur les données et demandant l'utilisation de RdP de haut niveau).

### III.3.4.1. Stockage des données

Il est parfois nécessaire de stocker plusieurs valeurs d'une même donnée et ce, afin de conserver leur évolution au cours du temps. Ce stockage peut être représenté par une accumulation de jetons dans une place associée à un état d'une donnée. Un jeton représente alors la valeur d'une donnée à un instant. Mais il peut être choisi de représenter ces différentes instances de données par différentes places. Ce choix est en général effectué quand la conservation des données a une influence sur le contrôle de l'objet (généralement pour l'initialisation de ces données).

Ainsi, si nous détaillons le calcul du retour de force, nous voyons que ce calcul (*CalculateFB*) nécessite les deux derniers mouvements du joystick (place *AggregateMvt*), ainsi qu'une consigne de régulation (*ProtectedFBReg*), afin d'élaborer le retour de force à appliquer au joystick. Afin de représenter le fait que nous devions prendre le précédent mouvement, la place *PastJoyMvt* contient déjà un jeton. La présence de ce jeton se justifie essentiellement pour la représentation de l'initialisation et modélise l'attribution d'une valeur initiale et réaliste à la position précédente du joystick avant tout premier calcul.

### III.3.4.2. Échantillonnage des données

Concernant l'utilisation du mouvement du joystick par d'autres objets du système, il se trouve que ces derniers n'ont pas besoin de tous les mouvements. En effet, la fréquence d'échantillonnage du joystick est rapide afin d'élaborer un retour de force réaliste. Or, les autres objets du système qui ont besoin du mouvement du Joystick, ne travaillent pas à cette fréquence et n'utiliseront donc qu'une partie de l'échantillonnage. Une politique doit alors être spécifiée afin de préciser quel échantillon sera conservé. Différentes politiques existent (moyenne des échantillons, choix d'un seul échantillon…). Ici (cf. Figure III-21), cette politique est représentée par les places (*CurrentJoyMvt2*, *WaitJoyMvt, C3*) et la transition *ChooseMvt*. Dans notre exemple, nous ne savons pas exactement quelle politique est choisie, nous savons seulement que tous les 2 échantillons (poids de l'arc entrant), un échantillon est conservé dans la place *WaitJoyMvt*. C'est l'opération *Keep1Sample* qui stipulera le contenu de l'échantillon à conserver.

### III.3.4.3. Accès exclusif d'une donnée

Comme nous l'avons vu, le calcul du retour de force nécessite, outre les deux derniers mouvements du joystick, une consigne. Cette dernière est représentative d'un autre type d'utilisation d'une donnée (accès protégé ou exclusif d'une variable). Cette consigne peut être modifiée à tout moment[43] afin de modifier la réactivité du joystick (le fait que celui-ci sera plus ou moins "difficile" à déplacer). En effet, suivant la phase de vol, le joystick peut se rigidifier afin que le pilote ne puisse pas demander de trop grandes amplitudes et, par-là, ordonner des mouvements dangereux pour l'avion. Naturellement, cette consigne ne peut être modifiée au cours de l'élaboration du retour de force du joystick. Cet accès protégé de la consigne de régulation est modélisée (cf. Figure III-21) par la place *ProtectedFBReg*. C'est la technique classique de représentation du partage de ressources par RdP. Cette place est partagée avec le fil de contrôle de modification de la consigne (places *C2*, *UpdatingFB* et transitions *Start_UpdateFB*, *EndUpdateFB*)

### III.3.4.4. Utilisation des RdP de haut niveau

Les RdP de haut niveau doivent parfois être utilisés pour deux raisons :

- Afin d'améliorer la lisibilité du modèle. Cela permet en effet de replier des parties

[43] Sauf lorsqu'elle est en cours de lecture.

identiques du RdP et, dès lors, d'obtenir une représentation plus simple du RdP.

- Afin d'enrichir l'aspect contrôle par l'aspect donnée. Généralement, plus le degré d'abstraction de la modélisation baisse et donc, plus nous entrons dans des phases de conception, plus la description du contrôle et des données doit être fine[44].

Dans ce cas, des variables sont associées aux jetons. Ces variables sont susceptibles de conditionner le franchissement d'une transition. Nous indiquons le type de jetons (avec la notation <Type de jeton>) que peut accueillir la place. Une place ne peut traiter qu'un type de jeton.

Ce mécanisme d'abstraction doit être manié avec précaution[45], car en faisant porter une partie du contrôle sur les données, nous perdons les possibilités de vérification du RdP. Il faut donc veiller à ce que ce contrôle porté par les données, ne remette pas en cause le comportement général de l'objet. Ou bien, il faut s'assurer qu'il est toujours possible de déplier le RdP afin de représenter explicitement ce contrôle.

A la Figure III-23, nous avons représenté différentes manipulations des jetons (et des données portées par ceux-ci). Cet ajout de l'aspect donnée n'a pas remis en cause le comportement de l'objet joystick (et donc les possibilités de vérification). Par exemple, détaillons le traitement de l'échantillonnage de la position du joystick. La place I_*JoyPosI* reçoit des jetons ayant comme structure de données <*Joy_Pos_I*>. Les actions liées à la transition permettent alors de préciser comment est transformé le jeton <X> en <Y>.

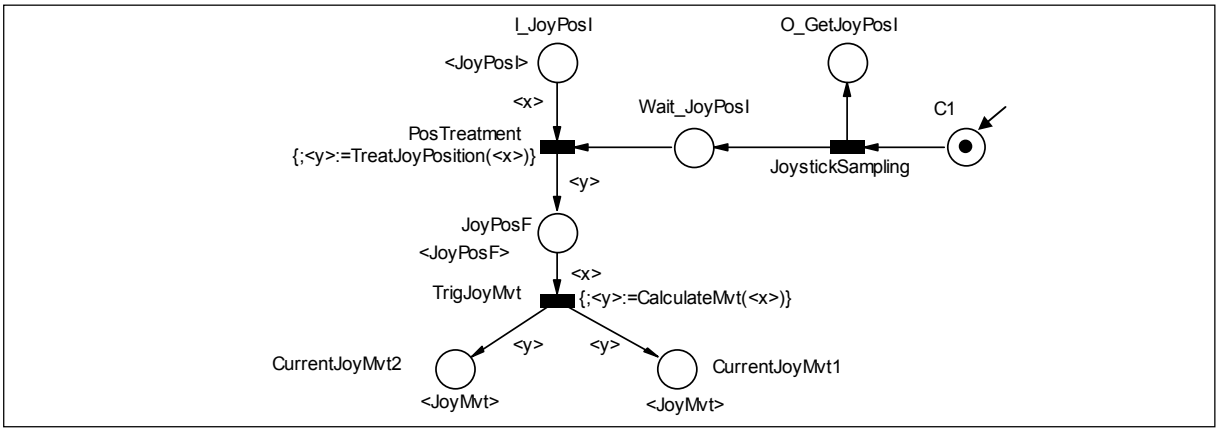


**Figure III-23 : Partie de l'iBPN de *S_Joystick* avec un RdP de haut niveau**

Précisons que la représentation des données, en utilisant les RdP de haut niveau, est aussi possible sur les eBPN. Dans ce cas, cela sert surtout à préciser les types des arguments passés par les opérations de l'eBPN.

---

[44] Cela permet, sans complexifier inutilement le RdP, de pouvoir le simuler et faciliter le passage à l'implémentation en explicitant les détails sur la transformation des données (des règles ont d'ailleurs été définies pour permettre un passage direct à un langage d'implémentation en Ada 83 [Paludetto-1991] ou en C++ [Paludetto-1993]).

[45] Néanmoins, il arrive qu'il ne puisse être évité. En effet, des données peuvent conditionner le comportement du système. Ces données peuvent être le résultat d'un calcul ou provenir de l'environnement et, suivant leurs valeurs, demander l'exécution de différentes activités. Par exemple, imaginons la lecture d'un capteur indiquant la disponibilité d'un tapis roulant qui conditionne le transport d'une marchandise sur l'un des tapis libres. Ainsi, au niveau de la description du comportement de l'objet, nous aurons un indéterminisme (possibilité de l'utilisation de chacun des tapis) qui ne peut être levé que par une information (quel tapis est libre) et n'appartenant pas à l'aspect "contrôle".

Naturellement, l'utilisation de RdP de haut niveau se fait en enrichissant la définition de la classe à places et transitions temporelles donnée en § III.1.4.2. Cet enrichissement reste simple et ne rentre pas en contradiction avec les règles, déjà définies, de franchissement des transitions et d'états du RdP. Par contre, il n'est pas possible, en utilisant les RdP de haut niveau, de poser des pré-conditions ou des actions lors du franchissement des transitions sur les dates de production et de consommation du jeton dans les places, et de manière plus générale, d'utiliser la syntaxe des RdP de haut niveau afin de définir des contraintes temporelles.

# III.4. Résumé sur la dynamique dans UML/PNO

Nous avons montré comment nous formalisons la représentation du comportement des CO (et CC) en utilisant les réseaux de Petri. Pour cela, nous avons proposé un nouveau diagramme UML (les diagrammes à RdP) dédié à la représentation de l'aspect dynamique du système. Une nouvelle classe de RdP, les RdP à places temporelles et transitions stochastiques temporelles, a été définie.

Cette classe de RdP a été introduite afin de permettre une représentation distincte de deux types de contraintes temporelles :

- Les contraintes temporelles attendues (CTA), qui modélisent des hypothèses temporelles à vérifier et qui sont portées sur des places du RdP.
- Les contraintes temporelles posées (CTP), qui modélisent des relations d'ordre supposées et qui sont portées sur des transitions du RdP.

Grâce au pouvoir de description des RdP et aux informations temporelles supplémentaires apportées par les places et transitions temporelles, nous disposons d'un grand pouvoir d'expression des contraintes temporelles. Par ailleurs, nous avons défini une syntaxe afin que ces CT puissent être présentes sous forme de notes UML dans les diagrammes UML.

Dans la définition des diagrammes à RdP, nous avons conservé la séparation entre vue interne et vue externe d'un CC (cf. § II.2.3).

Deux types de diagrammes à RdP ont donc été définis :

- Le diagramme à RdP de comportement externe (eBPN) qui modélise la vue externe d'un CC et complète la description du contrat du CC.
- Le diagramme à RdP de comportement interne (iBPN) qui est une représentation de la structure de contrôle interne du CC.

Le comportement complet d'un CC (son BPN) est décrit par la composition de son eBPN et de son iBPN. Cette description distincte de la dynamique sert plusieurs de nos objectifs :

- Le premier est de toujours respecter le principe d'encapsulation. Un CO peut être utilisé par d'autres CO alors que ceux-ci ne connaissent que la partie externe du CO. Cela est possible car nous avons restreint les possibilités de communications et de visibilités des sous-CO. L'utilisation d'un CO se fait toujours par invocation des opérations qu'il offre. De ce fait, des CO peuvent utiliser un CO dont seule la partie externe est définie.
- Le second était de se rapprocher des concepts de ports et de protocole de la méthode ROOM [Selic-1994] et RT-UML[Selic-1998]. En effet, dans cette approche, la modélisation des communications entre entités est séparée de la description de leur comportement interne. Cela offre une plus grande souplesse dans la description des collaborations entre entités et permet de se resservir d'un même protocole de communication avec différentes entités. Mais dans ROOM, le protocole correspond à une description d'un ordre séquentiel des messages échangés, sans CT explicite, alors que dans UML/PNO, ce protocole est décrit par un eBPN et a donc un pouvoir d'expression plus grand.
- Le troisième concerne la gestion des contraintes temporelles et, de manière plus générale,

la gestion de la dynamique du système. Cela vient en grande partie du principe de modélisation des eBPN. Grâce à la notion de traitement supposé, les relations d'ordre (qualitatives et quantitatives) entre invocations et requêtes des opérations entre CO sont capturées. Cela permet à un analyste-concepteur de définir des collaborations entre CO à un niveau de composition donné et de définir des CT, sans se préoccuper de leur réalisation. A ce niveau d'abstraction, l'analyste-concepteur peut alors s'assurer de la cohérence des CT qu'il vient de définir, en se basant sur les eBPN de chacun des CO. A charge à la partie interne du CO (ou des sous-CO le composant) de réaliser ces CT. Nous reviendrons sur ce principe de composition et de décomposition lorsque nous détaillerons les techniques de vérification des CT en partie V.

Auparavant, dans la partie IV qui suit, nous allons présenter les règles de dérivation de certains diagrammes UML permettant de générer des parties de diagramme à RdP.

# Partie IV Dérivation et validation partielles des diagrammes UML dynamiques



Le procédé de dérivation des diagrammes dynamiques UML vers les diagrammes à RdP est présenté dans cette partie. Les principes généraux de cette dérivation sont tout d'abord donnés. Nous motivons ensuite nos choix pour la dérivation d'une partie des diagrammes UML.

L'application des règles de passage des diagrammes d'interaction et des diagrammes d'activités est ensuite illustrée sur un exemple du simulateur.

Ce processus de dérivation, outre la traduction des diagrammes UML vers les diagrammes à RdP, apporte une assistance lors de la validation de la modélisation. En effet, au cours et à la fin de la dérivation, tout un ensemble d'aides est proposé à l'analyste-concepteur afin de vérifier une partie de la cohérence de la syntaxe et de la sémantique des diagrammes UML dérivés. Ces techniques sont alors présentées.

# IV.1. La dérivation des diagrammes UML

## IV.1.1. Principes généraux de la dérivation UML/PNO

Le terme de dérivation des diagrammes UML désigne ici un processus de génération des diagrammes à RdP à partir de certains diagrammes UML utilisés pour représenter la dynamique du système. Dans la première partie de ce mémoire, nous avons motivé nos choix pour un processus de dérivation semi-automatique basé sur le modèle UML (et non sur son méta-modèle).

Nous avons choisi de restreindre le moins possible la sémantique et la syntaxe des diagrammes UML dérivés. En effet, nous tenions à ne pas dénaturer l'un des rôles de la notation UML qui est de capturer les besoins préliminaires (généralement vagues et exprimés de manière informelle) et de les traduire en une spécification semi-formelle (nécessairement ambiguë et imprécise). Or, en restreignant la syntaxe et la sémantique d'UML, il y a risque de rendre son utilisation trop contraignante (cf. § I. I.2.6.2.b).

Nous pensons que ce principe est applicable si un processus de dérivation semi-automatique est employé. C'est grâce aux questions posées à l'analyste-concepteur, lors de la dérivation des diagrammes UML, que les ambiguïtés des diagrammes UML peuvent être levées (et les informations manquantes complétées). C'est d'ailleurs la principale différence entre UML/PNO et d'autres approches existantes de dérivation des diagrammes UML vers les RdP.

## IV.1.2. Diagrammes UML traités et non traités

Ni les diagrammes statiques UML (diagrammes de classes et d'objets, cas d'utilisation), ni les diagrammes dédiés aux phases de conception détaillée (diagrammes de déploiement et de composants) ne sont traités puisque notre travail porte sur l'aspect dynamique de la modélisation de STR dans les premières phases du développement.

Nous nous sommes principalement intéressés à la dérivation des représentations UML de la dynamique du système, à savoir les diagrammes d'interaction (diagrammes de séquence et de collaboration) et les diagrammes d'activités.

Par ailleurs, les diagrammes d'états/transitions ne sont pas traduits. En effet, il existe déjà un grand nombre de travaux et de propositions sur leur dérivation vers les RdP (toutes les approches recensées au § I.3.3.3). Certains sont directement applicables à UML/PNO.

Cependant, ces techniques de dérivation des diagrammes d'états/transitions sont soit très simples, soit complexes. Les caractéristiques sont les suivantes :

- Cas simples : les états et leurs transitions sont directement traduits en RdP respectivement par des places et des transitions [Baresi-2001b]. De nombreuses restrictions à la syntaxe des diagrammes d'états/transitions (pas de garde, d'état orthogonal, de super état ou d'historique….) sont alors introduites.
- Cas complexes : peu de restrictions sont faites sur la syntaxe (à part l'historique qui semble n'être jamais traité). Toutefois, les diagrammes d'états/transitions doivent d'abord être transformés (implicitement ou explicitement) en une machine à états sans hiérarchie, ni état concurrent, ni action d'état interne (*entry*, *do*, *exit*).

Dans le premier cas, les restrictions sont, à notre sens, trop contraignantes et l'utilisation des diagrammes d'états/transitions devient très limitée. Nous préférons, dans ce contexte, utiliser les diagrammes d'activités qui ont alors un plus grand pouvoir d'expression.

Dans le second cas, les RdP générés deviennent extrêmement complexes [Huszerl-2002]

[Bondavalli-1999]. La mise à plat du diagramme d'états/transitions mène à l'apparition d'un très grand nombre d'états. Les auteurs de ce type de dérivation utilisent le RdP généré uniquement à des fins d'analyse, mais pas de modélisation.

Dans notre recherche de règles de dérivation, nous avons été confrontés au même constat. Nous n'avons pas trouvé de technique offrant un compromis satisfaisant entre une restriction peu conséquente de leur syntaxe et une bonne lisibilité des RdP générés.

Enfin, nous avons émis des réserves concernant l'utilisation des diagrammes d'états/transitions (cf. §I.2.3.2) pour la modélisation de comportements dynamiques complexes, en présence de processus fortement parallèles et manipulant des CT explicites. Leur lecture devient alors difficile et leur sémantique peut introduire des risques de mauvaise interprétation par un non-spécialiste. Dans cette situation, nous pensons que les RdP sont plus adaptés ; leur syntaxe et lisibilité peuvent, certes, être considérées comme tout aussi complexes, mais ils présentent moins de risques de mauvaise interprétation tout en offrant une syntaxe beaucoup plus riche (en particulier pour les CT). De plus, comme il a été montré dans [Eshuis-2002], les RdP ont un plus grand pouvoir d'expression de la concurrence que les diagrammes d'états/transitions (certaines modélisations par RdP ne sont pas représentables par les diagrammes d'états/transitions). En outre, notre processus de dérivation accompagne l'analyste-concepteur dans sa modélisation et lui apporte donc une meilleure compréhension des diagrammes à RdP et, nous l'espérons, une plus grande facilité de manipulation de ce formalisme.

Pour toutes ces raisons, l'utilisation des diagrammes d'activités dans UML/PNO est préférée à celle des diagrammes d'états/transitions.

## IV.1.3. Principe du processus de dérivation

Naturellement, suivant la famille de diagrammes UML (diagrammes de séquence, de collaboration ou d'activités) qui est traduite, le processus et les règles seront différents. Néanmoins, les principes des processus de passage restent identiques et font appel à la même stratégie. Cette dernière consiste à traduire, un par un, les diagrammes UML d'une même famille et a reporter immédiatement leurs informations sur les diagrammes à RdP.

Ces informations sont regroupées en « type d'information » suivant qu'elles portent sur la communication, sur les relations d'ordre entres les activités, sur les contraintes temporelles…

Ainsi, pour un diagramme donné d'une famille donnée, ces éléments de modélisation sont traités par lots en fonction de leurs types. L'ordre de traitement des lots diffère suivant le diagramme UML dérivé. Cette stratégie permet de regrouper les questions portant sur une même thématique.

De cette façon, les diagrammes à RdP de chaque entité de la modélisation sont construits de manière incrémentale, élément de modélisation après élément de modélisation, type d'information après type d'information, diagramme après diagramme. Lors du report des informations des diagrammes UML sur les diagrammes à RdP, des incohérences avec des dérivations précédentes (de diagrammes d'une même famille ou d'une famille différente) peuvent être détectées et résolues par des questions à l'analyste-concepteur.

Cette détection des incohérences est rendue possible par la convention de nommage interne des éléments des diagrammes à RdP. Cette dernière a déjà été présentée dans la partie III de ce mémoire. Ces processus de dérivation des diagrammes de séquence ou de collaboration et des diagrammes d'activités sont maintenant illustrés sur l'exemple du simulateur SIMONA (SRS).

# IV.2. Les règles de dérivation des diagrammes UML

## IV.2.1. Dérivation des diagrammes de séquence

### IV.2.1.1. Quelques problèmes pour la dérivation des diagrammes de séquence

Dans le cadre de la dérivation des diagrammes de séquence, un des problèmes majeurs réside dans le double rôle qui leur est alloué [Simons-1998] : d'une part, modéliser une simple histoire (une exécution sans alternative) décrivant des interactions entre objets et d'autre part, modéliser un ensemble d'histoires, plus ou moins proches, dérivant généralement d'un même cas d'utilisation (et donc ayant des alternatives).

Il est donc du ressort de l'analyste-concepteur de comprendre quelle en est l'utilisation. Dans le cas d'un ensemble de diagrammes de séquence décrivant des alternatives, il a peu de moyens d'indiquer les liens entre ces différents diagrammes (à l'exception de notes textuelles sans syntaxe définie). Dans le second cas, la représentation des alternatives reste limitée à des gardes complémentaires portant exclusivement sur les invocations d'opérations (cf. Figure IV-1). De plus, l'expression de la condition est libre (sans syntaxe, ni sémantique précise) et peut faire référence à n'importe quel élément (attribut, contraintes temporelles…) de la modélisation UML.

Concernant la représentation des lignes de vie des objets, des changements sont apparus dans UML 1.3 en réponse aux critiques portées sur la version 1.1. La notion de « barre d'activation ou de contrôle (*activation bar*) » est maintenant proposée. Elle permet de représenter des activités et sous-activités internes de l'objet (*nested procedure*). Cette notation peut être étendue afin de traiter le cas d'objets actifs lorsque ceux-ci comportent plusieurs fils de contrôle. Il semblerait alors que tout appel d'opération synchrone doit être modélisé par une sous-barre d'activation (cf. Figure IV-1, invocation m4 entre l'objet 3 et l'objet 5).

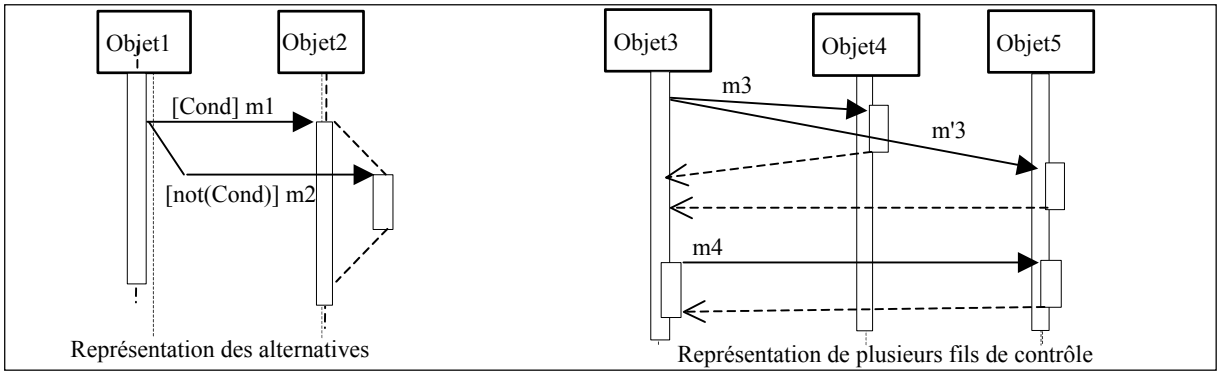


**Figure IV-1 : Exemple de diagramme de séquence**

Toutefois, cette utilisation des barres d'activation semble optionnelle et non systématique, l'analyste-concepteur pouvant revenir à une représentation où il n'est pas possible de distinguer les différents fils de contrôle. Ainsi, certains exemples graphiques donnés dans la spécification UML 1.3 ([OMG-1999b] fig 3.49 pp 3.99) restent ceux d'UML 1.1 et ne reflètent pas ces changements de syntaxe et de sémantique.

Par ailleurs, certaines imprécisions demeurent quant à la représentation du contrôle :

- Par exemple, Figure IV-1, nous ne savons pas comment différencier les attentes de l'objet 3 entre l'envoi de deux messages simultanés (m3 et m'3). Faut-il plutôt faire figurer une barre d'activation pour chaque appel m3 et m'3 sur l'objet3, même si ces appels sont faits dans un même fil de contrôle ?
- La représentation des communications avec attente limitée n'est pas clairement définie

dans UML 1.3. Une extension a été proposée mais, dans ce cas, l'alternative (cas où le délai d'attente est expiré) doit être exprimée sur un autre diagramme de séquence.

- Enfin, l'ordre des messages est un ordre partiel et il n'est pas possible, comme dans les *Messages Sequence Charts*[46] [ITU-TS-1996], d'indiquer s'il y a un ordre total ou non entre les messages.

Ces quelques exemples sont représentatifs des ambiguïtés et des imprécisions des diagrammes de séquence. Celles-ci devront être levées lors de la dérivation par des questions explicites à l'analyste-concepteur.

### IV.2.1.2. Principes généraux de la dérivation UML/PNO des diagrammes de séquence

Le principe UML/NPO de report des informations du diagramme de séquence sur des réseaux de Petri est donné Figure IV-2. Ce sont essentiellement les informations sur les communications entre objets qui sont traduites, ainsi que l'ordre de ces communications. Un RdP est dédié à la représentation de chaque CO (ici, CO1 et CO2). Les communications entre CO sont représentées par des places partagées.

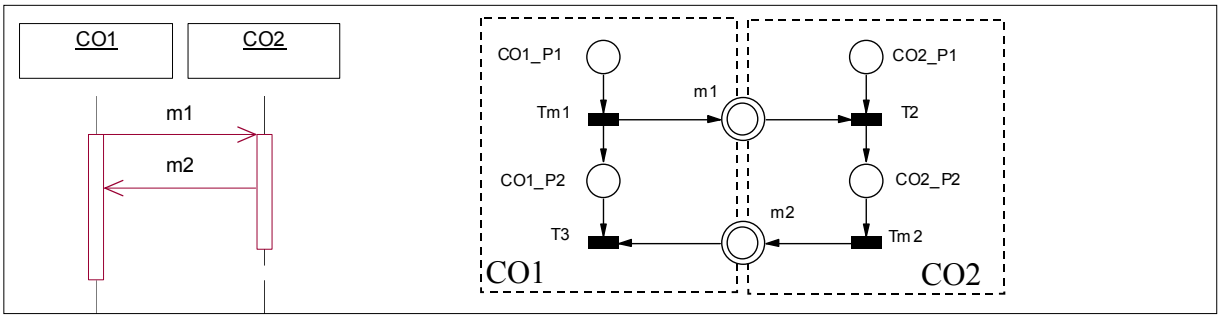


**Figure IV-2 : Principe de « traduction » des diagrammes de séquence**

Un diagramme de séquence se focalise essentiellement sur la description des interactions entre les objets. C'est donc plutôt le comportement externe de ces objets qui est décrit et non leur fonctionnement interne. Par conséquent, le diagramme à RdP que nous pouvons générer (ou compléter) à partir de diagrammes de séquence est un diagramme à RdP de comportement externe (eBPN).

### IV.2.1.3. Différences avec des approches similaires de dérivation

Des travaux similaires [Gehrke-1999] [King-1999] ont été initiés par d'autres équipes de recherche. Ils sont basés sur le même principe de traduction que celui donné ci-dessus. La plupart de ces travaux redéfinissent (explicitement ou implicitement) la sémantique des diagrammes de séquence afin de retrouver celle des *Message Sequence Chart*. Dès lors, ils peuvent s'appuyer sur la grande quantité de travaux existants [Heymer-2000]. Pour notre part, nous avons choisi de rester au plus proche de la sémantique semi-formelle des diagrammes de séquence.

Par conséquent, l'ordre des messages donné par le diagramme de séquence n'est pas considéré comme un ordre total. Nous traitons la représentation des contraintes temporelles et prenons en compte les spécificités des diagrammes de séquence UML 1.3 (telles que les barres d'activation). Naturellement, les concepts UML/PNO tels que les différences entre CTA et CTP et le comportement externe et interne d'un objet restent spécifiques à notre démarche.

[46] Dans la prochaine version d'UML (2.0) et aux vues des modifications proposées, il semblerait que les *Messages Sequence Charts* se substituent aux diagrammes de séquence. Cependant, ces propositions n'étant pas encore adoptées, nous avons travaillé sur le formalisme actuel des diagrammes de séquence.

### IV.2.1.4. Processus de dérivation des diagrammes de séquence

Chacun des CO actifs du diagramme est traité en reportant directement les informations sur son RdP de comportement externe. Pour chaque ligne de vie du diagramme de séquence, quatre types d'informations sont distingués suivant qu'ils se rapportent :

1. Aux communications entrantes et sortantes,
2. Aux effets observables des invocations,
3. Aux contraintes temporelles,
4. Aux alternatives et boucles.

Chaque type d'information sera traité dans l'ordre donné ci-dessus lors du processus de dérivation et reporté sur l'eBPN du CO correspondant.

### IV.2.1.5. Illustration sur le simulateur SIMONA

Au lieu de donner l'intégralité des règles de dérivation des diagrammes de séquence, nous avons préféré illustrer le principe de leur application sur l'exemple du simulateur. En effet, en partie III, un premier ensemble de règles a déjà été fourni lors de la présentation des eBPN. De plus, le lecteur intéressé trouvera un descriptif complet de ces règles en annexe.

Nous allons donc montrer le principe du processus de dérivation sur le SRS global. Nous nous focaliserons ensuite sur la génération des diagrammes à RdP pour deux entités de la modélisation non encore présentées : les acteurs et les objets médium.

#### IV.2.1.5.a) Application du processus de dérivation sur le SRS global

Le diagramme de séquence donné en Figure IV-3 est représentatif des diagrammes établis en début de modélisation. Les protocoles de communication ne sont pas encore définis et seules sont identifiées les informations entrantes et sortantes du système(stimuli et réponses), ainsi que les CT globales en relation avec ces informations.

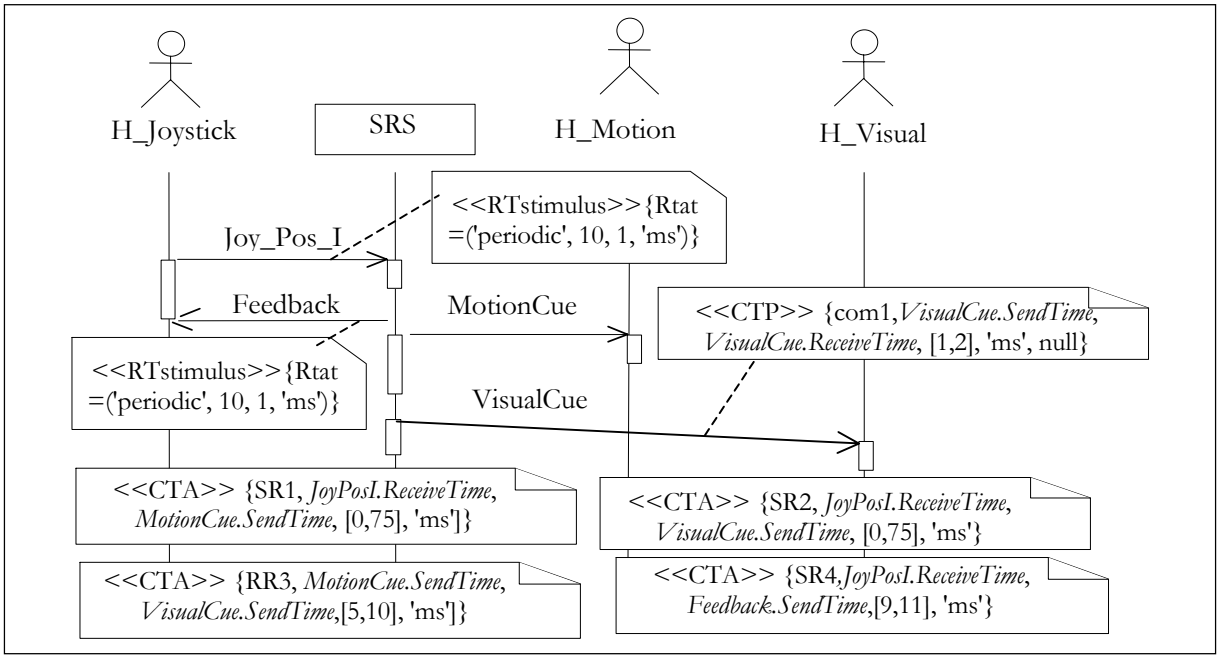


**Figure IV-3 : Quelques contraintes temporelles externes du SRS**

Sur ce diagramme, l'arrivée de la position du joystick induit la production de réponses pour la cabine hydraulique, l'écran principal du simulateur, ainsi que le retour de force sur le joystick. Des CTA sont données, elles définissent principalement les temps de réponse attendus.

Notons une CTA un peu particulière entre deux réponses ; elle spécifie un décalage (de 5 à 10 ms) entre la production des informations de déplacement de la cabine hydraulique et l'affichage. Elle est définie afin d'éviter « le mal du simulateur[47] » qui se produit si ce décalage n'est pas respecté.

*IV.2.1.5.a.i) Dérivation des communications entrantes et sortantes*

En Figure IV-4, nous avons traduit les informations des communications de la ligne de vie du CO SRS du diagramme de séquence donné en Figure IV-3. A ce stade, le protocole de communication est inconnu. Par conséquent, les appels de méthodes, le nom des places en entrée et en sortie sont aussi inconnus (préfixe *Unk* pour *unknown*). Seul le type des données peut être indiqué.

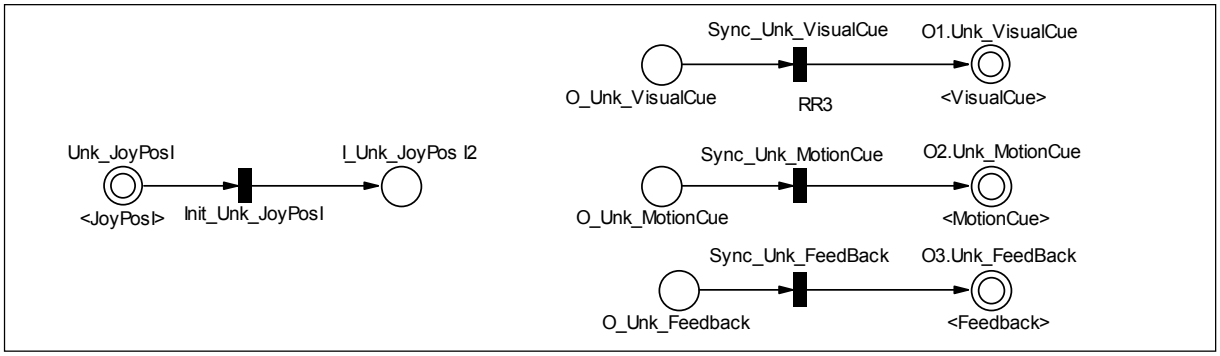


**Figure IV-4 : eBPN partiel du SRS, étape 1**

*IV.2.1.5.a.ii) Dérivation des effets observables*

Ensuite, la recherche des « effets observables » est effectuée. Il s'agit de déterminer, pour chaque requête effectuée par l'objet, s'il y a ou non, invocation de méthodes d'autres objets. Dans le cas du SRS, une demande à l'analyste-concepteur est effectuée. Suite à la confirmation de l'arrivée de la position du joystick qui déclenche la production des réponses *FeedBack*, *MotionCue* et *visualCue*, l'eBPN est complété par des traitements supposés (cf. Figure IV-5). Rappelons que pour chaque utilisation différente de la donnée *JoyPosI*, une place et un traitement différent sont représentés. Afin de simplifier le RdP, la convention de nommage des traitements supposés n'a pas été respectée ici (*ST_treat1* devrait se nommer *ST_Unk_JoyPosI->Unk_VisualCue*).

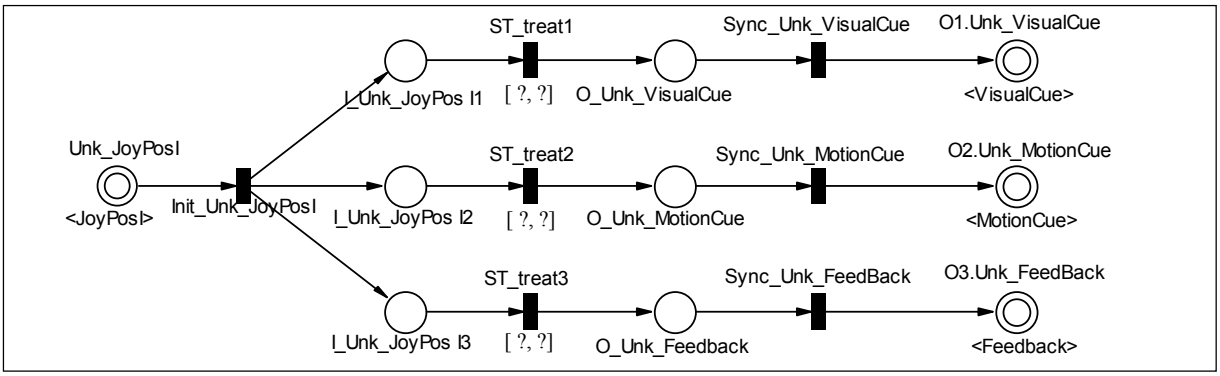


**Figure IV-5 : eBPN partiel du SRS, étape 2**

*IV.2.1.5.a.iii) Dérivation des contraintes temporelles*

Les CT sont ensuite rajoutées. Un premier traitement automatique des CTA en relation avec le SRS conduit à l'eBPN donné en Figure IV-6. Puis, les CT de périodicité[48] appliquées au SRS sont

[47] Dont les symptômes sont similaires au mal de l'air ou au mal de mer…

[48] Elles utilisent le formalisme défini dans la réponse au RFP « UML *Profile for Schedulability, Performance and Time* ».

traitées. La traduction de la note sur la contrainte périodique du retour d'effort consiste à ajouter une place horloge de traitement supposé et une CTA sur la production du retour force.

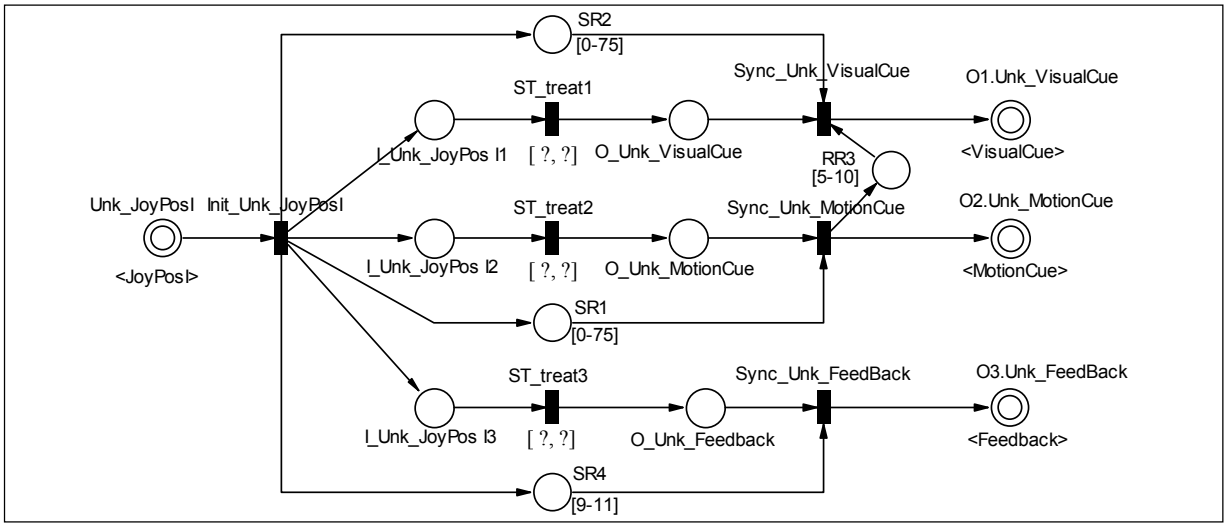


**Figure IV-6 : eBPN partiel du SRS, étape 3**

En Figure IV-7, la traduction automatique est donnée en ne montrant que la sous-partie concernée de l'eBPN. L'existence d'une autre transition (*ST_Treat3*) produisant déjà le retour de force (*FeedBack*) va être détectée lors du processus de dérivation. Une série de questions sont alors posées à l'analyste-concepteur afin de vérifier s'il s'agit du même traitement ou si celui-ci est différent. Dans le cas présent, il s'agit du même traitement et l'analyste concepteur peut choisir différentes traductions (conservation de l'horloge ou non dans les traitements supposés, fusion ou non des places CTA). Les choix effectués ici conduisent au RdP déjà présenté en Figure IV-6.

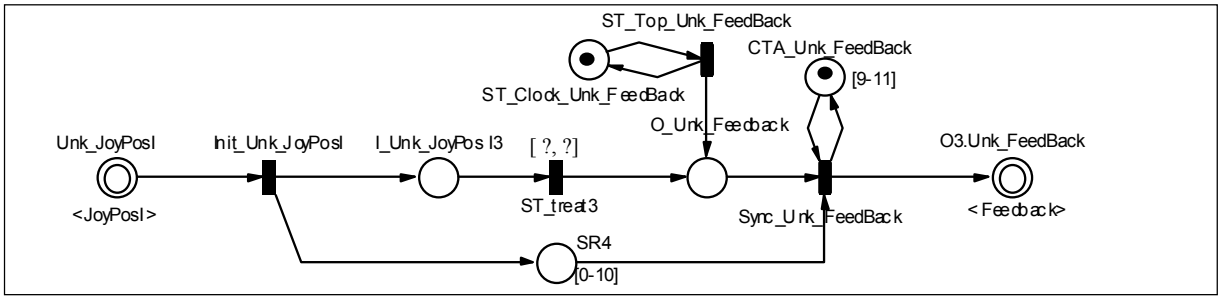


**Figure IV-7 : eBPN partiel du SRS, étape 4 temporaire**

La dérivation des diagrammes de séquence vers le comportement externe d'un CO vient d'être illustrée à l'aide de l'exemple du simulateur. Toutefois, d'autres éléments d'informations présents sur le diagramme de séquence ont été ignorés pour l'instant. Il s'agit des CT portant sur les acteurs (CT de périodicité sur le retour d'effort du joystick ou sur la position du joystick) et sur les temps de communication (du message *VisualCue*).

Nous avons choisi de représenter chacune d'elle sur des diagrammes à RdP particuliers, distincts des BPN. Ces diagrammes à RdP sont alors nommés : RdP d'environnement ou EPN (pour *Environmental Petri Net*). En effet, il n'y a pas, comme pour les RdP de comportement, de distinction entre comportement externe et interne. Ils modélisent directement tout le comportement de l'entité (objet acteur ou objet medium).

IV.2.1.5.b) Le comportement des acteurs

Notre objectif n'est pas de modéliser tout le comportement de l'environnement[49], mais

[49] Le travail de représentation exhaustif de l'environnement sort du cadre de notre mémoire et n'a pas été étudié.

uniquement les informations temporelles décrivant comment les occurrences des stimuli sont produites et comment les réponses sont attendues par l'environnement. Nous ne représentons que les informations de l'environnement qui imposent des contraintes temporelles sur le système. Ces informations nous seront utiles par la suite en analyse ou en simulation.

En fonction des CTA et CTP en relation avec l'acteur, il est proposé tout un ensemble de comportements prédéfinis (cf. annexe). L'EPN du *H_Joytisck* résultant des choix effectués est donné en Figure IV-8.

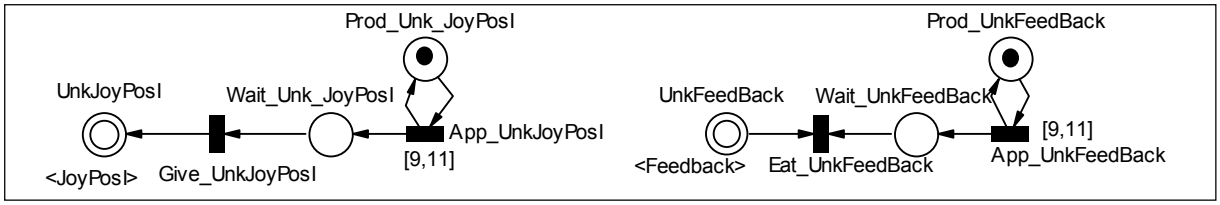


**Figure IV-8 : EPN de l'acteur *H_Joystick*, version préliminaire**

IV.2.1.5.c) Les temps de communication entre CO

Les temps de communication entre les entités de la modélisation sont représentés, lorsqu'ils ne sont pas négligeables, sur un objet médium.

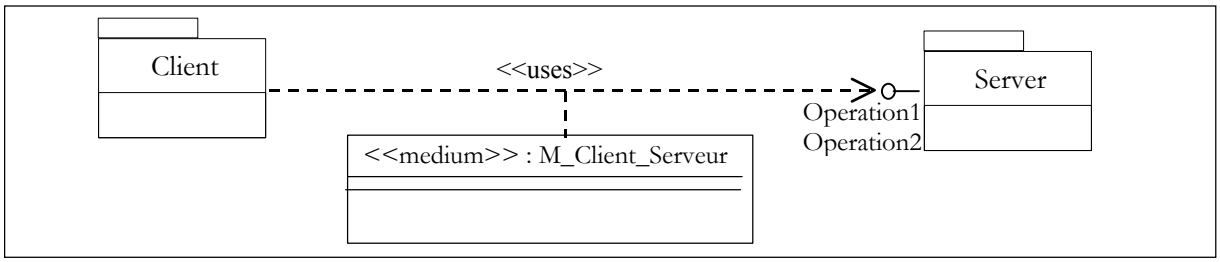


**Figure IV-9 : Représentation graphique d'un objet médium**

Cet objet est, en quelque sorte, représentatif d'une partie du médium de communication du système (d'où son nom). Sa représentation en UML est faite en définissant un objet stéréotypé <<medium>>. Ce dernier ne fait que préciser les relations <<uses>> entre les CO et/ou les acteurs de l'environnement. Il ne contient que des références sur les opérations. Sa représentation graphique est donnée par la Figure IV-9. Par convention, le nom d'un objet médium est préfixé de la lettre M.

Un EPN est associé à cet objet. Il modélise uniquement les CT de communication. Dans le cas de l'exemple traité (cf. Figure IV-3), un temps de communication entre le SRS et l'acteur *H_Visual* est défini. Il est traduit sur l'EPN comme indiqué en Figure IV-10.

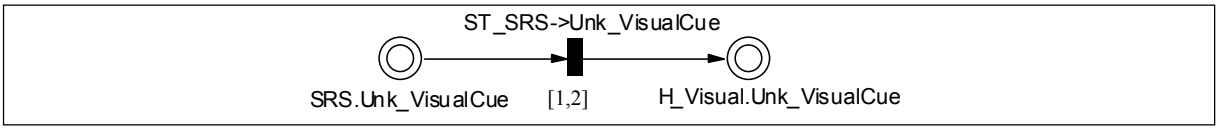


**Figure IV-10 : EPN de l'objet médium M_*SRS_H_Visual***

## IV.2.2. Dérivation des diagrammes d'activités

### IV.2.2.1. Rappels sur les diagrammes d'activités

IV.2.2.1.a) Quelle utilisation des diagrammes d'activités ?

Dans UML 1.3, les diagrammes d'activités sont considérés comme une sous-famille des diagrammes d'états/transitions. Ils sont utilisés afin de décrire un ensemble d'histoires, voir

l'ensemble des comportements possibles d'un classifieur (un objet mais aussi un cas d'utilisation ou un sous-système). Leur formalisme s'apparente à celui des diagrammes de flux, mais aussi à celui des réseaux de Petri. Nous ne nous sommes intéressés qu'à la dérivation des diagrammes d'activités modélisant le comportement d'un objet. Dans ce cas, trois utilisations des diagrammes d'activités peuvent être distinguées :

- La première est de détailler les opérations d'un objet (publiques et privées). Ainsi, pour chaque opération d'un objet, un diagramme d'activités est fourni.
- La seconde utilisation permet de décrire un ensemble d'interactions entre objets. La notion de corridor (*swimlane*) est introduite et permet de différencier les activités de chaque objet. Aux frontières de ce corridor, les données échangées entre objets peuvent être indiquées. Cette utilisation se rapproche de celle des diagrammes de séquence, l'analyste-concepteur disposant d'une plus grande richesse d'expression des alternatives, boucles et conditions.
- La dernière utilisation étend les deux premières à la représentation de toutes les opérations d'un objet sur un seul diagramme d'activités. C'est tout le fonctionnement interne d'un objet qui est décrit. Cette utilisation se rapproche alors de celle des diagrammes d'états/transitions.

Dans ces trois situations, les diagrammes d'activités se focalisent essentiellement sur la description des activités et des événements internes des objets.

IV.2.2.1.b) Syntaxe et sémantique des diagrammes d'activités

Une activité peut être considérée comme atomique (*action state*) ou composite (*sub-activity state*). Dans le premier cas, elle modélise un état exécutant une action indivisible. Dans le second cas, une activité composite est liée à un autre diagramme d'activités, qui est initialisé à son entrée. La sortie de cette activité composite correspond à la terminaison du sous-diagramme d'activités.

Des pseudo-états (*pseudo-state*) sont utilisés pour modéliser des synchronisations/duplications de fils de contrôle (*join/fork pseudo-states*) ou encore des alternatives (*junction pseudo-states*). Des transitions (représentées par des arcs orientés) relient les états et/ou pseudo-états afin de modéliser les flux de contrôle (*control flow*).

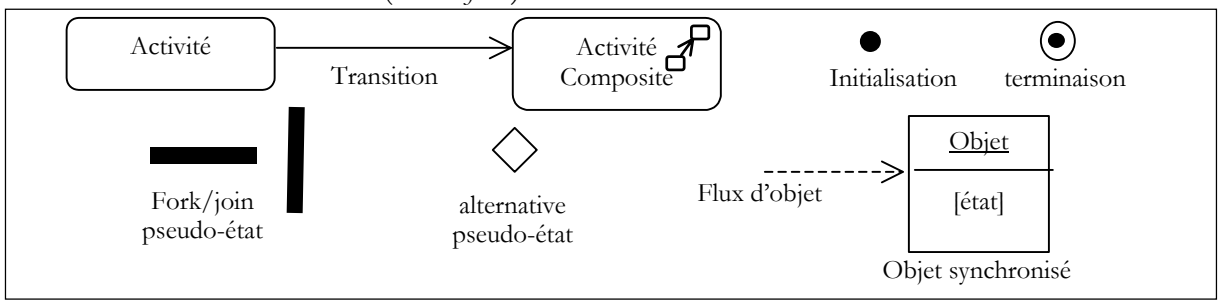


**Figure IV-11 : Principaux éléments syntaxiques des diagrammes d'activités**

Des flux d'objets (*object flows*), représentés par des arcs en pointillés, peuvent être indiqués entre des activités et des objets. Ils modélisent généralement une synchronisation entre objets.

IV.2.2.1.c) Limitation de l'expression du parallélisme

Une activité peut modéliser du parallélisme et être exécutée en même temps par plusieurs fils d'exécution.

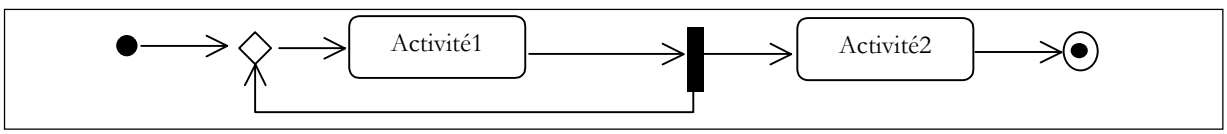


**Figure IV-12 : Modélisation de duplication non bornée de fils d'exécution**

Ainsi, le diagramme d'activités de la Figure IV-12 modélise un fil de contrôle pouvant avoir un

nombre non borné de fils d'exécution. En effet, à la fin de l'activité 1, les activités 1 et 2 sont relancées. En fonction de la durée de l'activité 2 (que l'on supposera beaucoup plus longue que l'activité 1), l'activité 1 peut se terminer et relancer une autre activité 2. Plusieurs exécutions concurrentes de l'activité 2 peuvent alors se dérouler.

Dans UML, les diagrammes d'activités doivent respecter la sémantique des diagrammes d'états/transitions. Or, l'exemple de la Figure IV-12 montre un comportement qui n'est pas modélisable pas un diagramme d'états/transitions [Eshuis-2002]. Pour cette raison, UML 1.3 interdit une telle utilisation des barres *fork* en demandant qu'à toute barre *fork* il corresponde une barre *join* et que l'imbrication des *fork*/*join* soit correcte (*well-nested*).

IV.2.2.1.d) Utilisation des états d'attente

En plus des éléments syntaxiques propres au diagramme d'activités, il faut parfois aussi utiliser des éléments des diagrammes d'états/transitions.

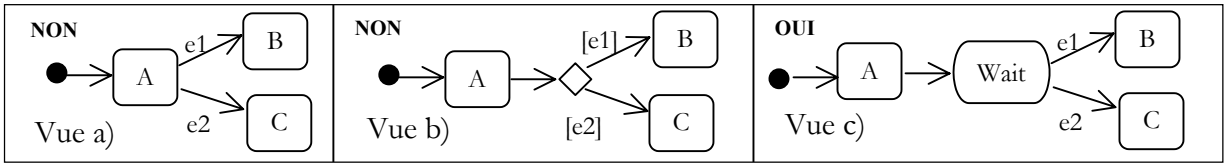


**Figure IV-13 : Branchements évènementiels incorrects et corrects**

La vue a) de la Figure IV-13 modélise un branchement conditionné par l'arrivée d'un évènement. Une telle représentation n'est pas permise dans UML car la sortie d'une activité doit se faire sur la terminaison de l'action liée à l'activité. Afin de pouvoir prendre en compte les évènements e1 ou e2, il faut déjà que l'action de l'activité A soit terminée. Par conséquent, il faut définir un état intermédiaire entre le moment où le système est sorti de A et le moment où les événements sont pris en compte, une transition n'étant pas considérée comme un état. Par ailleurs, des pseudo états (alternative ou duplication) ne peuvent pas être utilisés afin de modéliser cet état d'attente (vue b de la Figure IV-13), car il est interdit d'exprimer des attentes d'évènements dans leurs transitions. Il faut donc faire apparaître explicitement un état d'attente qui n'est pas une activité (vue c de Figure IV-13).

## IV.2.2.2. Quelques problèmes pour la dérivation des diagrammes d'activités

IV.2.2.2.a) Les ambiguïtés des flux d'objets

La notion de flux d'objets peut être ambiguë, car ces flux de contrôle sont parfois interprétés comme des flux de données [Bock-1999].

Ainsi, sur l'exemple (repris de [Booch-1998]) donné en Figure IV-14 a), Booch a voulu exprimer que : sur le retour d'un article par un client, l'activité *Restock item* est lancée. A la fin de cette activité, l'article (*item*) est de nouveau disponible et le compte du client crédité. Or, en appliquant la règle sémantique UML spécifiant qu'un flux d'objets est un flux de contrôle, la synchronisation n'est plus respectée. Après la sortie de l'activité *Restock item*, soit un article disponible est produit, soit l'activité *credit account* est activée, mais pas les deux ! La synchronisation doit être explicitement représentée (cf. Figure IV-14.b).

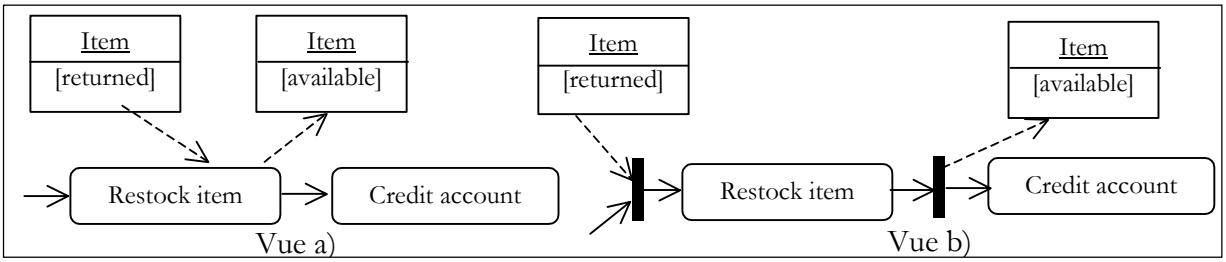


**Figure IV-14 : Représentations imprécise et correcte des flux d'objets**

Par ailleurs, ce n'est pas le seul problème posé par les flux d'objets. La Figure IV-15 présente deux imprécisions d'UML. Sur ce diagramme, les flux d'objets sont tous considérés comme synchronisés. Afin de ne pas complexifier le dessin, les barres de synchronisation avec l'objet ne sont pas représentées.

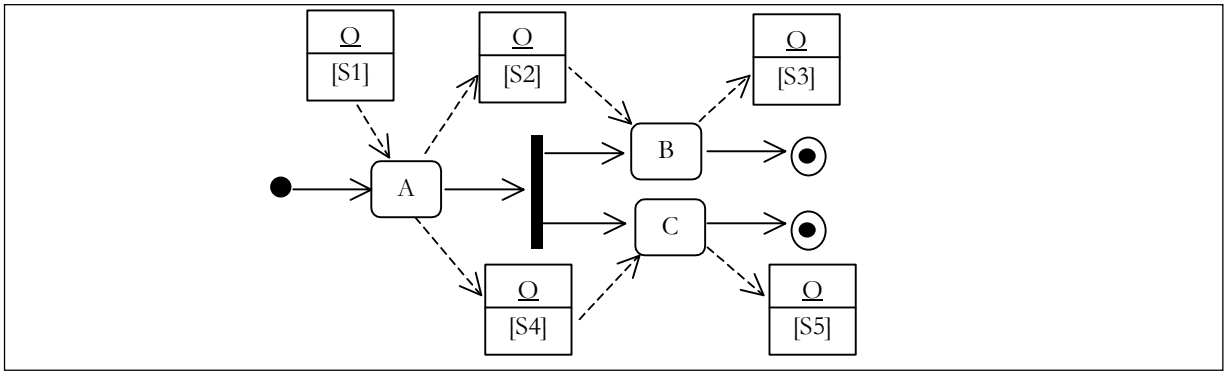


**Figure IV-15 : Multiples ambiguïtés des flux d'objets**

La première ambiguïté concerne l'objet O : s'agit-il de la même occurrence ou bien est-ce une instance différente pour chaque activité (B et C) s'exécutant en parallèle ? La seconde ambiguïté porte sur l'état de l'objet O (figurant entre crochets sur les objets) : est-ce un état complet ou bien un état partiel ? Dans le premier cas, le diagramme est faux si S2 et S4 ne représentent pas le même état. Dans le second cas, les états S2 et S4 doivent être parallèles et le couple (S2,S4) définit l'état complet de l'objet O. Dans cette situation, se posent alors les problèmes de l'accès concurrent à un même objet ainsi que de sa représentation .

La spécification UML 1.3 ne répond pas à ces questions. Elle laisse l'analyste-concepteur libre dans ses interprétations.

IV.2.2.2.b) Les différentes représentations des communications

Pour modéliser une communication (synchrone ou asynchrone), plusieurs représentations sont possibles et cette diversité rend plus complexe le processus de dérivation. Ainsi, pour une communication asynchrone (*request*) entre un client et un serveur, la Figure IV-16 présente différentes notations utilisables.

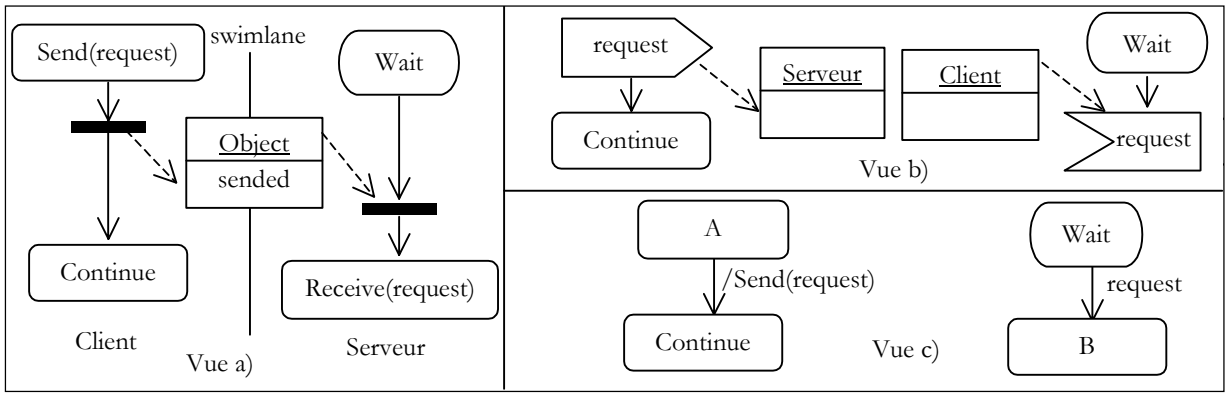


**Figure IV-16 : Représentations différentes d'une même communication asynchrone**

En vue a), la communication est représentée sur un seul diagramme d'activités. L'objet échangé est représenté sur le corridor (*swimlane*) séparant les deux objets. En vue b), des notations abrégées d'invocation et de réception asynchrones de signaux sont utilisées (symboles identiques à ceux employés dans l'échange de signaux en SDL). Ces représentations se font sur deux diagrammes d'activités séparés (un pour le client et un pour le serveur). Les flux d'objets

indiquent alors quel est l'objet envoyant ou recevant le signal et non plus l'objet transmis.

En vue c), une utilisation plus proche des diagrammes d'états/transitions est employée. L'envoi du message est effectué par une action sur la transition, la réception de l'évènement associé au message étant représentée sur la garde d'une transition.

Le même constat peut être effectué pour une communication synchrone (l'événement n'est plus un *signal* mais un *call event*).

IV.2.2.2.c) La difficile représentation des comportements périodiques

Une autre difficulté gênante pour la modélisation de STR concerne la représentation des comportements périodiques. La Figure IV-17 propose différentes modélisations d'une activité B (par exemple un échantillonnage) effectuée toutes les 5 secondes. Malheureusement, aucune des représentations n'est réellement satisfaisante.

En effet, dans le cas de la vue a), le comportement périodique de l'activité B est modélisé à l'aide d'une boucle et d'un état d'attente. La durée de l'attente correspond à la période moins le temps passé dans l'activité B. Cette modélisation suppose que le temps de l'activité périodique (ici B, mais cela pourrait être un ensemble d'activités avec des communications) soit fixe et connu à l'avance. Cette hypothèse n'est pas toujours possible.

En vue b, on considère que le *tick* de l'horloge est produit de manière externe à l'objet modélisé et que, sur la réception de ce *tick*, l'activité B est lancée. Malheureusement, si la durée de l'activité B est plus grande que la période, le *tick* suivant sera ignoré : il y a alors risque de désynchronisation.

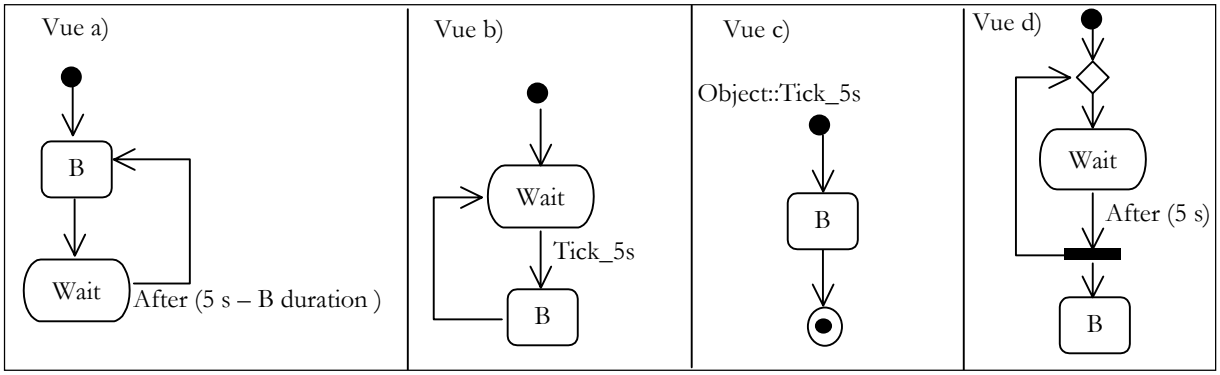


**Figure IV-17 : Différentes représentations d'un comportement périodique**

Afin de pallier cet inconvénient, le *tick* active directement une méthode (*Tick_5s*) de l'objet en vue c). Ainsi, même si l'activité B de la précédente occurrence du top horloge n'est pas terminée, une nouvelle activité B est lancée en parallèle avec celle non achevée.

Cependant, dans les deux derniers cas, la supposition d'une horloge externe à l'objet est peu adaptée à des modélisations en phase de spécification ou de conception préliminaire. Elle est trop proche des détails de l'implémentation.

La vue d) résout les problèmes des autres représentations. Cependant, elle ne respecte plus les règles UML d'utilisation des barres de synchronisation/duplication.

Ces quelques exemples sont représentatifs des ambiguïtés et des imprécisions des diagrammes d'activités. Elles devront être levées lors de la dérivation à l'aide de questions explicites posées à l'analyste-concepteur.

### IV.2.2.3. Principes généraux de dérivation UML/PNO des diagrammes d'activités

En raison de la parenté entre les diagrammes d'activités et les RdP, le principe de report des informations sur le RdP est simple (cf. Figure IV-18). Chaque activité est traduite par un ensemble de deux transitions (modélisant le début et la fin de l'activité) et d'une place (modélisant l'activité elle-même). Les connecteurs UML *fork* et *join* sont représentés par une transition et des places modélisant explicitement les synchronisations/duplications.

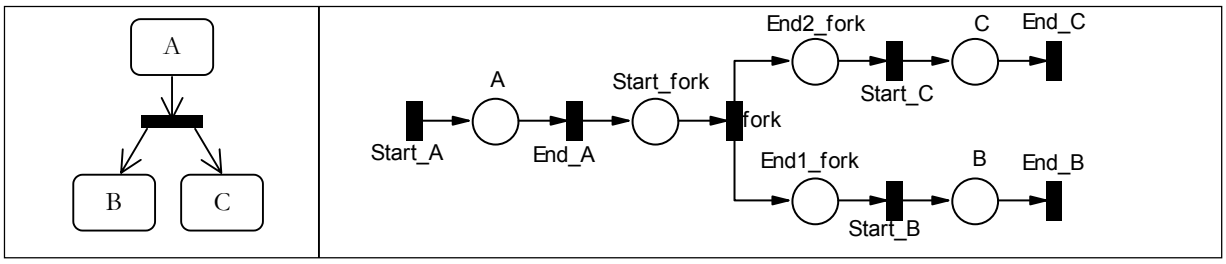


**Figure IV-18 : Principe de dérivation UML/PNO des diagrammes d'activités**

### IV.2.2.4. Différences avec des approches similaires de dérivation

Les autres approches UML/RdP pour les STR ne traitent pas ou peu des diagrammes d'activités (cf. §I.3.3.3). Ce sont surtout dans celles développées pour la modélisation des workflows que l'on trouve des mécanismes de dérivation des diagrammes d'activités. Deux stratégies différentes de dérivation existent (cf. Figure IV-19) suivant la sémantique donnée aux places et aux transitions des RdP. Dans le cas de la première stratégie de dérivation, les activités sont liées aux places du RdP [Eshuis-2002]. Dans le second cas, elles sont représentées par des transitions [Gehrke-1999]. Dans les deux cas, les barres *fork* et *join* sont traduites par une transition.

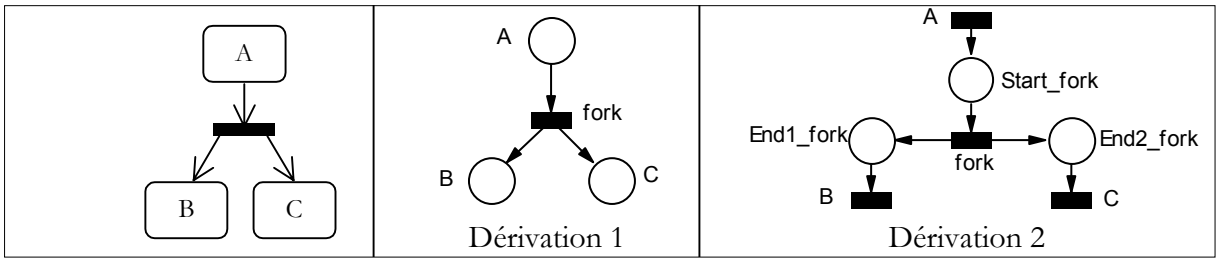


**Figure IV-19 : Autres stratégies possibles de dérivation des diagrammes d'activités**

De par la sémantique donnée aux RdP dans UML/PNO, une opération est représentée par une place, mais peut aussi, sous certaines conditions, être modélisée par une transition (représentation agrégée). Notre stratégie de dérivation englobe les deux stratégies existantes (cf. Figure IV-18). Nous pouvons nous ramener à l'une ou à l'autre en fonction des éléments modélisés. Il s'agit simplement d'une agrégation/simplification différente du RdP dérivé. Cette agrégation ayant un impact sémantique fort, elle ne peut être faite qu'avec l'assistance de l'analyste-concepteur et en respectant certaines règles élémentaires de réduction des RdP.

Comme pour les diagrammes de séquence, notre approche diffère essentiellement des autres techniques de dérivation par la prise en compte des spécificités UML/PNO et par le traitement des CT (non abordé par les autres approches).

### IV.2.2.5. Processus de dérivation des diagrammes d'activités

Comme pour la dérivation des diagrammes de séquence, le passage des informations des diagrammes d'activités vers les iBPN, est réalisé par les étapes suivantes :

1. Représentation des initialisations et terminaisons.

2. Représentation des comportements périodiques.
3. Représentation des fils de contrôle.
4. Représentations des communications.
5. Représentation des ressources partagées.
6. Simplification par agrégation guidée de l'iBPN.

Puisque les diagrammes d'activités représentent le fonctionnement interne d'un objet, sans détailler les évènements externes, leurs traductions fournit naturellement les RdP de comportement interne (iBPN) des CO. Précisons que, dans UML/PNO, la traduction des diagrammes d'activités se fait généralement après celle des diagrammes de séquence. De ce fait, un certain nombre d'informations sur les RdP de comportement est souvent connu (comme les protocoles de communication des CO, mais également certains traitements supposés).

L'intégralité des règles de dérivation n'est pas donnée ici eu égard à la place et à leur relative simplicité. Nous en avons déjà publié une partie [Paludetto-1999] et d'autres restent à définir, en particulier, pour la version 2.0 d'UML[50]. Dans ce mémoire nous préférons ainsi illustrer le processus de dérivation sur un exemple concret.

### IV.2.2.6. Illustration sur le simulateur SIMONA

Nous continuons sur le même exemple étudié lors de la dérivation des diagrammes de séquence. Nous considérons maintenant que la modélisation se déroule dans une phase plus avancée de la spécification. Une première décomposition du SRS a été définie (cf. §II.2.3.3) et les communications (ainsi que leurs protocoles) avec les acteurs de l'environnement sont connues.

La Figure IV-20 donne une partie du scénario (simplifié) de la boucle principale de calcul du simulateur. Toutes les 75 ms, le CO *S_Vehicle* doit calculer le nouveau déplacement de la cabine hydraulique. Pour effectuer ses calculs (méthode interne *ComputeModels*), il doit récupérer le mouvement du joystick (méthode synchrone forte *GetJoyMvt* du CO *S_Cockpit*) et des informations sur les conditions aérodynamiques environnant l'avion (méthode *AskWeather* du CO *S_Environment*). Le CO *S_Vehicle* peut être amené à modifier la consigne de régulation du Joystick (méthode asynchrone *SetJoyFBRegulation*).

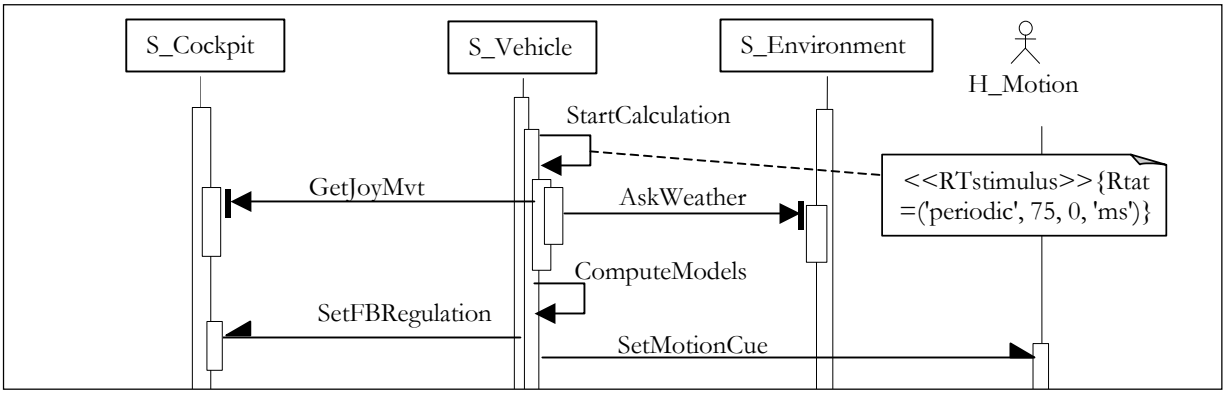


**Figure IV-20 : Scénario partiel de la boucle principale de calcul du SRS**

[50] Les premières propositions effectuées autour de la définition d'UML 2.0 offrent une version complètement revue des diagrammes d'activités. Ces derniers ne sont plus considérés comme une sous-famille des diagrammes d'états/transitions. Il semblerait que leur nouvelle sémantique soit très proche de celle des RdP (par exemple, la notion de jeton transitant dans des activités apparaît explicitement). Pour cette raison, nous préférons attendre la standardisation d'UML 2.0 avant de compléter nos règles de dérivation.

IV.2.2.6.a) Application du processus de dérivation sur le CO S_Vehicle

La Figure IV-21 présente une partie du comportement du CO *S_Vehicle* à l'aide d'un diagramme d'activités.

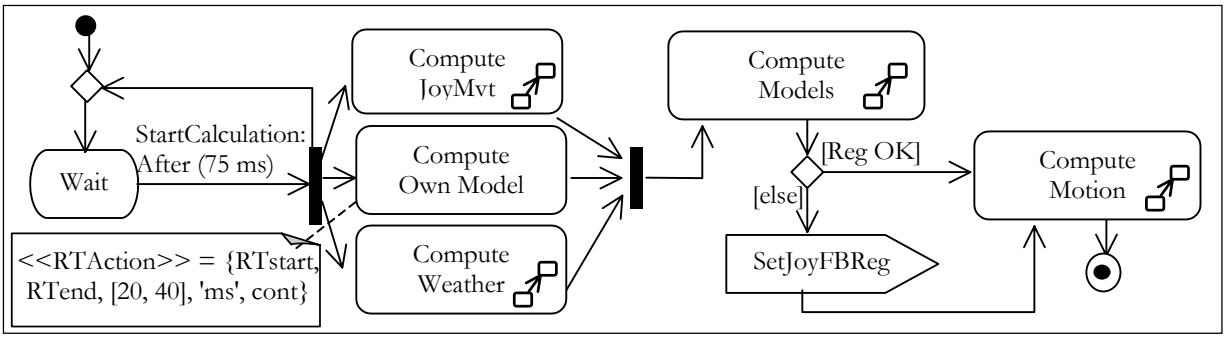


**Figure IV-21 : Partie du comportement global du CO S_Vehicle**

On retrouve une partie des informations déjà portées sur le diagramme de séquence de la Figure IV-20. Une activité interne et une CTP associée (*ComputeOwnModel*) supplémentaires apparaissent, ainsi qu'une alternative conditionnant la mise à jour de la consigne de régulation du joystick pour le retour d'effort. La plupart des activités sont composites et doivent être détaillées par d'autres diagrammes d'activités.

Seul le diagramme d'activités lié à l'activité composite *ComputeJoyMvt* est donné en Figure IV-22.

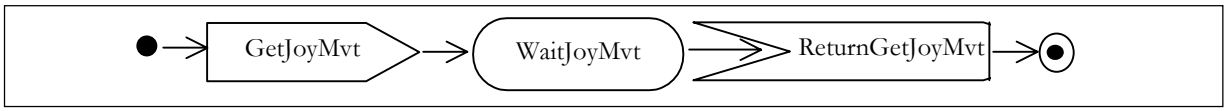


**Figure IV-22 : Détail de l'activité composite *ComputeJoyMvt***

Le processus de dérivation va principalement être appliqué au diagramme de la Figure IV-21.

IV.2.2.6.b) Dérivation des initialisations et terminaisons

Les états d'initialisation et de terminaison peuvent avoir plusieurs significations en fonction de l'utilisation du diagramme d'activités (description du comportement global d'un objet ou d'une méthode/sous-activité de l'objet). Ils peuvent modéliser : soit la création et la destruction d'un fil de contrôle de l'objet (Figure IV-21), soit la prolongation d'un fil déjà existant (Figure IV-22).

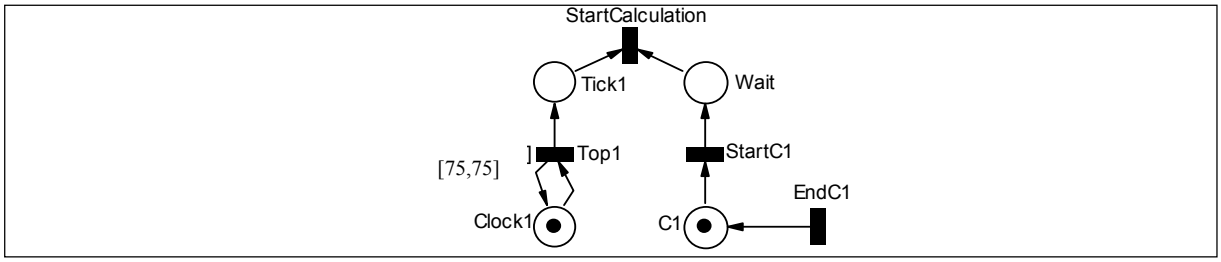


**Figure IV-23 : iBPN partiel du CO *S_Vehicle*, étapes 1 et 2**

Puisque l'ordre de traitement des diagrammes d'activités n'est pas imposé, il est possible que la dérivation s'effectue d'abord sur un diagramme détaillé (Figure IV-22) et non global (Figure IV-21). Par ailleurs, dans les premières phases de modélisation, les états d'un diagramme global ne modélisent pas forcément les étapes réelles d'initialisation et d'arrêt des fils de contrôle (au sens donné dans UML/PNO, cf. §III.3.3.). Ils peuvent simplement représenter une étape postérieure à l'initialisation ou précédant l'arrêt du fil de contrôle. En effet, il est courant, au début de la modélisation, que l'analyste-concepteur se concentre sur le comportement nominal

des objets en ignorant les phases d'initialisation et de destruction.

Par conséquent, il est difficile de déterminer automatiquement le sens exact à donner à ces états d'initialisation et d'arrêt des diagrammes d'activités. Nous préférons demander à l'analyste-concepteur quelles traductions choisir (par défaut, un démarrage ou un arrêt du fil de contrôle est proposé). Ce fil est modélisé par une place (nommée *CX* où X est un nombre) et deux transitions de synchronisation (*StartCX* et *EndCX*) (cf. Figure IV-23). Ce n'est que lorsque les fonctions *revive* ou *kill* sont explicitement utilisées (ou sur demande de l'analyste-concepteur) qu'une relation avec l'état *dead* de l'iBPN est faite.

IV.2.2.6.c) Dérivation des comportements périodiques

Pour cette dérivation, la plus grande difficulté consiste à identifier l'expression du comportement périodique dans le diagramme d'activités (cf. §IV.2.2.2.c). S'il est représenté comme en Figure IV-21, l'identification reste simple, car c'est un motif particulier. Dans le cas contraire (cf. Figure IV-17 vues a, b, ou c), seule l'utilisation de mot-clefs comme *tick*, *top, clock* ou la présence d'un comportement périodique dans l'eBPN, permet de demander plus d'informations à l'analyste-concepteur lors de la dérivation. Si l'analyste-concepteur ne respecte pas, soit la représentation telle que donnée en Figure IV-21, soit l'utilisation des termes précédemment cités, la détection et la dérivation des comportements périodiques ne sont pas assurées. Le comportement périodique (cf. III.3.2) est traduit par deux places (*ClockX* et *TickX*) et une transition *TopX* comme indiqué sur la Figure IV-23.

IV.2.2.6.d) Dérivation des fils de contrôle

Les fils de contrôle dans un diagramme d'activités sont modélisés à l'aide des activités et des pseudo-états. Le principe de dérivation de chacun de ces éléments a été donné en Figure IV-18.

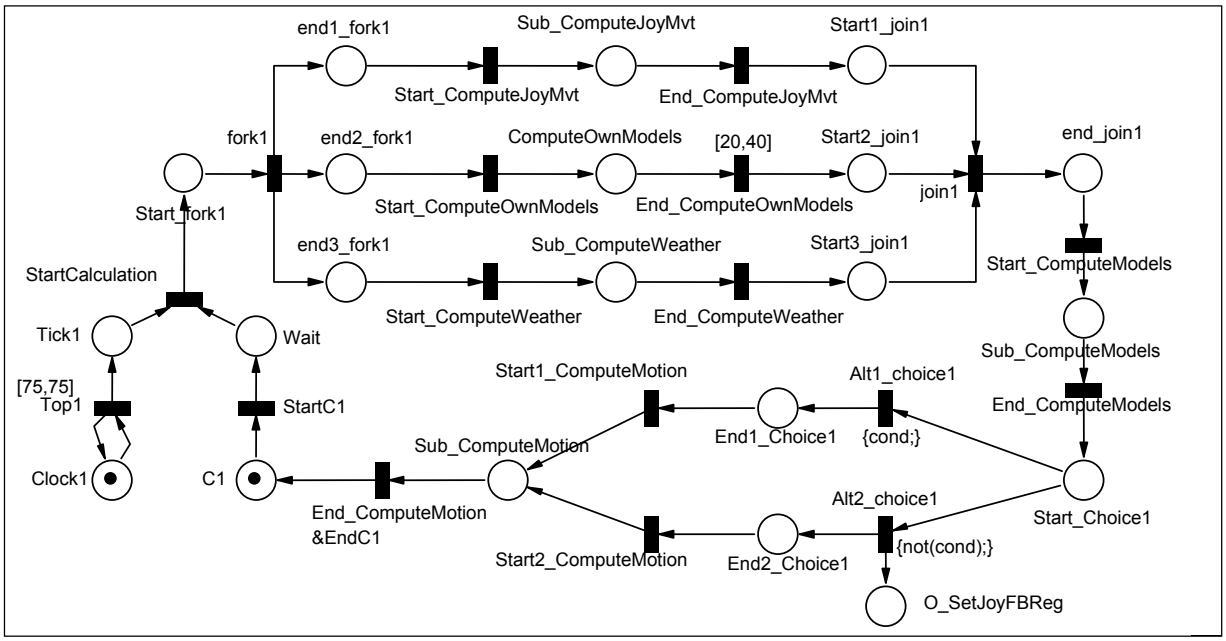


**Figure IV-24 : iBPN partiel du CO *S_Vehicle*, étapes 3 et 4**

Les CTP et les CTA sont reportées sur les transitions et/ou places suivant des principes similaires à ceux déjà présentés pour la dérivation des diagrammes de séquence.

La Figure IV-24 présente le résultat de la dérivation du diagramme d'activités du CO *S_Vehicle* (cf. Figure IV-21). Toutes les activités composites sont préfixées par « *Sub_* ». Cela permettra, lors de la dérivation ultérieure des sous-diagrammes d'activités, de les remplacer par les places et transitions correspondantes.

Cette dérivation des fils de contrôle, par ailleurs automatique, ne prend pas en charge l'expression des alternatives. L'analyste-concepteur doit compléter les conditions des transitions.

Le lecteur aura remarqué le cas particulier de la transition *End_ComputeMotion&EndC1* qui modélise la fusion des transitions *End_ComputeMotion* et *EndC1*.

IV.2.2.6.e) Dérivation des communications

En raison du grand nombre possible de représentations des communications, nous avons choisi de ne traduire que l'une d'elle. Seule l'utilisation des symboles, empruntés à SDL, d'envoi et de réception de signaux est permise. Une communication synchrone forte est alors représentée à l'aide de deux communications asynchrones. La dérivation des communications consiste à relier l'activité communicante à la place interne de communication (cf. place *O_SetJoyFBReg* de la Figure IV-24). La dérivation s'assure, par ailleurs, que le protocole de communication indiqué correspond bien à celui donné par les places internes de communications de l'eBPN.

IV.2.2.6.f) Dérivation des ressources partagées

Dans un diagramme d'activités, il est difficile d'indiquer, autrement que par des notes, la représentation de données et de ressources à accès protégés. Or, en temps réel, la modélisation de ce type d'informations est souvent nécessaire. Pour cela, nous proposons d'utiliser la notion d'objet des diagrammes d'activités tout en modifiant leur sémantique. En effet, nous avons présenté précédemment les ambiguïtés des flux d'objets et leur non-respect du principe d'encapsulation. Pour cette raison, dans UML/PNO, un objet modélise une ressource (par exemple une variable) en accès partagé. Les flux d'objets indiquent, dans ce cas, l'utilisation ou la libération de cette ressource. Au lieu de donner l'état de l'objet, on indique le nombre d'accès simultanés qu'autorise la ressource. La Figure IV-25 présente la notation UML et le principe de dérivation d'accès synchronisés entre deux activités (A et B). Par convention, la prise d'une ressource s'effectue avant l'exécution de l'activité. Sa libération se fait sur la terminaison de l'activité.

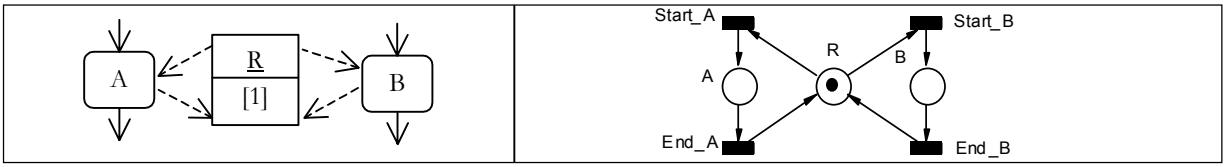


**Figure IV-25 : Ressource UML/PNO et sa dérivation**

IV.2.2.6.g) Simplification par agrégation guidée de l'iBPN

Une fois les règles de dérivation appliquées, le RdP obtenu est beaucoup plus complexe que le diagramme d'activités d'origine. Il peut être simplifié en agrégeant certaines séquences de places et de transitions. Les agrégations les plus répandues sont données en Figure IV-26.

Ces techniques de simplification de la représentation sont applicables si les actions (alternatives et synchronisations/duplications) sont considérées comme immédiates. Par ailleurs, la convention de nommage interne gardant une trace des simplifications effectuées, il est toujours possible de revenir à une représentation détaillée. Rappelons que cette convention de nommage est simplement destinée à faciliter les opérations de dérivation mais aussi de validation. L'analyste-concepteur peut ajouter ses propres noms aux places et transitions des diagrammes à RdP, mais il y a toujours conservation du nommage interne. L'iBPN réduit (agrégé) du CO *S_Vehicle* est donné Figure IV-27.

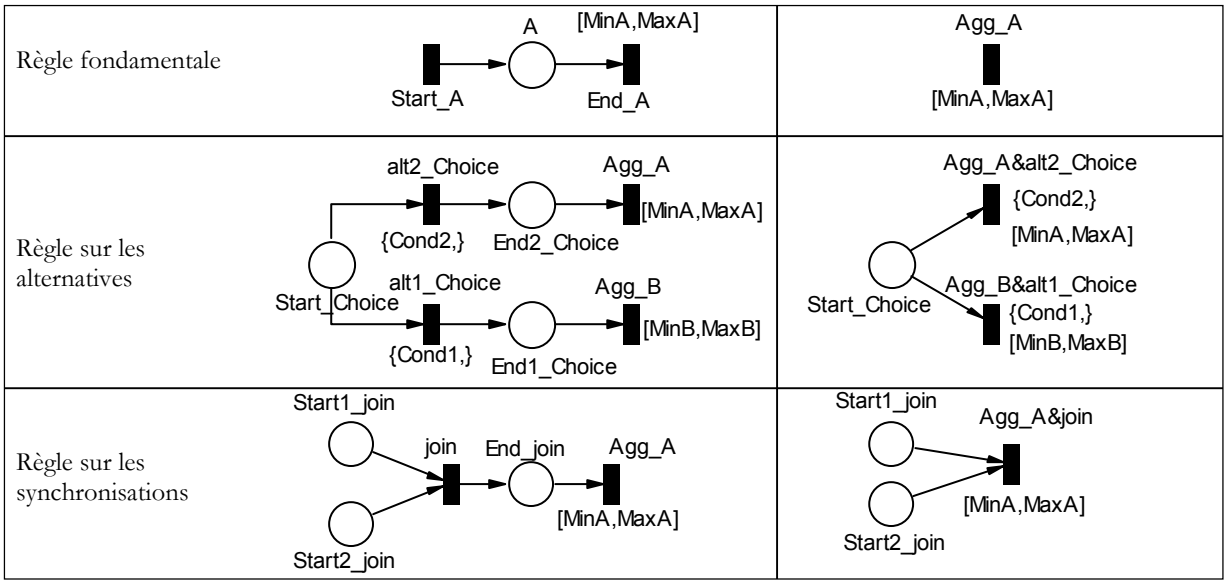


**Figure IV-26 : Agrégations courantes dans les iBPN**

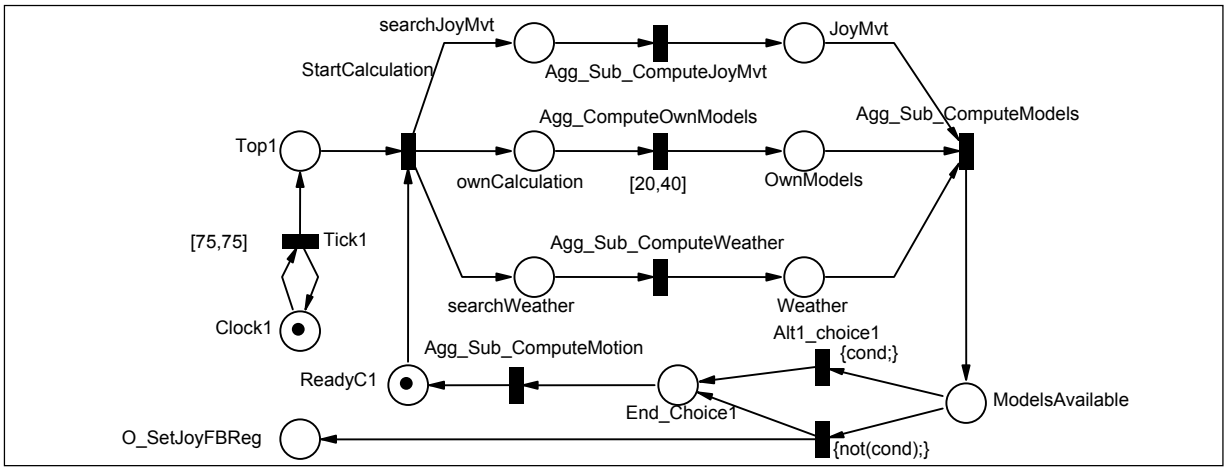


**Figure IV-27 : iBPN partiel de *S_Vehicle*, vue agrégée**

## IV.2.3. Bilan sur la dérivation des diagrammes

Les règles de dérivation présentées dans les chapitres précédents permettent de traduire la plupart des informations présentes dans les diagrammes d'interaction et d'activités. Une grande partie des eBPN et des iBPN, ainsi que des EPN, est générée sous le contrôle de l'analyste-concepteur. Ce dernier peut alors enrichir les diagrammes à RdP par des informations parfois plus complexes à représenter sur des diagrammes UML que sur les RdP.

Le processus de dérivation offre une première assistance à l'analyste-concepteur par des guides qu'il lui apporte. Ces guides qui ont été définis dans le processus de modélisation l'aident dans l'identification des informations (manquantes ou à préciser) et lui permettent une meilleure compréhension des diagrammes à RdP.

Notons que nous avons dû modifier la sémantique des flux d'objets dans les diagrammes d'activités et limiter leur représentation des communications. C'est là notre seul écart à notre volonté de rester le plus proche de la notation UML.

Par ailleurs, une autre aide importante résultant du processus de dérivation concerne les

activités de validation. En effet, puisque toutes les informations concernant les aspects dynamiques (disséminées dans les différents diagrammes UML dérivés) sont reportées sur une représentation unique (les RdP), et surtout formelle, il devient possible de détecter des incohérences entre les diagrammes UML. Ces activités de validation sont maintenant présentées.

# IV.3. Validation partielle des diagrammes UML

L'activité de validation ici proposée s'assure que la capture des diagrammes UML et le report des informations de ces diagrammes vers les diagrammes à RdP sont corrects et correspondent aux attentes des clients et des analystes-concepteurs.

Pour mener à bien cette tâche, nous proposons deux types de techniques :

- D'une part, une simulation des diagrammes à RdP et une retranscription de la trace de cette simulation sous la forme de diagramme de séquence.
- D'autre part, au cours et en fin de dérivation, un ensemble de règles est proposé afin de s'assurer, en partie, de la cohérence de la modélisation. Il s'agit essentiellement de cohérence[51] horizontale (cohérence entre plusieurs vues du système au même instant du développement) et non de cohérence verticale (cohérence des représentations tout au long du cycle de développement).

## IV.3.1. La simulation des diagrammes à RdP

La simulation est l'un des moyens les plus simples de détecter des erreurs, des situations et des histoires non prévues. Par exemple, une simulation pourra déceler le non-respect de CTA (dû à la mort de jetons dans des places temporelles ou à leur sollicitation prématurée). Elle peut aussi servir à valider la dérivation en régénérant les diagrammes de séquence initiaux (ayant servi à la génération des eBPN) à partir de la trace de simulations guidées.

Toutefois, la simulation est rapidement confrontée au problème de l'explosion des états : il est quasiment impossible de simuler tous les chemins. Les tests n'étant pas exhaustifs, la simulation n'apporte que des réponses partielles et c'est pour cette raison que nous la considérons comme une activité de validation. Elle reste néanmoins un outil apportant une aide appréciable durant la modélisation et peut être vue comme une technique complémentaire de la vérification formelle.

### IV.3.1.1. Construction du RdP à simuler

Afin de pouvoir simuler les diagrammes à RdP, l'analyste-concepteur doit, dans un premier temps, identifier les RdP concernés. En effet, pour un CO donné, il peut choisir d'utiliser son eBPN ou la composition de son eBPN et de son iBPN. De plus, il peut décider de la participation ou non des EPN des acteurs et des objets « medium ».

Les RdP choisis sont composés par fusion des places et transitions de communication afin d'obtenir un RdP global qui sera simulé.

Il existe, dans la littérature, un grand nombre de techniques (et d'outils informatiques) permettant la simulation des RdP. Citons, pour mémoire, [Bako-1990], [David-1992], [Valette-1985], [Feldbruge-1991], [Atamna-1994] ainsi que les outils logiciel CPN-AMI, Design CPN, Miss RdP, Syroco… Ces joueurs de RdP traitent une grande variété de classes de RdP (colorés, objet, t-temporel, stochastique…).

Toutefois, il n'y a pas, à notre connaissance, de joueur de RdP pour la famille pt-temporel stochastique que nous utilisons. Cependant, la réalisation d'un tel joueur, quoique tout à fait possible, reste accessoire. En effet, nous pouvons toujours nous ramener à une classe de RdP directement utilisable par les simulateurs existants en transformant nos RdP en RdP t-temporel ou t-temporel stochastique. Cette transformation peut simplement ignorer les places

[51] Pour une définition complète de la notion de cohérence dans UML, nous proposons au lecteur de consulter [Kuzniarz-2002]

temporelles[52] (les places et arcs associées ne sont pas reportés sur le RdP t-temporel) ou traduire ces places en un ensemble de places non temporelles et de transitions temporelles (cf. §V.1.2).

#### IV.3.1.2. Les différents modes de simulation

Différents modes de simulation sont alors possibles : simulation exhaustive, guidée, automatique, pas à pas, jusqu'à l'atteinte d'un marquage donné, avec arrêt sur conflit de transition ou de jeton…

Dans tous les cas, l'analyste-concepteur devra définir un marquage initial. Ensuite, différents choix de simulation sont concevables : simulation exhaustive à partir du marquage ou guidée par un scénario. Ce scénario pourrait être défini : soit directement à l'aide d'une liste de transitions des RdP (avec dates et ordres de tir ou non) et d'un marquage final, soit à partir d'un diagramme de séquence, soit en choisissant, au cours de la simulation, les transitions en conflits. Différentes politiques de tir des transitions temporelles peuvent être envisagées (au plus tôt, au plus tard, aléatoirement, sur demande à l'analyste-concepteur…) en fonction du joueur de RdP utilisé.

La trace de cette simulation peut alors être retranscrite sur un diagramme de séquence. En effet, du fait de notre construction des diagrammes à RdP et de la convention de nommage interne, les liens entre éléments des RdP et UML sont conservés. Par exemple, lors d'une communication entre deux entités, nous savons qu'au moins une place externe du RdP est mise en jeu. Il suffit de regarder avec quelle méthode UML cette place est en relation pour la retranscrire directement sur le diagramme de séquence. Nous pouvons aussi représenter, à chaque transition faisant évoluer le comportement de l'objet, l'état de cet objet (par un marqueur d'état donnant l'ensemble des places marquées du RdP). Des notes sur les CT peuvent être ajoutées, par exemple pour indiquer les dates de tir (ou l'intervalle de tir possible) ou encore pour modéliser les CTA déclenchées.

Ainsi dans son principe, la simulation et sa traduction sur un diagramme de séquence est simple, mais sa réalisation par un outil informatique reste complexe. Nous avons commencé ce travail en intégrant un éditeur et un joueur à RdP [Delatour-2003] dans un AGL UML (ArgoUML). Ce développement sera présenté au §V.4 de ce mémoire.

### IV.3.2. Vérification partielle de la cohérence des diagrammes à RdP

#### IV.3.2.1. Principes

Au cours de la dérivation, un ensemble de règles est utilisé afin de vérifier la cohérence du diagramme en construction. Il s'agit de détection de contradictions (entre éléments d'information du même type) avec les dérivations précédentes de diagrammes UML (protocoles différents pour une même communication, alternatives différentes…) ; des incohérences qui auraient pu échapper à l'analyste-concepteur car présentes dans différentes représentations UML. Dans cette situation, la différence entre ambiguïtés et incohérences est parfois ténue, car certaines incohérences apparaissent seulement lorsque l'ambiguïté est levée (et vice-versa).

Néanmoins, d'autres erreurs sont plus difficiles à détecter (étreintes mortelles, relations d'ordres contradictoires…) car ce n'est qu'une fois le diagramme à RdP complètement généré qu'elles constituent un ensemble d'informations incohérentes. La détection de ces incohérences est donc traitée une fois la dérivation terminée. Elles sont détectées : soit par la recherche des propriétés du RdP (pour les iBPN), soit par des algorithmes spécifiques (pour les eBPN).

Dans les deux cas, ces recherches ne sont qu'une aide pour la validation. Elles ne peuvent pas être considérées comme des techniques de vérification, car elles restent partielles et leur bon

[52] Dans ce cas, le non-respect d'une CTA n'est plus détectable.

déroulement (sans détection d'erreur) ne garantit pas l'absence de comportements indésirables.

Nous présentons ces deux ensembles de règles sur les diagrammes à RdP sur lesquels elles peuvent être appliquées.

### IV.3.2.2. La cohérence des eBPN

#### IV.3.2.2.a) Exemples d'incohérences détectées lors de la dérivation

La liste ne pouvant pas être exhaustive, nous avons préféré donner des exemples caractéristiques d'incohérences détectées lors de la dérivation des diagrammes de séquence vers les eBPN. Pour chaque incrément du processus de dérivation, un exemple est donné.

*IV.3.2.2.a.i) Incohérences de la représentation des communications*

Typiquement, il s'agit de détecter des incohérences concernant le protocole de communication. Dans un diagramme de séquence donné, un protocole est indiqué, tandis que dans un autre, c'est un protocole différent qui est modélisé. Cette incohérence est décelée lors de la traduction du deuxième protocole car il y a déjà des places et des transitions dédiées (construites lors de la première génération du message) sur l'eBPN, dont la représentation du protocole diffère (cf. Figure IV-28).

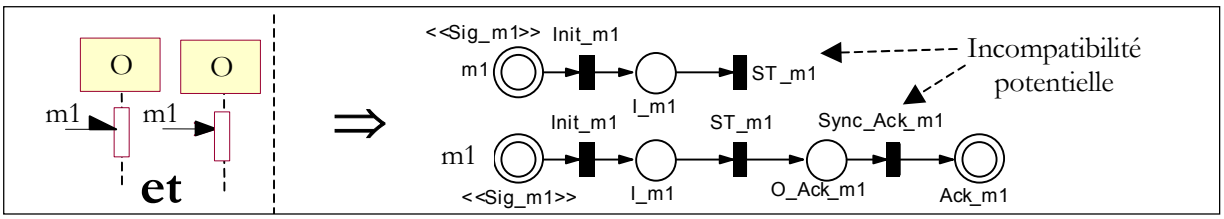


**Figure IV-28 : Exemple d'incohérence entre communications entrantes**

*IV.3.2.2.a.ii) Incohérences de la représentation des relations d'ordre*

Il s'agit d'incohérences entre les relations d'ordre des messages entrants et sortants : un ordre a été imposé dans un certain diagramme, mais il n'est pas respecté dans un autre. Ces incohérences sont détectées lors de la dérivation des CTA modélisant ces ordres. Il suffit de vérifier qu'il n'existe pas déjà de CTA sur les transitions d'initialisation et de synchronisation avec un ordre contradictoire (cf. Figure IV-29).

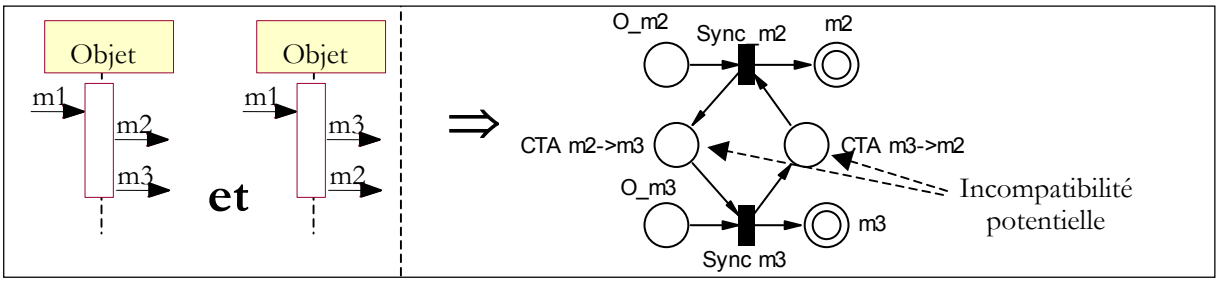


**Figure IV-29 : Exemple d'incohérence entre relations d'ordres**

Le seul cas où cette situation est tolérée correspond à une synchronisation de deux activités avec envoi simultané des deux messages (cf. Figure IV-30). De toute manière, il est nécessaire de procéder à une transformation des diagrammes de séquence et du RdP associé.

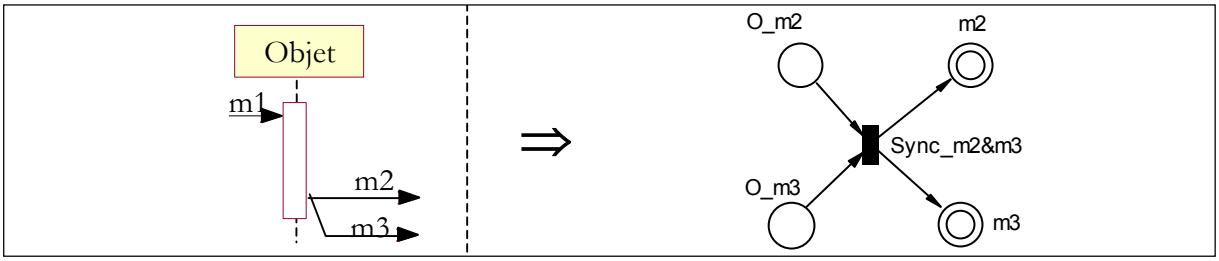


**Figure IV-30 : Cas de synchronisation et envoi simultané de messages**

*IV.3.2.2.a.iii) Incohérences à la représentation des alternatives*

Des incohérences peuvent se produire sur les conditions de tir des alternatives. Précisément, nous détectons les possibilités d'incohérence lors de la dérivation du second diagramme de séquence. Dans ce cas, nous demandons à l'analyste-concepteur de vérifier les conditions de déclenchement de ces transitions (que ce soit pour des post-conditions, des pré-conditions ou des conditions internes). Un exemple d'incohérence entre des alternatives de sorties est donné en Figure IV-31.

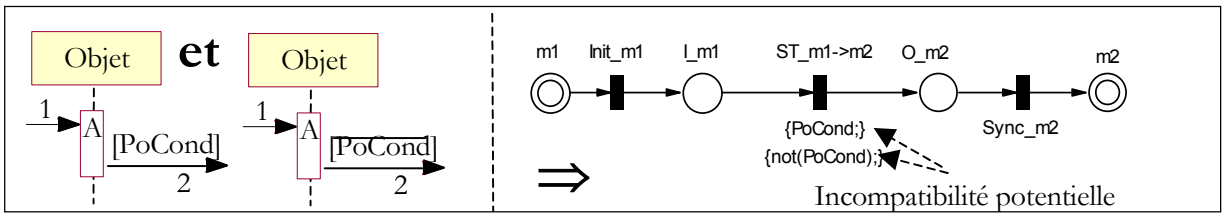


**Figure IV-31 : Exemple d'incohérence entre des alternatives de sorties**

IV.3.2.2.b) Cohérence temporelle des CTA des eBPN

*IV.3.2.2.b.i) Principe*

Nous présentons ici une aide à la détection d'une partie des incohérences entre les CT exprimées sur les eBPN. Elle se fait en vérifiant la cohérence de toutes les CTA ayant des transitions d'initialisation et de synchronisation distinctes (les CTA d'occurrences ne sont donc pas considérées).

Pour chacune de ces CTA, nous appliquons l'algorithme suivant :

- Recherche de tous les chemins structurels passant par la transition de synchronisation et d'initialisation de la CTA. Le marquage et les synchronisations sont ignorés.
- Chacun de ces chemins est temporellement quantifié (somme des intervalles temporels des places et des transitions temporelles rencontrées)
- On vérifie que l'intervalle calculé est inclus dans celui de la CTA à vérifier

La recherche des chemins est un simple parcours de graphe (du fait des principes de construction des eBPN).

*IV.3.2.2.b.ii) Exemple*

Nous reprenons l'exemple du SRS global déjà présenté dans cette partie IV, afin d'illustrer les principes de la vérification. Nous ne vérifierons que la CTA *SR2* de la Figure IV-32.

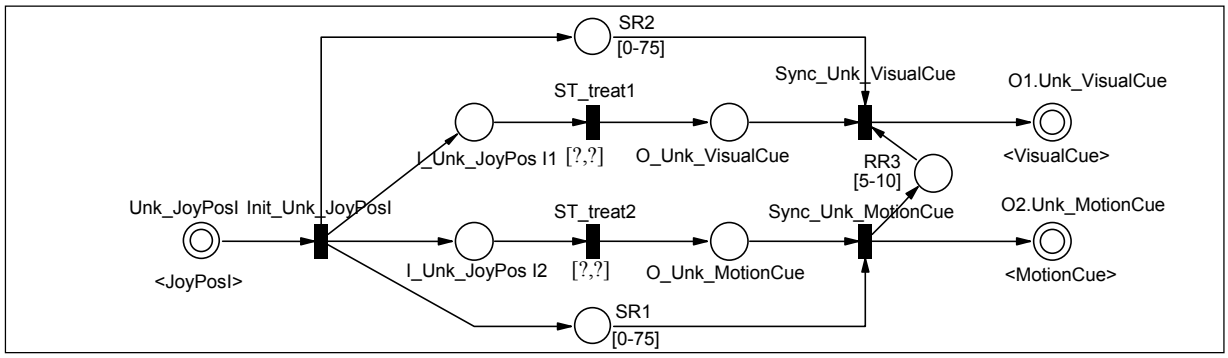


**Figure IV-32 : eBPN partiel du SRS**

En appliquant l'algorithme, nous identifions trois chemins structurels possibles pour cette CTA : le premier passant par *ST_Treat1*, le second par (*ST_treat2*, *RR3*) et le troisième par (*SR1*, *RR3*). La quantification du premier n'est pas possible (pas de CTP encore définie pour le traitement supposé de *ST_Treat1*). Pour le second chemin, nous obtenons [5,10] (là encore, la CTP de *ST_treat2* n'est pas définie). Pour le troisième, nous obtenons [5,85] (qui n'est donc pas inclus dans l'intervalle temporel [0,75] de la CTA *SR2*). Un problème est possible et sa présence indique des situations dans lesquelles la CTA *SR2* ne pourra pas être respectée. Une question est alors posée à l'analyste-concepteur afin d'identifier quelle CT doit être modifiée.

### IV.3.2.3. La cohérence des EPN

Pour les EPN des acteurs et des objets « medium », les vérifications sont plus simples. En effet, pour les acteurs, puisque les traductions concernent des motifs entiers de RdP, les incohérences sont directement retrouvées si, pour un même stimulus (ou une même réponse), l'analyste-concepteur choisit des motifs différents.

Pour les objets « médium », il s'agit typiquement d'incohérences consistant à donner des temps de communication différents pour un même message (cf. Figure IV-33).

Cette incohérence sera détectée lors du report du deuxième temps de communication du message sur l'EPN, car nous constaterons alors qu'il existe déjà une transition modélisant les temps de communication entre les places de communication.

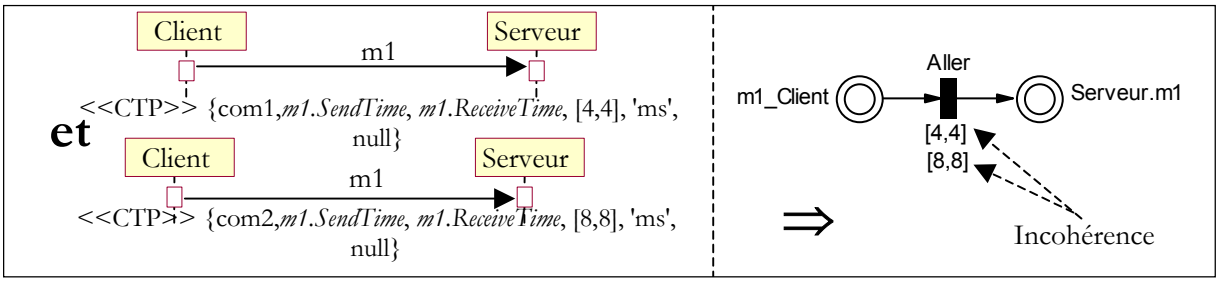


**Figure IV-33 : Exemple d'incohérence entre des temps de communications**

### IV.3.2.4. La cohérence des iBPN

Comme pour la détection des incohérences des diagrammes de séquence, nous retrouvons deux types d'aides à la validation des diagrammes d'activités. Le premier, effectué au cours de la dérivation, va surtout détecter des incohérences syntaxiques entre plusieurs diagrammes d'activités traduits vers un même iBPN. Le second, effectué à la fin de la dérivation, est basé sur la recherche des propriétés dynamiques et structurelles du RdP. Il offre une aide concernant la sémantique et la détection de comportements généralement considérés indésirables (blocage…).

IV.3.2.4.a) Exemples d'incohérences détectées lors de la dérivation

Naturellement, nous retrouvons des incohérences déjà abordées pour les diagrammes de séquence : protocoles contradictoires de communication, conditions illogiques sur des alternatives ou CT différentes pour une même activité… Toutefois, d'autres détections sont plus spécifiques aux diagrammes d'activités.

Une première catégorie correspond à la détection d'erreurs syntaxiques, par exemple : le diagramme d'activités est mal formé ou incomplet (activités pas toutes reliées par des transitions, absence de barre de synchronisation…). Une seconde catégorie est plus en relation avec des incohérences ou des ambiguïtés sémantiques.

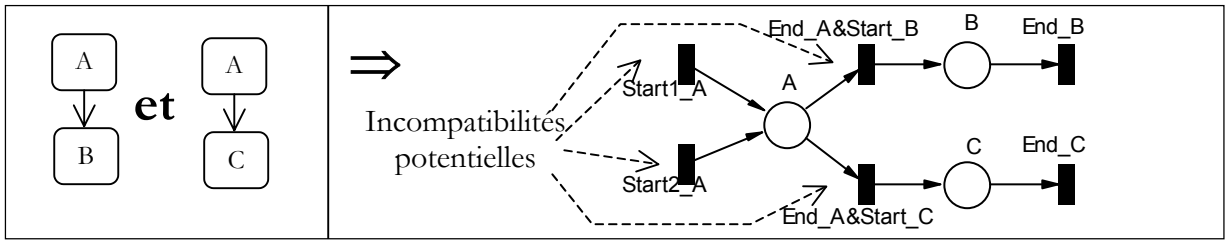


**Figure IV-34 : Relations contradictoires entre deux diagrammes d'activités**

Par exemple, en Figure IV-34, deux fragments de deux diagrammes d'activités sont représentés mettant tous les deux en jeu une activité A avec, pour chacun, une activité différente lui succédant. La dérivation du second diagramme entraîne la création d'un deuxième couple de transitions (*Start_A*, *End_A*) rattaché à la place *A* modélisant l'activité. Des questions doivent alors être posées à l'analyste concepteur afin de savoir si les deux fils de contrôle sont réellement différents. Dans le cas contraire (même fil de contrôle), il faut savoir si les activités B et C sont lancées en parallèle ou s'il s'agit d'une alternative.

IV.3.2.4.b) Incohérences détectées par la recherche des propriétés des RdP

Des incohérences peuvent être détectées par la recherche des propriétés dynamiques et structurelles de l'iBPN.

*IV.3.2.4.b.i) Préparation de l'iBPN*

L'iBPN doit être au préalable simplifié afin d'éliminer toutes les transitions et places qui gêneraient ces recherches. Par conséquent, les places internes de communication sont ignorées, car elles induisent soit des blocages (communications entrantes), soit des accumulations de jetons (communications sortantes).

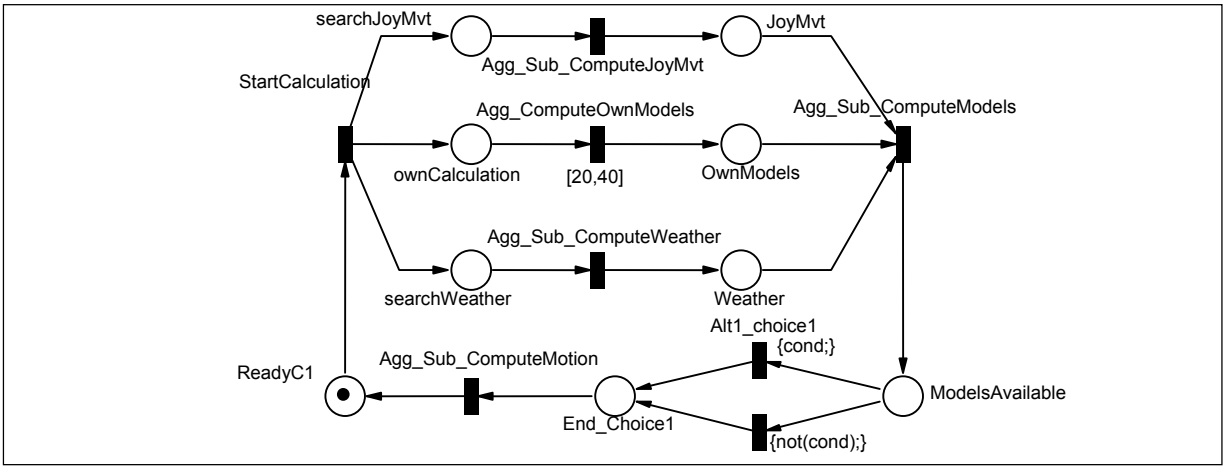


**Figure IV-35 : iBPN *S_Vehicle* simplifié pour l'analyse**

Pour les mêmes raisons, les places et transitions modélisant des comportements périodiques sont supprimées pour l'analyse. Cette simplification de l'iBPN de *S_Vehicle* est donnée en Figure IV-35.

*IV.3.2.4.b.ii) Recherche des bonnes propriétés*

De par les principes de construction des iBPN, cette simplification doit alors déboucher sur des RdP vivants, réinitialisables et bornés. Le cas contraire dénote généralement le signe d'un problème de modélisation.

En Figure IV-36, un diagramme d'activités incohérent est représenté. Sa dérivation en un iBPN est donnée. Sur la représentation simplifiée, on constate immédiatement un blocage du marquage dès que le jeton arrive dans l'une des places *StartX_Join*. Grâce à la recherche de l'accessibilité du graphe de marquage, ce type d'incohérence est facilement détecté.

Précisons que ces recherches se font sur des RdP marqués, en ignorant les informations de haut niveau portées par les jetons et les gardes des transitions. Elles ne doivent donc pas être considérées comme des techniques de vérification des bonnes propriétés du RdP de haut niveau associé et encore moins du comportement complet (composition de l'eBPN et de l'iBPN) du CO.

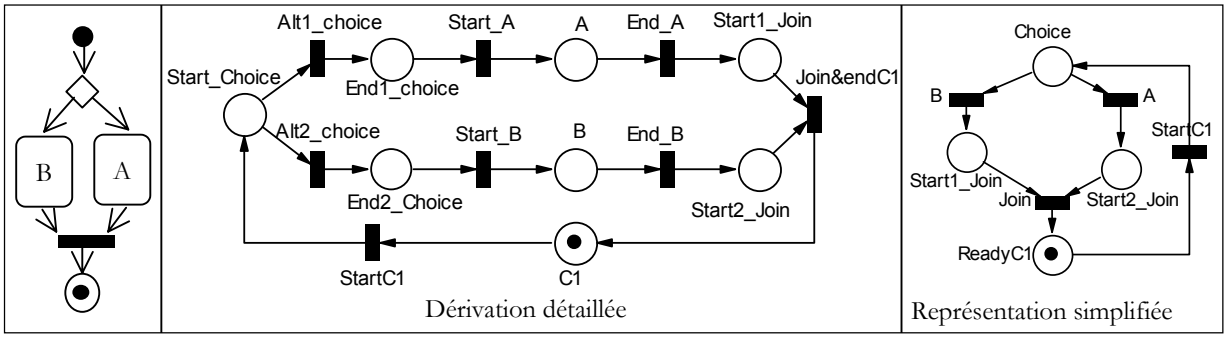


**Figure IV-36 : Diagramme d'activités incohérent**

*IV.3.2.4.b.iii) Recherche des propriétés structurelles*

La recherche des propriétés structurelles peut, elle aussi, apporter des aides à la modélisation. Nous réutilisons les travaux déjà développés sur la décomposition et la composition des RdP de comportement et la détermination automatique des sous-objets par recherche des composantes conservatives et répétitives des RdP [Paludetto-1991]. Leurs principes généraux sont simplement rappelés.

Ces aides sont basées sur la constatation qu'à une composante conservative correspond généralement un fil de contrôle de l'objet ou une utilisation particulière d'une donnée. La valeur de l'invariant linéaire de place permet de connaître le nombre maximum de fils d'exécution. Le fait de ne pas trouver de composante conservative sur l'iBPN analysé est signe de l'existence d'un nombre non borné de fils d'exécution. Il traduit généralement un mauvais partitionnement des fils de contrôle de l'objet modélisé.

La présence d'une composante répétitive stationnaire positive dénote un cycle de traitement. En appliquant ce principe aux diagrammes d'activités, nous pouvons considérer que nous devrons toujours retrouver, pour tout ensemble séquentiel d'activités situées entre une étape d'initialisation et de terminaison, une composante répétitive qui y correspond sur l'iBPN. Dans le cas contraire, il y a probablement une incohérence dans la modélisation.

# IV.4. Résumé sur la dérivation et la validation partielles

Dans cette partie, les processus de dérivation des diagrammes d'interactions et d'activités ont été présentés. Des règles de traduction ont été définies. Elles permettent de traiter la plupart des informations portées par les diagrammes d'interactions (séquence et collaboration), ainsi que les diagrammes d'activités. A une exception près (modification de la sémantique des flux d'objet afin de modéliser des ressources partagées), nous avons tenu à rester le plus proche de la syntaxe et de la sémantique semi-formelle d'UML.

En outre, deux grands types d'aides sont proposés à l'analyste-concepteur au cours des processus de dérivation :

- D'une part, une assistance à la construction des diagrammes à RdP, qui sont générés de manière semi-automatique à partir des diagrammes UML. Le processus de dérivation peut être considéré comme un guide méthodologique. En effet, il identifie les ambiguïtés (et manques d'informations) de la modélisation UML et demande alors à l'analyste-concepteur de les lever (ou de les compléter) par ses réponses à un ensemble de questions.
- D'autre part, tout au long de la dérivation, des vérifications sur la cohérence des diagrammes UML dérivés sont effectuées.

L'analyste-concepteur peut profiter des diagrammes à RdP générés pour mener des activités de validation, que ce soit par de la simulation ou par utilisation d'algorithmes spécifiques recherchant des incohérences globales.

Toutefois, ces aides de validation restent partielles et ne peuvent être considérées comme des activités de vérification. La partie suivante de ce mémoire présente des techniques de vérification complètes axées sur les CTA.

# Partie V Vérification des contraintes temporelles



Au §I.3.3.2, les principales techniques d'analyse temporelle des RdP t-temporel ou stochastiques ont été présentées. Parmi celles-ci, nous n'avons retenu que le graphe de classe.

En effet, puisque nous travaillons sur les STR fermes ou durs, nous avons focalisé notre recherche sur le respect d'une CT dans tous les cas, plutôt que sur la recherche de la probabilité du non-respect d'une CT.

Par ailleurs, nous avons utilisé une technique moins conventionnelle et relativement récente, non plus basée sur la recherche des états accessibles, mais plutôt sur l'évaluation quantitative de scénarii. Cette technique utilise une logique non monotone, nommée logique linéaire. Elle apporte une aide complémentaire au graphe de classe.

Ces deux techniques de calcul ont été définies pour des classes de RdP t-temporel. Or, nous utilisons une classe différente de RdP : les pt-temporel. Pour pouvoir utiliser ces analyses formelles, il nous fallait donc : soit transformer nos RdP pt-temporel en RdP t-temporel, soit adapter l'algorithme de preuve. Ce sont ces transformations ou adaptations qui sont présentées.

La démarche UML/PNO d'utilisation de ces deux techniques est ensuite illustrée sur un exemple issu du SRS. Cette approche est en cours d'implémentation dans un prototype d'AGL UML et RdP. Nous présenterons ce dernier ainsi que son état d'avancement. En conclusion, les limites actuelles de cette démarche sont données.

# V.1. Vérification des CT à l'aide du graphe de classe

Le graphe de classe [Menasche-1982] (cf. §I.3.3.2) est une technique développée pour les RdP t-temporel. Elle permet de décider, lorsque le RdP analysé est borné, de l'accessibilité des états du RdP, même avec la sémantique de tir forte (toute transition continûment sensibilisée doit être tirée avant sa borne temporelle supérieure).

Pour utiliser cette technique, nous avons effectué les choix de transformer :

- Les RdP pt-temporel utilisés dans UML/PNO en des RdP t-temporel,
- Les parties non bornées (typiquement, la représentation des comportements périodiques à l'aide de places et de transitions « horloges ») afin que l'analyse soit possible.

Ces transformations sont présentées ci-après. Auparavant, la technique du graphe de classe est abordée sur un exemple en guise de rappel.

## V.1.1. Exemple de graphe de classe

Afin de mieux comprendre l'utilisation du graphe de classe, nous proposons un exemple simple (voir Figure V-1), adapté de [Pradin-Chézalviel-2000], montrant les intérêts et limites de cette technique d'analyse.

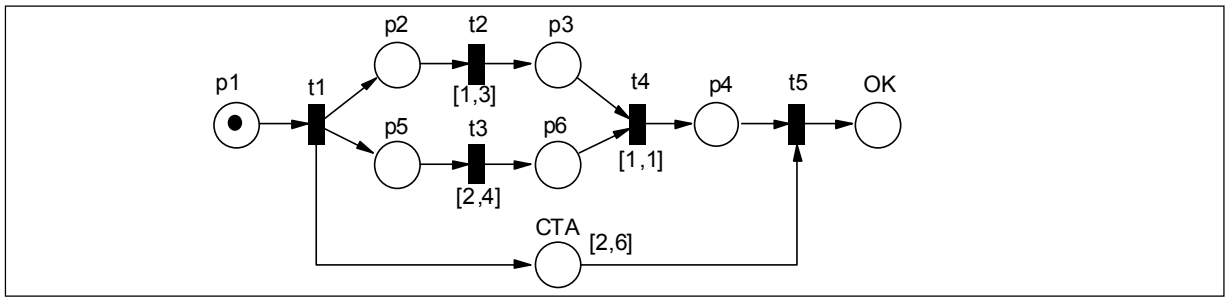


**Figure V-1 : Réseau de Petri avec parallélisme et une CTA**

Le graphe de classe correspondant au RdP donné en Figure V-1 (en ignorant pour l'instant la place CTA) est donné en Figure V-2.

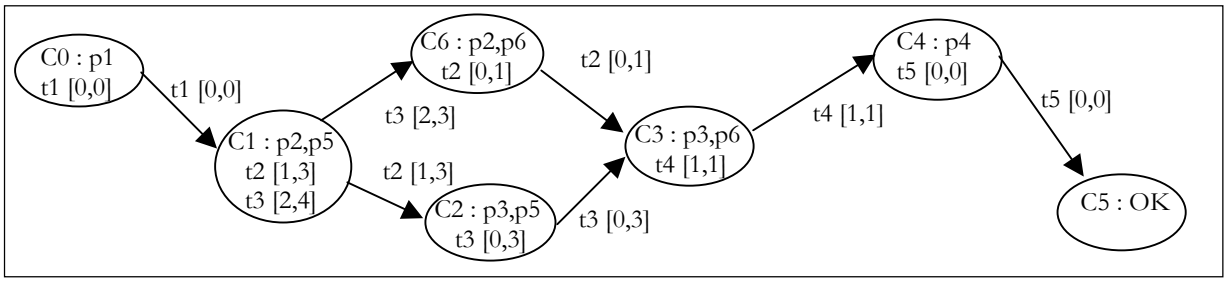


**Figure V-2 : Graphe de classe complet**

Pour chaque classe d'états (représentée par un ovale sur la figure), le marquage atteint et les durées de sensibilisation restantes des transitions sensibilisées sont indiquées. Les arcs, reliant les classes, précisent l'intervalle de tir possible ainsi que le nom de la transition à franchir.

Grâce au graphe de classe, nous pouvons déterminer les marquages accessibles avec leurs durées de séjour. En revanche, il n'est pas possible de l'utiliser pour déterminer des durées de chemins. En effet, sur l'exemple traité, deux chemins sont possibles ($t_1$, $t_3$, $t_2$, $t_4$, $t_5$) ou ($t_1$, $t_2$, $t_3$, $t_4$, $t_5$) dont les valeurs temporelles sont respectivement ([2,3]+[0,1]+[1,1] soit [3,5]) et ([1,3]+[0,3]+[1,1] soit [2,7]). Puisque le système peut prendre l'un ou l'autre des deux chemins, la

durée totale possible est [2,7]. Or, la durée réelle est [3,5]. Cette sur-estimation des durées provient des principes mêmes de construction du graphe de classe basé sur un formalisme qui ne considère pas les tirs simultanés de plusieurs transitions.

Par conséquent, afin d'appliquer la technique du graphe de classe à la classe des RdP pt-temporel que nous utilisons, nous ne pouvons pas considérer simplement une place temporelle comme une place normale du RdP. En effet, l'approche consistant à identifier tous les chemins possibles reliés à chaque place CTA et en quantifiant chacun d'eux serait incorrecte. Sur l'exemple traité, on pourrait alors statuer qu'il existe des cas où la CTA n'est pas respectée, ce qui serait inexact.

Par contre, le graphe de classe est une technique qui permet de déterminer tous les états accessibles d'un RdP. Il suffit donc, pour les utiliser afin de vérifier les CTA, de traduire explicitement en terme d'états (et donc de places) les tirs des jetons d'une CTA intervenant trop tôt ou trop tard. Pour cela, nous proposons une transformation des places CTA.

## V.1.2. Transformation des CTA

Pour expliquer le passage d'un RdP pt-temporel à un t-temporel afin d'utiliser le calcul du graphe de classe, il nous faut revenir sur ce que représente une place temporelle. Elle modélise une hypothèse temporelle (CTA) que l'on aimerait voir respectée. Elle ne modélise pas la partie contrôle et ne doit pas avoir d'influence sur le comportement. Ces places CTA peuvent être considérées comme des parties d'un RdP observateur, détectant le non-respect des CT et entraînant ou non alors l'arrêt de l'analyse.

Le principe de la transformation est de remplacer la place temporelle par un ensemble de places simples et de transitions temporelles représentant l'utilisation prématurée ou tardive des jetons de la CTA. Par exemple, la mort du jeton (resté une durée supérieure à la borne max de la place CTA) est spécifiée à l'aide d'un chien de garde : une transition (généralement nommée « viol ») et une place « CTA_Trop_Tard» rendent explicite (en terme d'états atteints) le non-respect de la CTA. Suivant le même principe, le tir prématuré d'une CTA conduira lui aussi à la présence d'un jeton dans une place « CTA_Trop_Tôt ». De ce fait, lorsque nous calculons le graphe de classe sur un RdP dont les CTA sont ainsi transformées, le marquage d'une de ces places « CTA_Trop_Tard » et « CTA_Trop_Tôt » indique alors le non-respect des CTA associées.

Plusieurs types de transformations des CTA sont possibles suivant :

- Que les transitions d'initialisation et de synchronisation de la CTA sont séparées (CTA de validité) ou confondues (CTA d'occurrence).
- La volonté de l'analyste-concepteur de bloquer ou non l'évolution du RdP sur le non-respect d'une CTA.

### V.1.2.1. Règles de transformation des CTA de validité

La Figure V-3 donne la transformation bloquante d'une place CTA associée à un traitement et à une attente.

Les règles de passage sont :

1. La place CTA est transformée en 4 places (place « CTA Début », « CTA OK », « CTA Trop Tôt », « CTA Trop tard »). L'arc entrant de la CTA de la transition « initialisation de la CTA » pointe alors sur la place « CTA début ».
2. La transition « synchronisation CTA » est dédoublée en deux transitions « Synchro_1 » et « Synchro_2 ». La première relie les places « Attente » et « CTA Début » vers la place « CTA Trop Tôt ». Elle modélise un traitement se terminant trop tôt. La seconde correspond à un respect de la CTA. Elle relie les places « Attente » et « CTA OK ».

3. Deux transitions temporelles sont créées. La première modélise le passage de la place « CTA Début » vers la place « CTA OK ». C'est une transition dont l'intervalle est [Min CTA, Min CTA]. La seconde représente l'expiration du délai d'attente de la fin du traitement. Elle relie la place « CTA OK » vers la place « CTA Trop Tard ». Son intervalle temporel est [Max CTA - Min CTA, Max CTA - Min CTA].

Remarque : si la borne min de la CTA est nulle, le RdP t-temporel est simplifiable. Les places « CTA début » et « CTA Trop Tôt » sont alors assimilées à la place « CTA OK ». La transition « Synchro 1 » disparaît.

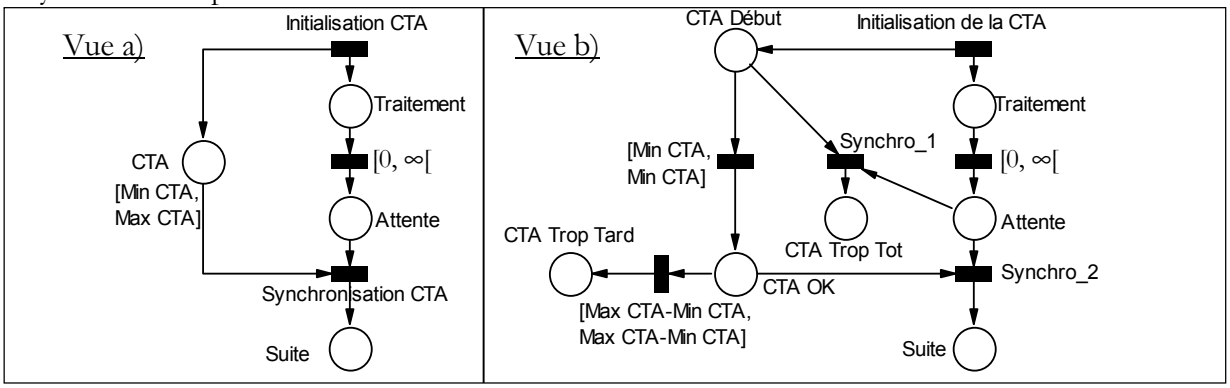


**Figure V-3 : Transformation bloquante des CTA de validité**

Cette transformation de la place CTA est ici intrusive par rapport au comportement du RdP car elle stoppe l'évolution du RdP lors du tir prématuré ou tardif du jeton de la CTA. Cela facilite l'analyse du graphe de classe et sa compréhension afin de ne pas avoir, dans le cas de plusieurs CTA analysées, un déclenchement en cascade des CTA.

Une transformation non bloquante (plutôt utilisée pour de la simulation avec des joueurs à RdP t-temporel) est donnée en Figure V-4 à titre d'exemple. Elle nécessite l'utilisation de RdP de haut niveau afin de différencier des sollicitations successives de la CTA.

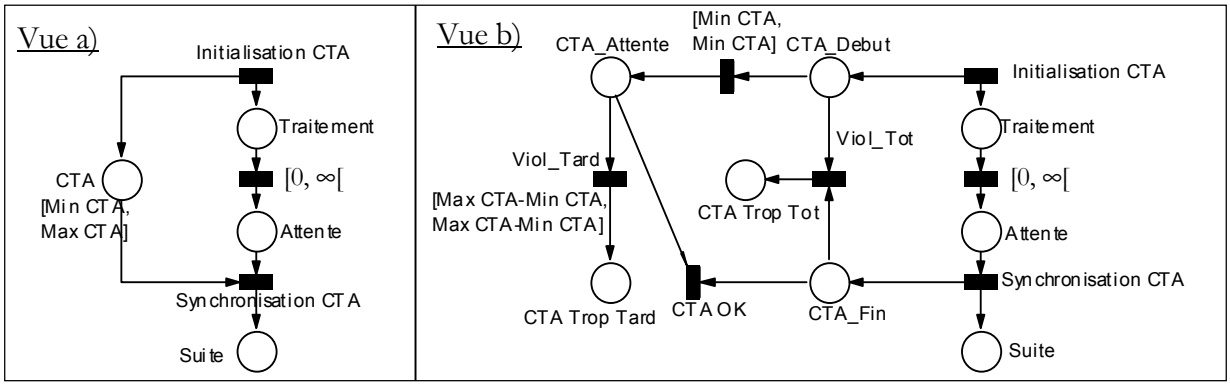


**Figure V-4 : Transformation non bloquante des CTA de validité**

### V.1.2.2. Règles de transformation des CTA d'occurrence

Les CTA d'occurrence modélisent des contraintes temporelles entre les occurrences successives d'un même événement. De ce fait, les transitions d'initialisation et de synchronisation sont fusionnées et la place CTA doit être initialisée par un jeton. La transformation n'est donc pas tout à fait similaire à celle présentée précédemment.

Néanmoins, le même principe est appliqué. Un ensemble semblable de places et de transitions est obtenu. La Figure V-5 illustre le principe de transformation de ce type de CTA. Nous avons donné ici une transformation bloquante.

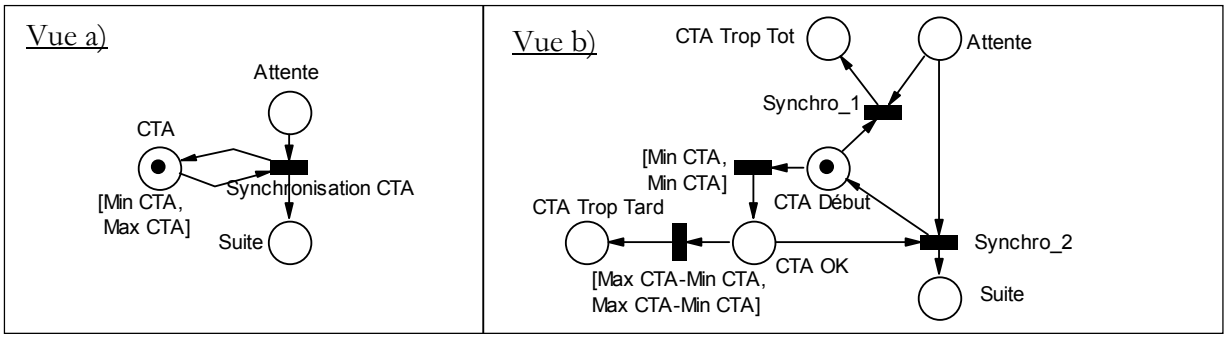


**Figure V-5 : Principe de transformation des CTA d'occurrence**

### V.1.2.3. Vérification des règles de transformation

Ces transformations ont été vérifiées en établissant les graphes de classe des RdP t-temporel (cf. Figure V-3.b et Figure V-5.b) et en vérifiant que des comportements similaires à ceux donnés par les RdP pt-temporel (cf. Figure V-3.a et Figure V-5.a) sont obtenus.

Ainsi, pour la transformation bloquante des CTA de validité, l'analyse du RdP donné permet de vérifier que la transition « Synchro_2 » ne peut être tirée que dans l'intervalle [MinCTA, MaxCTA]. A la mort d'un jeton dans la CTA correspond bien un marquage de la place « CTA Trop Tard », et ce pour les mêmes intervalles temporels. Une différence apparaît lors d'un tir prématuré de la CTA. Avec une place temporelle, ce tir est impossible (de par la sémantique des RdP p-temporel) car le jeton n'est pas utilisable avant une durée de séjour MinCTA. Toutefois, dans la transformation proposée, le jeton est disponible et un tir trop tôt (transition « synchro_1 ») peut donc se produire et être détecté. C'est d'ailleurs cette transformation vers les RdP t-temporel qui se rapproche le plus du sens que nous voulions donner à une CTA. Néanmoins, il est tout à fait possible de se ramener à la sémantique des RdP p-temporel en appliquant la simplification précédemment présentée en considérant la borne min de la CTA comme nulle (cf. remarque de V.1.2.1).

## V.1.3. Transformation des comportements périodiques

La construction du graphe de classe ne peut être faite que sur des RdP bornés (sinon, l'analyse ne peut pas se terminer automatiquement[53]). De ce fait, il faut éliminer, dans la mesure du possible, toute structure de places et de transitions ne respectant pas cette condition. Généralement, lorsqu'une modélisation est correcte, ce genre de situation est naturellement évité. Cependant, notre représentation des comportements périodiques (cf. §III.3.2) comporte une telle structure (cf. Figure V-6 a).

Il y a risque d'une accumulation des jetons dans la place *Tick1*. Cette situation dénote alors que le temps de traitement périodique (*Start_Calculation*, *Periodic_Calculation*, *End_Calculation*) est supérieur à la période. Il y a donc non-respect d'une CT.

Plusieurs stratégies de transformation sont possibles et dépendent de l'outil de construction de graphe de classe disponible, mais aussi du RdP à analyser :

- Si l'outil permet l'arrêt de l'analyse lorsque certains états sont atteints, il suffit de borner le RdP en y ajoutant une transition détectant l'accumulation de jetons dans la place *Tick1*. Elle

[53] Néanmoins, la plupart des outils de calcul des graphes de classe permettent, en présence de RdP non bornés, de définir des conditions d'arrêt sur le nombre de classes d'états calculées ou sur l'accessibilité de certains marquages.

arrêtera alors l'analyse (transition « Pb » et place « Viol » représentées sur la Figure V-6 b) puisqu'il n'y a plus respect de la périodicité.

- Parfois, l'outil d'analyse ne permet de spécifier des arrêts, lors de RdP non bornés, que sur le nombre de classes générées : il faut alors limiter le nombre de cycles qui vont être analysés (cf. Figure V-6 c). Il y a dans ce cas une limitation à l'exhaustivité de l'analyse.
- Dans d'autre cas, il suffit simplement d'enlever les places et transitions symbolisant l'horloge, car des CTA de production de réponses (ou de consommation de stimuli) sont déjà liées à ce traitement périodique. Leurs non-respects seront détectés avant celui du viol de la périodicité.

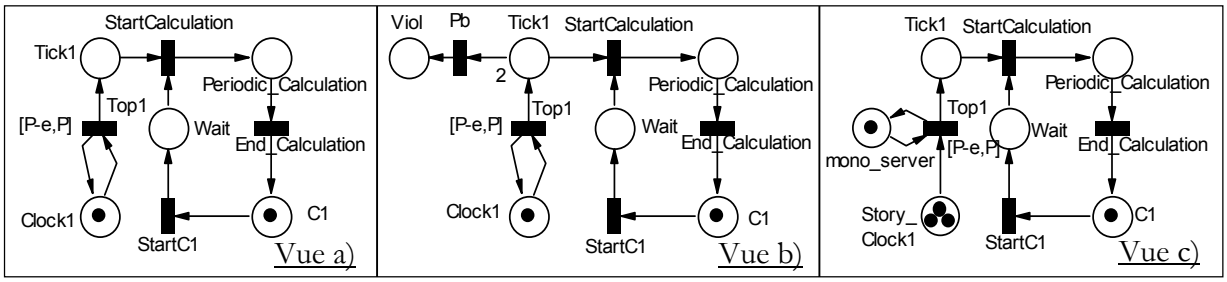


**Figure V-6 : Transformation des comportements périodiques**

Cette transformation des comportements périodiques peut être automatisée dans une certaine partie (la même stratégie est toujours choisie, par exemple la dernière, en ajoutant systématiquement une CTA au traitement périodique). Ce n'est que si cette transformation mène à un RdP non borné que l'intervention de l'analyste-concepteur est requise.

## V.1.4. Utilisation UML/PNO du graphe de classe

Comme lors de la simulation, l'analyste-concepteur doit identifier les RdP sur lesquels l'analyse portera. Pour des raisons pédagogiques, nous supposons pour l'instant que l'analyse ne porte que sur le comportement d'un CO et la vérification de ses CTA. Le comportement de ce CO est constitué par composition de son eBPN et de son iBPN (fusion des places internes de communications). Au §V.3, la composition des RdP de plusieurs CO sera détaillée.

L'analyste-concepteur devra donc définir des situations (les états du système) pour lesquelles la CTA sera vérifiée : c'est-à-dire un ensemble de marquages initiaux du RdP analysé. Les règles de transformations bloquantes des CTA et des comportements périodiques sont ensuite automatiquement appliquées. Pour chaque marquage à étudier, le calcul du graphe de classe est effectué. Différents outils peuvent être utilisés à cette fin. Citons Tina [Berthomieu-2001], Romeo [Lime-2003] ou Papetri [Berthelot-1991].

Du fait de la particularisation du non-respect des CTA en places "CTA trop tôt" et "CTA trop tard" (cf. Figure V-3), il suffit de rechercher tous les états du graphe de classe où ces deux places sont marquées. Si aucun état de ce type n'est rencontré pour la situation vérifiée, c'est que les CTA sont toujours respectées (et ce, quelle que soit l'évolution du RdP analysé).

Si des états de ce type existent, il y a viol de la CTA. Les chemins (séquences ordonnées de tirs de transitions) conduisant à ces états non désirés sont facilement identifiables (ce sont les arcs du graphe de classe menant de l'état initial à l'état non désiré).

Si le graphe de classe est maintenant calculé[54] sur le RdP après la transformation de sa CTA (cf. Figure V-7), nous pouvons alors vérifier son respect : les transitions $t_{5\_1}$ et $t_7$ ne sont bien jamais tirées.

[54] Pour des raisons de place, ce graphe de classe n'est pas donné.

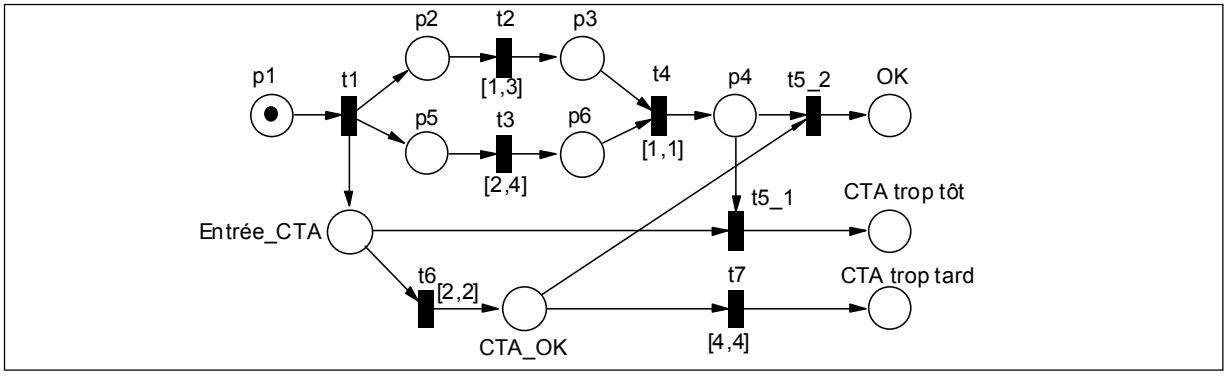


**Figure V-7 : RdP avec CTA transformée**

La valeur de la CTA est modifiée. Elle est remplacée par la valeur [2,4] afin d'obtenir des situations où elle ne sera plus respectée. Le graphe de classe obtenu est celui donné en Figure V-8.

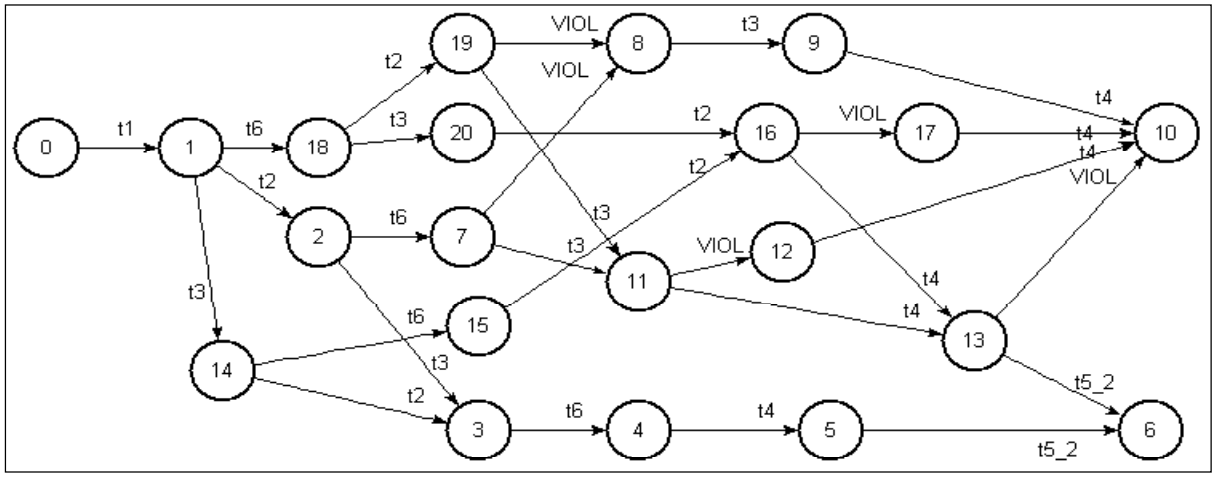


**Figure V-8 : Graphe de classe simplifié avec non-respect d'une CTA**

Pour des raisons de clarté, nous ne faisons pas figurer toutes les informations : seul le nom des transitions sur les arcs a été conservé. La transition « $t_7$ » a été renommée « VIOL » afin de rendre plus explicite les cas de non-respect de la CTA. On constate effectivement la présence de chemins respectant la CTA et d'autres ne la respectant pas.

## V.1.5. Bilan sur le graphe de classe

Le graphe de classe est l'une des seules[55] approches permettant une analyse exhaustive des états accessibles d'un RdP t-temporel. A l'aide des techniques de transformation des places temporelles présentées précédemment, il peut être utilisé pour la classe de RdP définie dans UML/PNO. Nous pouvons alors vérifier le respect ou non d'une CTA.

Néanmoins, cette technique de calcul présente quelques limites. Elle est très rapidement confrontée à l'explosion combinatoire. De plus, la modification d'une valeur d'un intervalle temporel d'une transition oblige alors à relancer le calcul.

Son utilisation dans UML/NPO n'est pas complètement automatisable. Bien que nous proposions des règles de transformation allant dans ce sens (pour la prise en compte des places temporelles), il reste de nombreuses situations requérant un spécialiste en RdP. En effet, lors de

[55] Une autre approche, nommée graphe des régions et issue de l'analyse des automates temporisés, est en cours de définition [Lime-2003]. Elle est implémentée dans l'outil Romeo. Elle reste similaire, dans son principe, à celle du graphe de classe, car elle est aussi basée sur un regroupement des états.

l'analyse de RdP ayant des comportements périodiques (ou des parties non bornées), plusieurs choix de modification sur le RdP peuvent être effectués. D'autre part et suivant les outils utilisés, il est souvent nécessaire d'enrichir manuellement le RdP analysé (par exemple pour définir le marquage initial ou l'arrêt de l'analyse).

Enfin, dans le cas du non-respect d'une CTA, la compréhension des causes reste difficile, même pour un expert. Le graphe de classe obtenu est rapidement de très grande taille : son traitement par des outils de model-checking est indispensable afin de le simplifier (par agrégation et renommage des états et transitions). Même dans ce cas, en ayant identifié des « classes » de chemins menant aux états redoutés, il est difficile de comprendre quelle CT peut être relâchée ou non, car on ne peut pas raisonner sur les durées établies par le graphe de classe.

C'est principalement pour cette dernière raison que nous nous sommes intéressés à la logique linéaire.

# V.2. Vérification des CT à l'aide de la logique linéaire

Les limites du graphe de classe nous incitent à travailler sur une autre approche qui permet une meilleure compréhension des relations entre les CT (les schémas de synchronisation). Plutôt que de calculer tous les états accessibles à partir d'une situation donnée, nous allons seulement étudier certains chemins caractéristiques et évaluer leurs durées. Le plus souvent, ces derniers seront les chemins menant à un non-respect d'une CTA. Leurs durées, données sous forme symbolique, permettent alors de mieux comprendre les relations entre CT. Cette quantification de la durée des chemins fait appel à une logique particulière : la logique linéaire.

## V.2.1. Principes de la logique linéaire

### V.2.1.1. Contexte de la logique linéaire

La logique linéaire[56] a été définie par J.Y. Girard à la fin des années 80 [Girard-1987]. Elle est basée sur l'idée consistant à « focaliser l'attention sur les propositions qui s'usent quand on s'en sert, par opposition aux vérités universelles et gratuites des logiques classiques et intuitionnistes » [Girault-1997].

Il s'agit d'une logique non monotone qui résulte de l'appauvrissement des règles de calcul de la logique classique. La règle d'idempotence n'est pas toujours applicable : une proposition « A » donnée n'est plus équivalente à « A et A » ou encore « A ou A » (et inversement[57]). De ce fait, deux connecteurs additifs (« avec » et « plus ») et deux connecteurs multiplicatifs (« fois », noté ⊗ et « par ») apparaissent dans cette logique, selon qu'ils respectent ou non la règle d'idempotence.

Dans cette partie, nous nous limitons à une présentation informelle du fragment MILL[58] de la logique linéaire qui nous sera nécessaire pour la compréhension de l'algorithme utilisé. Le lecteur intéressé par de plus amples informations pourra se reporter à l'exposé très didactique de Girault [Girault-1997] ainsi qu'aux nombreux articles de Girard [Girard-1990;Girard-1991].

### V.2.1.2. Notions élémentaires de la logique linéaire

Afin de mieux percevoir la différence entre une logique classique et la logique linéaire, considérons les raisonnements suivants :

[56] Si ce n'est par son nom, cette logique n'a aucun rapport avec la logique temporelle linéaire.
[57] Certains parlent plutôt de règles d'affaiblissement et de contraction
[58] Nommé fragment multiplicatif intuitionniste car seuls les deux connecteurs fois et implication sont utilisés.

| | |
|---|---|
| Pour trois €, j'ai un paquet de Gauloises (1) | Pour trois €, j'ai un paquet de Gitanes (2) |
| (or) j'ai trois € [A] | (or) j'ai trois € [A] |
| (donc) j'ai un paquet de gauloises [B] | (donc) j'ai un paquet de gitanes [C] |

Si on considère la première prémisse (1) comme l'implication classique A ⇒ B, l'utilisation de A permet d'obtenir B. Or, B ne se « consomme » pas : à partir du moment où on a A, B devient donc une vérité éternelle. Il est alors possible de réutiliser A avec (1) pour obtenir B ou bien de l'utiliser avec (2) pour obtenir C. De ce fait, une quantité infinie de paquets de cigarettes peut être obtenue…

P our introduire les idées de consommation et de production de ressources, la logique linéaire remplace l'implication classique ⇒ par l'implication linéaire ⊸. La proposition (1) devient donc A⊸ B qui signifie que l'on consomme A et A⊸ B pour produire B. On ne peut alors plus réutiliser ni (1) ni A. Il n'est plus possible de disposer d'une infinité de paquet de cigarettes.

De plus, puisque des connecteurs rejetant la notion d'idempotence ont été introduits, la proposition « j'ai trois € et j'ai trois € » peut se traduire par « j'ai six € » alors qu'en logique classique elle se traduit par « j'ai trois € ». La notion d'accumulation de ressources se représente à l'aide du connecteur « fois », noté ⊗. La proposition A peut se traduire par la formule €⊗€⊗€ ou $€^{⊗3}$.

### V.2.1.3. Introduction au calcul des séquents

Plutôt que d'utiliser une méthode axiomatique, qui est le formalisme syntaxique le plus utilisé avec les logiques classiques, Girard a choisi le calcul des séquents comme système de déduction pour la logique linéaire. Les séquents sont des déductions formées de la manière suivante : « formule prémisse ⊢ formule conclusion ». Au centre, le tourniquet, noté ⊢, sépare les deux formules du séquent :

- La formule à gauche du tourniquet est appelée prémisse. Elle consiste en une conjonction de métavariables, la virgule correspondant dans cette formule à un « et » (le « fois » en logique linéaire);
- La formule de droite est appelée conclusion. Elle consiste en une disjonction de méta-variables, la virgule, correspondant dans cette formule à un « ou » ( le « par » en logique linéaire).

La preuve d'un séquent se fait à l'aide de règles, appliquées à un séquent conclusion pour obtenir un ou plusieurs séquents prémisse. Elle se lit de bas en haut avec le sens suivant : pour prouver le séquent conclusion, il suffit de prouver les prémisses. Chaque étape étant séparée par une barre horizontale.

$$\frac{\text{séquent prémisse 1} \quad \text{séquent prémisse 2}}{\text{séquent conclusion}} \text{ nom de la règle appliquée}$$

Les règles du calcul des séquents dans le fragment MILL sont divisées en trois groupes : Le groupe fondamental, le groupe structurel et le groupe logique (cf. Figure V-9).

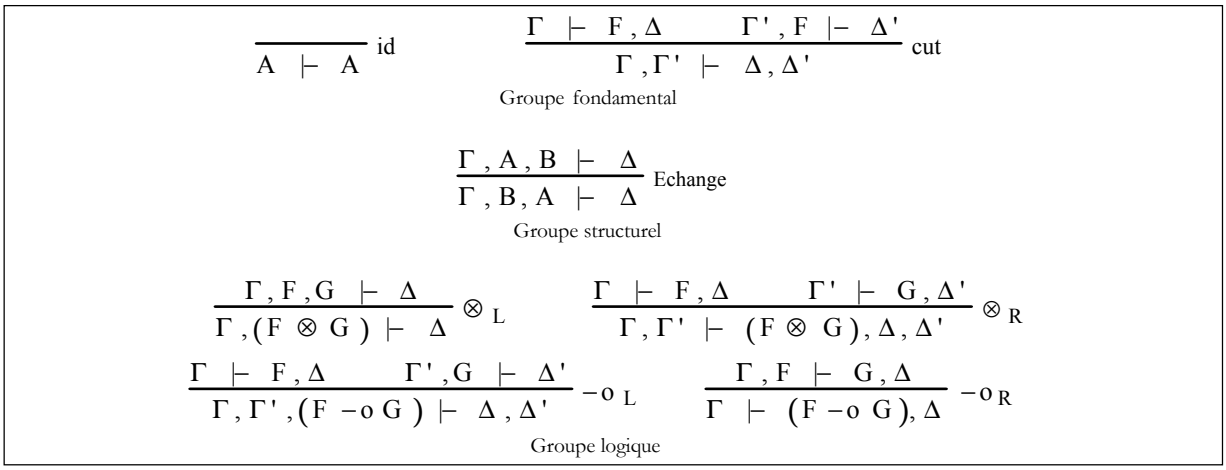


**Figure V-9 : Les règles du calcul des séquents du fragment MILL**

V.2.1.3.a) Règles du groupe fondamental

Le groupe fondamental est composé de deux règles : celle d'identité, notée *Id*, et celle de coupure, notée *Cut*. Ces deux règles ne font pas intervenir explicitement de connecteur :

- La règle d'identité est une règle d'axiome qui stipule que le séquent $F \vdash F$ est prouvable. Pour qu'un séquent soit prouvable, il faut que l'on retrouve cette règle sur toutes les feuilles de l'arbre de preuve de ce séquent.
- La règle de coupure indique que l'on a le droit de prouver un séquent en le divisant en deux séquents. Comme elle introduit une entité exogène, elle rend la preuve non canonique. Toutefois, cette règle a comme particularité d'être redondante. Un séquent pour lequel la preuve utilise la règle de coupure est aussi prouvable sans utiliser cette règle.

V.2.1.3.b) Règles du groupe structurel

Les règles du groupe structurel permettent de permuter des formules à droite et à gauche du tourniquet. En appliquant successivement ces règles, on peut effectuer n'importe quelle permutation de formules dans un côté donné (à gauche ou à droite) du tourniquet.

V.2.1.3.c) Règles du groupe logique

Le groupe logique définit les règles de calcul sur les connecteurs et les constantes de la logique linéaire. La description faite ici est restreinte aux connecteurs du fragment MILL.

Le connecteur ⊗ (« fois ») peut être introduit à droite ou à gauche du tourniquet :

- L'introduction de ce connecteur à gauche du tourniquet (⊗L) permet de séparer deux ressources liées par ⊗.
- L'introduction de ⊗ à droite du tourniquet (⊗R) permet de terminer la preuve d'un séquent. En effet, elle permet d'obtenir deux séquents identité à partir de la formule A,B ⊢ A⊗B.

Le connecteur ⊸ (« entraîne ») peut lui aussi être introduit à droite ou à gauche du tourniquet :

- Son introduction à gauche (⊸ L) permet d'effectuer l'action correspondant à la ressource d'action portée par le connecteur ⊸ . En utilisant cette règle, on consomme donc les ressources à gauche du connecteur pour produire celles qui sont à droite.
- L'introduction à droite (⊸ R) permet de supprimer une ressource d'action qui se trouverait à droite du tourniquet. Cette règle montre la relation de parenté entre ⊸ et ⊢

puisqu'elle permet d'écrire ⊢ A ⊸ B sous la forme A ⊢ B.

L'équivalence entre l'accessibilité de deux états d'un RdP et l'existence de la preuve d'un séquent de la logique linéaire a été montrée grâce à la théorie des catégories [Marti-Oliet-1991]. De ce fait, des couplages entre RdP et logique linéaire ont été effectués.

## V.2.2. Le couplage entre la logique linéaire et les RdP

Parmi les différentes approches établissant des liens entre la logique linéaire et les RdP, nous utilisons celle proposée par Brigitte Pradin Chézalviel et Robert Valette et initiée dans la thèse de Girault [Girault-1997]. La majeure partie de leurs travaux sur la logique linéaire est résumée dans [Pradin-Chézalviel-2003].

### V.2.2.1. Représentation d'un RdP

A chaque place d'un réseau de Petri (a**,** b**,** c sur la Figure V-10) est associé un type d'atome de la logique (notés A, B, C…). Un jeton d'une place est alors représenté par un atome de la logique linéaire associé à cette place. Un marquage est dénoté par une accumulation d'atomes, traduite par l'opérateur linéaire ⊗. Le marquage du RdP de la Figure V-10 est représenté par la formule A⊗A⊗A⊗B (aussi noté $A^{\otimes 3}$ ⊗B).

Les transitions d'un réseau de Petri sont exprimées par le biais de l'implication linéaire « entraîne », notée ⊸ . L'implication linéaire exprime alors la possibilité de produire des jetons à partir de la consommation d'autres jetons du RdP.

Ainsi, la transition t de la Figure V-10 sera représentée par A⊗A⊗B⊸ C. Cette formule ($A^{\otimes 2}$⊗B⊸ C) est elle-même une ressource (dite d'action). C'est-à-dire que la possibilité de production disparaît dès que l'implication est utilisée. Si nous voulions modéliser le fait que cette transition est franchissable un nombre non déterminé de fois (dépendant du marquage), nous devrions utiliser le connecteur « bien sur », noté « ! », ne faisant pas partie du fragment MILL.

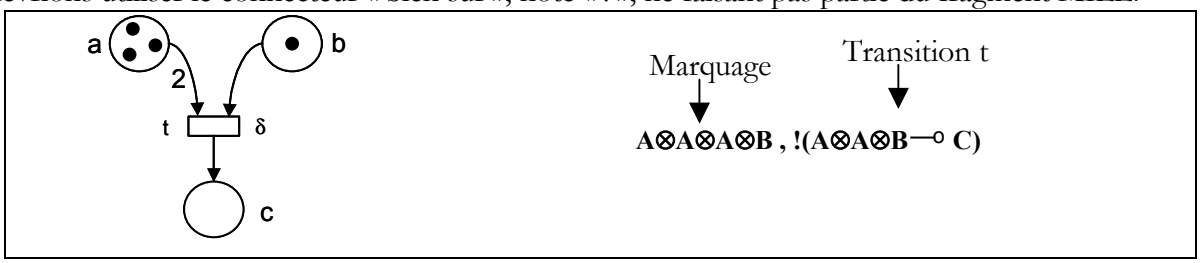


**Figure V-10 : Représentation monoïdale en logique linéaire d'un RdP**

La Figure V-10 présente le RdP et sa représentation (structure et marquage) en logique linéaire.

### V.2.2.2. Evolution du RdP

Une fois le RdP représenté sous forme de formules de logique linéaire, on peut modéliser le franchissement de la transition t (toujours avec l'exemple de la Figure V-10), qui fait passer le RdP du marquage $A^3B$ au marquage AC, à l'aide de la preuve d'un séquent.

La preuve du séquent (les prémisses se terminent tous sur une règle d'identité) est ici établie (cf. Figure V-11) et montre que le marquage est accessible. Cette preuve est une condition nécessaire et suffisante pour un RdP normal. Nous verrons que ce n'est plus qu'une condition nécessaire pour un RdP temporel.

Naturellement, ce principe de franchissement d'une transition peut être généralisé au franchissement d'une séquence de transitions. Nous parlons alors de scénario pour désigner l'ensemble définissant cette séquence. Un scénario est constitué des marquages initiaux et finaux

ainsi que d'une liste de transitions.

$$\dfrac{\dfrac{\dfrac{}{A \vdash A}\,id \quad \dfrac{}{B \vdash B}\,id}{A, B \vdash A \otimes B}\otimes R \quad \dfrac{\dfrac{}{A \vdash A}\,id \quad \dfrac{}{C \vdash C}\,id}{A, C \vdash A \otimes C}\otimes R}{\dfrac{A, A, B, \vdash A \otimes A \otimes B \qquad A, C \vdash A \otimes C}{A, A, A, B, A \otimes A \otimes B \multimap C \vdash A \otimes C}\multimap L}\otimes L$$

$$A \otimes A \otimes A \otimes B,\ A \otimes A \otimes B \multimap C \vdash A \otimes C$$

**Figure V-11 : Preuve d'un séquent**

### V.2.2.3. Calcul de la durée des scénarii par la logique linéaire

La durée du scénario est calculée en établissant un arbre de preuve d'un séquent de la logique linéaire. L'algorithme présenté est celui donné par [Pradin-chézalviel-1999b] et [Pradin-chézalviel-1999c]. L'avantage d'utiliser la logique linéaire est que la durée ainsi calculée ne prend en compte que les relations d'ordres partiels imposées par la structure du RdP et son marquage sans introduire de relation parasite (en particulier introduite par des techniques dites d'entrelacement).

Voici l'algorithme utilisé :

```
VERIFIER que le scénario est complètement spécifié,
APPLIQUER la règle ⊗L
TANT QUE la règle ⊸L est applicable
- Appliquer la règle ⊸L à l'une des transitions candidates (ordre
lexicographique)
- Terminer la preuve du séquent gauche généré en utilisant, si nécessaire,
la règle ⊗R
- Appliquer, si nécessaire, la règle ⊗L au séquent droit
FIN TANT QUE
```

Cet algorithme permet d'effectuer une preuve de séquent. Nous reviendrons plus tard sur la condition de vérification de l'algorithme portant sur la notion de scénario complètement spécifié. Le choix d'une transition candidate plutôt que d'une autre ne modifie pas le résultat de la preuve. Néanmoins, afin d'obtenir un arbre de preuve canonique, un ordre lexicographique a été choisi.

Afin de calculer la durée symbolique d'un scénario, il suffit d'introduire les notions de dates de production et de consommation des jetons du réseau. Ainsi, à chaque atome (représentant un jeton) de la preuve et pour chaque étape courante[59] est associé un couple (date de production, date de consommation). Pour ce faire, il suffit d'ajouter les opérations suivantes :

- L'utilisation de la règle $\multimap L$ correspond à la création d'un nouveau marquage ; c'est donc lors de son utilisation que l'on calculera la date de production du marquage produit et la date de consommation des atomes consommés : ces deux dates sont égales au maximum des dates de production des atomes consommés augmenté de la durée associée à la transition franchie,
- A chaque application de la règle $\otimes L$, pour déconnecter les atomes d'un marquage, le label temporel des atomes produits est égal à celui du marquage.

Appliquons cet algorithme sur l'exemple déjà donné en Figure V-1, ici reproduit en Figure V-12.

[59] Une étape courante dans la preuve d'un séquent est une liste d'atomes correspondant à des noms de places, séparés par le méta symbole « , ».

## V.2.3. Illustration du calcul de la durée avec la logique linéaire

Afin de simplifier l'arbre de preuve, nous ne considérons que le scénario menant du marquage $p_1$ au marquage $p_4$ (la transition $t_5$ est immédiate, elle n'a pas d'influence sur la durée) et passant par les transitions $t_1$, $t_2$, $t_3$, $t_4$.

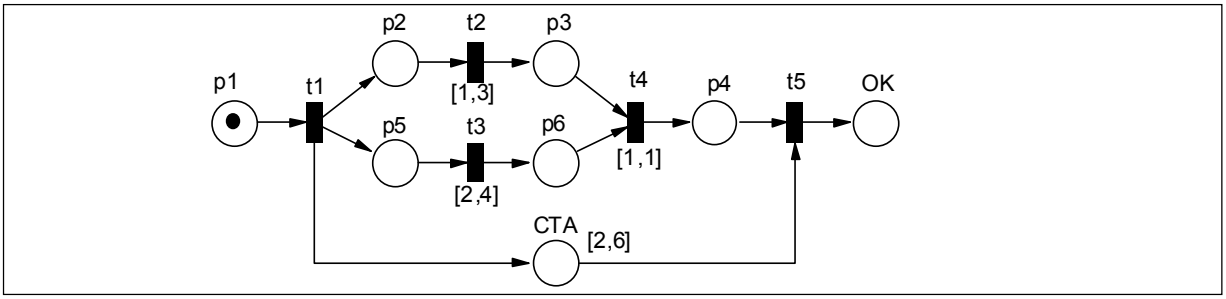


**Figure V-12 : Réseau de Petri avec parallélisme et une CTA**

Appliquons l'algorithme sur ce scénario. Voici l'écriture du RdP sous la forme de règles et de monoïdes :

$t_1$ ($d_1$) : $P_1 \multimap P_2 \otimes P_5$
$t_2$ ($d_2$) : $P_2 \multimap P_3$
$t_3$ ($d_3$) : $P_5 \multimap P_6$
$t_4$ ($d_4$) : $P_3 \otimes P_6 \multimap P_4$

Le pas initial est immédiat puisque le marquage initial n'est composé que d'un seul jeton. Ce jeton ($P_1$) est estampillé avec la valeur 0 (une autre valeur aurait pu être donnée). $t_1$ est la seule transition franchissable. Elle est donc franchie la première en utilisant la règle d'introduction à gauche de l'implication linéaire (cf. Figure V-13).

$$\dfrac{\dfrac{}{P_1 \vdash P_1}\,\mathrm{id} \qquad \dfrac{P_2(d_1), P_5(d_1), t_2, t_3, t_4 \vdash P_4}{(P_2 \otimes P_5)(d_1), t_2, t_3, t_4 \vdash P_4}\,\otimes L}{P_1(0), P_1 \multimap P_2 \otimes P_5, t_2, t_3, t_4 \vdash P_4}\,\multimap L$$

**Figure V-13 : Etape courante 1 du calcul du séquent**

Les ressources dans les prémisses sont ensuite déconnectées à l'aide de la règle d'introduction à gauche du connecteur « fois » (⊗L).

$$\dfrac{P_2(d_1, d_1 + d_2) \vdash P_2 \qquad P_5(d_1), P_3(d_1 + d_2), t_3, t_4 \vdash P_4}{P_2(d_1), P_5(d_1), P_2 \multimap P_3, t_3, t_4 \vdash P_4}\; t_2$$

**Figure V-14 : Etape courante 2 du calcul du séquent**

$t_2$ ou $t_3$ peuvent maintenant être franchies. L'ordre de franchissement de ces deux transitions ne modifiant pas le résultat final et la transition $t_2$ étant la première dans l'ordre lexicographique, elle sera franchie en premier (cf. Figure V-14). $t_3$ est maintenant tirée (cf. Figure V-15).

$$\dfrac{P_5(d_1, d_1 + d_3) \vdash P_5 \qquad P_5(d_1), P_3(d_1 + d_2), P_6(d_1 + d_3), t_4 \vdash P_4}{P_3(d_1 + d_2), P_5(d_1), P_5 \multimap P_6, t_4 \vdash P_4}\; t_3$$

**Figure V-15 : Etape courante 3 du calcul du séquent**

Il reste, pour finir la preuve du séquent, à franchir la transition $t_4$. Cette étape est détaillée (Figure V-16) afin de souligner l'utilisation de la règle d'introduction à droite du connecteur fois (⊗R).

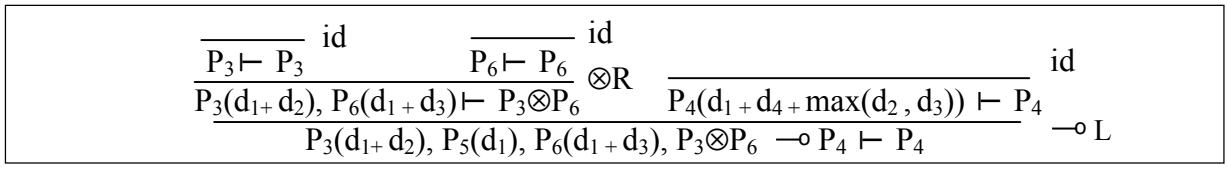


**Figure V-16 : Etape courante 4 du calcul du séquent**

Les dates de production et de consommation des différents jetons sont récapitulées dans le tableau suivant.

| Jeton | Date de production | Date de consommation |
|---|---|---|
| $P_1$ | 0 | $d_1$ |
| $P_2$ | $d_1$ | $d_1+d_2$ |
| $P_3$ | $d_1+d_2$ | $d_1+d_4+\max(d_2,d_3)$ |
| $P_4$ | $d_1+d_4+\max(d_2,d_3)$ | nc (non consommé) |
| $P_5$ | $d_1$ | $d_1+d_3$ |
| $P_6$ | $d_1+d_3$ | $d_1+d_4+\max(d_2,d_3)$ |

Nous connaissons ainsi la durée du scénario qui correspond à la date de production du jeton dans la place $P_4$ qui a pour valeur : $d_1+d_4+\max(d_2,d_3)$. Rappelons que, lors de la définition de la classe de RdP utilisée, nous avons défini les attributs *ProductionTime* et *ConsumptionTime* pour chaque jeton (cf. §III.1.4). Ce sont ces attributs qui sont mis à jour.

Bien qu'initialement présenté pour les RdP t-temporisés, le calcul de la durée des scénarii par la logique linéaire [Pradin-chézalviel-1999b] peut être utilisé avec des RdP t-temporels [Rivière-2001]. Avec les RdP t-temporels, les durées sont alors des intervalles temporels. Dans ce cas, il suffit de préciser le sens à donner à l'addition et à la fonction max.

Considérons deux intervalles temporels $d_1$ $[Min_1,Max_1]$ et $d_2$ $[Min_2,Max_2]$, la somme $d_1+d_2$ est égale à l'intervalle $[Min_1+Min_2,\ Max_1+Max_2]$, la fonction $\max(d_1,d_2)$ est égale à l'intervalle $[\max(Min_1,Min_2),\ \max(Max_1,Max_2)]$.

En appliquant ces dernières sur la formule $(d_1+d_4+\max(d_2,d_3))$ du scénario étudié, nous obtenons l'intervalle temporel [3,5]. Contrairement au résultat obtenu à l'aide du graphe de classe, cette durée est exacte. En outre, il est plus simple de raisonner sur l'expression symbolique du scénario afin d'identifier l'impact de la modification de la durée d'une transition. Par exemple, si nous voulions modifier la durée de ce scénario, il serait inutile de modifier la CT de $t_2$ si $t_3 > t_2$, puisque c'est la borne maximum des intervalles temporels de ces deux transitions qui est considérée.

Présentons maintenant les difficultés rencontrées lors du calcul du scénario par la logique linéaire. Ces difficultés apparaissent lorsqu'il y a un conflit de choix entre des transitions ou des jetons. Le scénario est alors dit incomplètement spécifié. Les circonstances dans lesquelles ce cas peut se produire sont détaillées ci-après.

## V.2.4. Difficultés pour le calcul de la durée d'un scénario incomplètement spécifié

La durée d'un scénario complètement spécifié est établie grâce à un seul arbre de preuve de logique linéaire. Cependant, des conflits de transitions ou des conflits de jetons entraînent une explosion combinatoire des arbres de preuve à construire. Nous allons expliquer pourquoi.

### V.2.4.1. Conflit de transitions

Un conflit de transition apparaît lorsque le réseau comporte au moins une place ayant plus d'une transition en sortie et si ces transitions appartiennent toutes au scénario. Dans ce cas, il y a indéterminisme sur le choix des transitions à franchir. Notons que, si cet indéterminisme peut être levé par examen des attributs temporels portés par le jeton, alors il n'y a plus de conflit. Dans le cas contraire, il faut établir plusieurs arbres de preuve correspondant chacun à un ordre donné pour le franchissement des transitions en conflit. C'est ensuite à l'analyste-concepteur de vérifier, par l'interprétation du RdP, si chacun de ces arbres de preuve correspond à un scénario qu'il voulait étudier.

### V.2.4.2. Conflit de jetons

Un conflit de jetons se produit lorsque, dans une place, plusieurs jetons de différentes provenances sont susceptibles d'être tirés par une transition. Cette situation ne peut se produire que dans le cas d'un RdP non sauf, dans lequel des transitions sont multi-sensibilisées. Prenons le cas d'un RdP où une place p a pour transitions d'entrées t1 et t2 et pour transition de sortie t3. Le choix d'un jeton produit par t1 pour franchir t3 induit une relation d'ordre entre t1 et t3. De même, pour un jeton produit par t2, une relation d'ordre entre t2 et t3 est induite. Les durées calculées seront bien sûr différentes. Deux arbres de preuve doivent donc être construits. Nous revenons alors à la même situation qu'avec les transitions en conflit.

### V.2.4.3. L'accessibilité et la sémantique de tir des RdP t-temporel

Les conflits de transitions et de jetons nous amènent naturellement à la notion d'accessibilité et de sémantique de tir (faible ou forte) choisie. Lorsqu'il y a conflit, plusieurs arbres de preuve sont produits, certains pouvant correspondre à des scénarii non réels. En effet, la logique linéaire ne travaillant que sur des valeurs symboliques, elle ne peut distinguer des scénarii non accessibles suivant la valeur des intervalles temporels.

Il est d'ailleurs important de noter qu'à une étape courante de l'algorithme ne correspond pas toujours un marquage courant du RdP. La présence d'un atome dans une étape courante signifie seulement qu'un jeton va transiter par la place correspondante. Le fait que deux atomes appartiennent à la même étape courante ne signifie pas qu'ils appartiendront à un même marquage. De même, il est possible que deux jetons appartenant à un même marquage ne soient jamais présents ensemble dans aucune étape courante.

Pour cette raison, la preuve d'un séquent dans un RdP t-temporel n'est plus qu'une condition nécessaire pour décider de l'accessibilité. Des travaux ont été développés afin de prendre en compte la sémantique faible dans le calcul des séquents et ainsi de décider de l'accessibilité ou non d'un marquage [Pradin-Chézalviel-2000].

La problématique de la sémantique forte commence à être abordée et des suggestions d'étude ont été proposées [Rivière-2001]. Cependant, il n'existe pas encore, à notre connaissance, de travaux traitant complètement cet aspect.

## V.2.5. Bilan sur la logique linéaire

L'utilisation de la logique linéaire permet d'obtenir une expression symbolique de la durée réelle d'un scénario complètement spécifié. Le fait de travailler sur des durées symboliques permet de raisonner plus facilement sur les contraintes temporelles, en identifiant les relations d'ordre réellement présentes dans le RdP, et ce même en présence de parallélisme. Il est alors plus simple d'identifier, dans le cas du non-respect d'une CTA, les CT à modifier, ainsi que d'établir les répercussions d'une modification. De plus, elle est applicable sur des RdP non-saufs, non bornés, présentant des cycles.

Toutefois, la logique linéaire comporte plusieurs limites. Tout comme le graphe de classe, elle reste confrontée à l'explosion combinatoire dans le cas de scénarii incomplètement spécifiés : chaque conflit de transitions ou de jetons nécessite la construction d'arbres de preuve différents. De plus, elle n'est, pour l'instant, pas utilisable pour décider de l'accessibilité du marquage de RdP basés sur une sémantique forte.

C'est essentiellement en raison de cette dernière limite que nous proposons une approche couplant l'utilisation du graphe de classe et la logique linéaire. Le graphe de classe est utilisé afin de déterminer des scénarii à étudier. La logique linéaire est ensuite employée afin de quantifier de manière symbolique la durée de séjour des jetons dans les places CTA. Cela permet de mieux comprendre les relations d'ordre entre CT, ainsi que d'appréhender les répercussions d'une modification éventuelle de l'une d'elles.

Dans ces conditions d'utilisation, l'algorithme de calcul des durées par la logique linéaire doit être légèrement adapté. Ce sont ces adaptations qui sont maintenant présentées.

## V.2.6. Algorithme UML/PNO utilisant la logique linéaire

### V.2.6.1. Principe de la démarche de vérification

Afin de mieux comprendre les raisons des adaptations de l'algorithme de calcul d'un scénario par la logique linéaire, il nous faut brièvement expliquer son utilisation dans UML/PNO. La démarche de vérification est la suivante :

- Identification, à l'aide de graphe de classe, des scénarii liés à des CTA,
- Quantification temporelle des CTA de ces scénarii par la logique linéaire,
- Analyse des résultats.

L'identification des scénarii est effectuée à l'aide du graphe de classe. Parvenir à ce résultat implique que l'analyste-concepteur ait : défini un marquage initial à analyser, identifié la ou les CTA à vérifier, transformé ces CTA en un ensemble de places et de transitions temporelles et enfin lancé le calcul du graphe de classe.

Dans son principe, cette identification reste simple : dans le cas du viol d'une CTA, les chemins menant à son non-respect sont simplement les listes de tirs séquentiels des transitions du graphe de classe menant à des marquages redoutés (issus des tirs de transitions « trop tôt » ou « trop tard » des CTA). Ces chemins peuvent être regroupés (en rassemblant les chemins ne différant que par les relations d'ordre dues à l'entrelacement) afin de décrire les scénarii (que nous appelons scénarii redoutés). Ces scénarii sans conflit de jeton ou de transition (le graphe de classe ayant été justement utilisé afin de résoudre ces questions) sont alors facilement analysables avec la logique linéaire.

### V.2.6.2. Algorithme de calcul proposé

Néanmoins, nous avons dû légèrement adapter l'algorithme de [Pradin-chézalviel-1999b] pour prendre en compte :

- L'ordre de tir des transitions qui est maintenant imposé par le scénario (et non plus basé sur un ordre lexicographique),
- Des conditions d'arrêt différentes (sur un marquage partiel donné et non plus sur la preuve finale du séquent),
- L'identification des conditions de respect des CTA impliquées dans le scénario étudié.

En effet, dans [Pradin-chézalviel-1999b], la durée du scénario est obtenue une fois que l'arbre de preuve est complètement terminé (c'est-à-dire que le marquage final est obtenu et que toutes les transitions de la liste des transitions ont été tirées). Pour notre part, nous arrêtons le calcul de

l'arbre de preuve dès que toutes les places CTA observées ont été marquées par des jetons et que ces derniers ont quitté ces places. En effet, nous utilisons la logique linéaire afin d'obtenir une durée de séjour des jetons (exprimée sous forme symbolique) dans les places CTA et non pour décider de l'accessibilité du scénario considéré (déjà prouvé par le graphe de classe).

Pour un scénario à analyser, défini par un marquage initial Mi, un marquage final partiel Mf et une liste de transitions S, l'algorithme devient alors :

```
APPLIQUER la règle ⊗L autant de fois que nécessaire (pour transformer le
marquage initial en une liste d'atomes séparés par le meta-connecteur
« , »)
TANT QUE Mf n'est pas atteint
– Appliquer la règle ⊸L à la première des transitions candidates du
scénario
– Terminer la preuve du séquent gauche généré en utilisant, si nécessaire,
la règle ⊗R
– Appliquer, si nécessaire, la règle ⊗L au séquent droit
FIN TANT QUE
```

Détaillons maintenant la quantification d'une CTA.

### V.2.6.3. Identification des conditions de respect des CTA

Afin de calculer la durée de séjour d'un jeton dans une CTA, il suffit de conserver les dates de production Dp et de consommation Dc (suite à l'utilisation de la règle $\multimap L$) des jetons arrivant et sortant de cette place. La durée de séjour du jeton dans une place CTA correspond alors à la formule Dc – Dp.

L'application de cet algorithme à l'exemple présenté en Figure V-12 est directe (avec le scénario menant du marquage $p_1$ au marquage OK et constitué des transitions $t_1$, $t_2$, $t_3$, $t_4$, $t_5$).

Connaissant la date de production du jeton dans cette place ($d_1$) et sa date de consommation ($d_1+d_4+\max(d_2,d_3)+d_5$), la CTA est définie par la formule : $d_4+\max(d_2,d_3)+d_5$.

Nous voyons ainsi l'un des avantages de l'utilisation de la logique linéaire et de l'individualisation des jetons : il est possible de conserver l'historique des différents jetons. De ce fait, nous pouvons simplifier les calculs lorsque des jetons ont un passé temporel commun. Dans l'exemple étudié, la durée de la transition t1 (d1) a pu être déduite (passé commun aux jetons avant le franchissement de la CTA) et elle n'interfère pas dans la formule obtenue.

Précisons que la valeur numérique de cette durée de temps de séjour présente ici peu d'intérêt (puisque nous savons déjà avec le calcul du graphe de classe si elle est respectée ou non). Ce qui nous intéresse, c'est de connaître les relations entre les CTP, afin d'identifier celles dont l'étude et la modification sont pertinentes.

En outre, le calcul numérique doit être utilisé avec précaution. En effet, l'opération de soustraction entre 2 dates A et B (exprimées sous forme d'intervalles [Amin,Amax] et [Bmin, Bmax] permettant d'obtenir une durée d où dmin = Bmin-Amax et dmax = Bmax-Amin) est imprécise[60]. Dans l'exemple étudié, la durée de la CTA est exacte, car l'opération de soustraction a pu être simplifiée. Dans le cas contraire, la durée sera pessimiste.

Les principes de notre démarche de vérification des CTA viennent d'être présentés sur un exemple simplifié et en ne considérant que l'analyse du comportement d'un CO. Voyons maintenant comment cette démarche est appliquée sur un ensemble de CO.

[60] Pour s'en convaincre, il suffit d'appliquer le calcul sur deux intervalles de même durée.

# V.3. Démarche de vérification UML/PNO des contraintes temporelles

## V.3.1. Principes

L'approche habituelle de vérification d'un système consiste à composer la totalité des diagrammes à RdP des objets (eBPN plus iBPN et EPN). Le RdP obtenu (que nous appelons RdP global) décrit le comportement total du système ainsi qu'une partie de son environnement. Puis, la démarche de vérification décrite précédemment est appliquée sur ce RdP global.

Toutefois, une telle approche reste difficilement envisageable sur un système de grande taille, car l'explosion combinatoire est quasiment immédiate.

Afin de la limiter, il faut pouvoir réduire la taille du RdP à analyser. Pour ce faire, nous partons des constats suivants :

- Pour vérifier une CTA donnée, par exemple un temps de réponse, il est souvent inutile d'analyser tout le RdP global. Généralement, seule une partie de ce dernier a une incidence sur la CTA.
- Par ailleurs, plutôt que de composer tous les iBPN terminaux du système, seuls les eBPN de plus haut niveau peuvent être utilisés, afin de travailler sur un RdP global simplifié et donc de taille moindre. Naturellement, il faudra s'assurer, à un moment donné au cours du développement, que la composition des iBPN des sous-CO respecte le comportement externe de leurs CO de haut niveau.

Outre la réduction du RdP à analyser, l'utilisation des eBPN présente d'autres avantages. Elle permet de vérifier le système en cours de construction en travaillant sur des représentations simplifiées de parties non modélisées du système (dont la décomposition en sous-CO n'est pas encore établie, par exemple).

La démarche de vérification peut alors aussi être utilisée comme une aide à la modélisation, afin de quantifier ou de définir des CT. En reprenant l'exemple des parties non modélisées, des sous-CTA peuvent y être définies afin de s'assurer du respect des CTA globales du système (entre les stimuli et les réponses). La démarche inverse peut aussi être appliquée. A partir de sous-CO, leur composition permet d'obtenir des CO de plus haut niveau et d'en déterminer les CT de leurs eBPN par analyse des iBPN des sous-CO.

Ces deux démarches de dérivation des CT par décomposition et composition sont souvent nécessaires lors du développement du système. Nous pouvons, à un instant donné de la modélisation, supposer qu'une CT est une CTP (généralement lorsque la décomposition des CO n'est pas encore finalisée), puis, plus tard, transformer cette CTP en CTA afin de s'assurer de son respect par les sous-CO (et vice-versa).

Avant de présenter ces techniques de vérification et de dérivation des CT sur un exemple du simulateur SIMONA, nous détaillons la génération du RdP réduit, RdP que nous nommons FPN (pour *Functional Petri Net*).

## V.3.2. Construction du FPN

### V.3.2.1. Principe du FPN

Nous avons vu que pour vérifier une CTA, il est souvent inutile d'analyser en totalité le RdP global du système car seule une partie contribue à la trace de celle-ci.

Par exemple, afin de quantifier une CTA entre l'arrivée d'un stimulus et la production d'une réponse (un temps de réponse), il suffit de connaître les parcours possibles de ce stimulus à travers les différents objets du système jusqu'à sa transformation finale en réponse. C'est la raison pour laquelle nous construisons un réseau de Petri fonctionnel (FPN) pour chaque CTA à vérifier. L'objectif est de représenter simplement tous les chemins empruntés par les réactions à un stimulus à travers les objets du système.

Ce principe de construction peut, en fait, être étendu à tous les types de CTA : le FPN ne représente plus simplement les parcours entre le moment de la production d'une donnée et sa transformation, mais le moment où un jeton franchit la transition d'initialisation de la CTA et le moment où il franchit la transition de synchronisation.

Le FPN peut donc être vu comme un RdP obtenu par composition des BPN et EPN des objets impliqués dans les transformations successives du jeton lié à la CTA et débarrassé des parties qui n'ont pas d'influence sur la durée de ces chemins. Cela mène, bien sûr, à une vue fonctionnelle du système (d'où le nom de ce diagramme à RdP).

Le FPN est généré automatiquement dès lors que les transitions d'initialisation et de synchronisation de la CTA ont été identifiées.

### V.3.2.2. Génération du FPN

L'algorithme de construction du FPN auquel est lié une CTA repose sur l'idée selon laquelle un FPN est au moins caractérisé par les transitions d'initialisation et de synchronisation de la CTA.

La transition de synchronisation de la CTA (ainsi que les places lui succédant directement) appartient donc au FPN. Les places précédentes de la transition de synchronisation feront, elles aussi, partie du FPN, puisque le tir de la transition de synchronisation dépend de la présence d'un jeton dans ces places. Il en est de même pour les transitions d'entrée de ces places. Ainsi, en remontant de proche en proche, nous allons ajouter des places et des transitions au noyau initial du FPN. Ce processus s'arrête lorsque les places et les transitions à ajouter :

- Sont déjà présentes sur le FPN.
- Ne comportent pas de place ou de transition d'entrée.
- Sont liées à la transition d'initialisation de la CTA.

Lorsqu'une place ou une transition à ajouter est associée à une communication avec un autre objet, le processus est propagé au diagramme à RdP de cet objet. Pour les BPN, cette propagation s'effectue par défaut sur le RdP de comportement externe.

Ce processus de construction s'apparente donc à une « recherche des chemins inverses » (*back tracking*[61]). Nous ne conservons que des parties de RdP liées aux chemins de la CTA et aux synchronisations nécessaires qui pourraient entraîner des attentes lors de son parcours.

Naturellement, ce FPN peut être très important (pour peu qu'il y ait un grand nombre d'objets impliqués), mais rappelons que le but du FPN est de modéliser précisément toutes les dépendances de la CTA. De plus, dans le cas où les eBPN sont utilisés, de nombreux traitements sont souvent séquentiels (les eBPN ne modélisent pas les ressources partagées) et peuvent donc être facilement agrégés. Le FPN peut contenir des sous-CTA, qui correspondent à des sous-parties du FPN (cf. Figure V-17a).

Parfois, la transition d'initialisation de la CTA n'est pas présente dans le FPN obtenu (cf. Figure V-17b). Cela indique alors qu'il n'y a pas de chemins entre la transition d'initialisation et la transition de synchronisation de la CTA. Ce cas dénote une incohérence qui résulte, soit d'une erreur dans la définition des transitions de synchronisation et d'initialisation de la CTA, soit d'un

[61] chaînage arrière.

problème de modélisation.

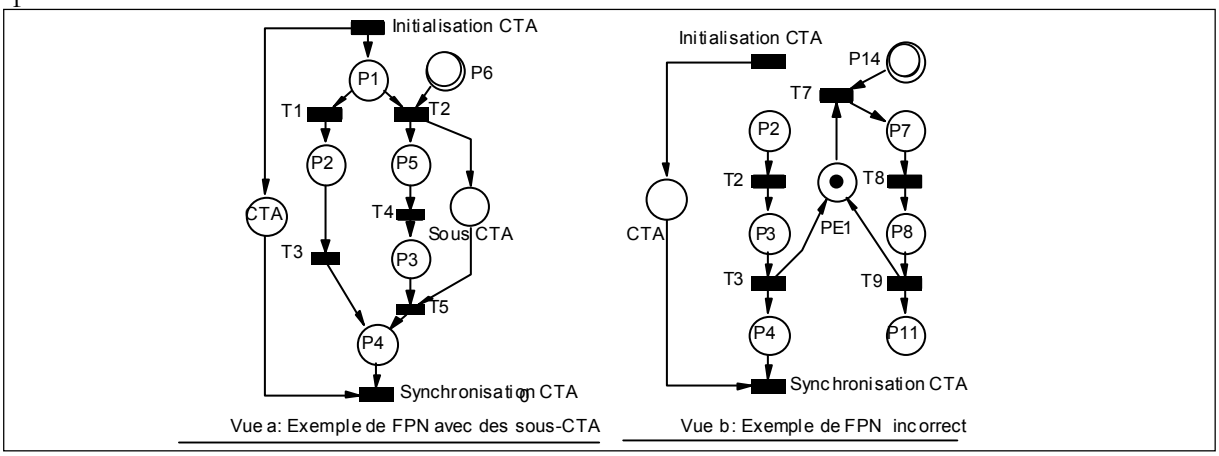


**Figure V-17 : Exemple de FPN**

### V.3.2.3. Représentation du FPN

La représentation du FPN ne diffère pas fondamentalement de celle utilisée pour les BPN ou les EPN. Signalons, néanmoins, quelques particularités :

- Puisque le FPN est une composition de parties de BPN, il est nécessaire d'identifier l'appartenance (ou l'origine) des places et transitions concernées. Cette identification se fait en ajoutant le nom du CO à ceux des places et des transitions. Exemple : soit la transition T1, contenue dans l'iBPN du CO "CO1", elle sera nommée « CO1.T1 ».
- En outre, nous associons au FPN la liste des sous-CO des BPN qui le composent. Cette liste nous permettra de vérifier et d'identifier rapidement les CO impliqués dans le respect d'une CTA. De plus, cela permet, sur demande de l'analyste-concepteur, de remplacer les traitements supposés de cet eBPN par les traitements réels de son iBPN.
- Enfin, les places et les transitions de communication partagées entre BPN et/ou EPN sont agrégées en une seule place ou en une seule transition (suivant la représentation de la communication utilisée). Dans ce cas, la place ou la transition porte le nom de l'objet serveur (celui qui offre l'opération).

L'utilisation d'un FPN, ainsi que des techniques associées, va être présentée sur l'exemple du simulateur.

## V.3.3. Vérification temporelle d'une CTA du simulateur

### V.3.3.1. Présentation de la partie du SRS étudiée

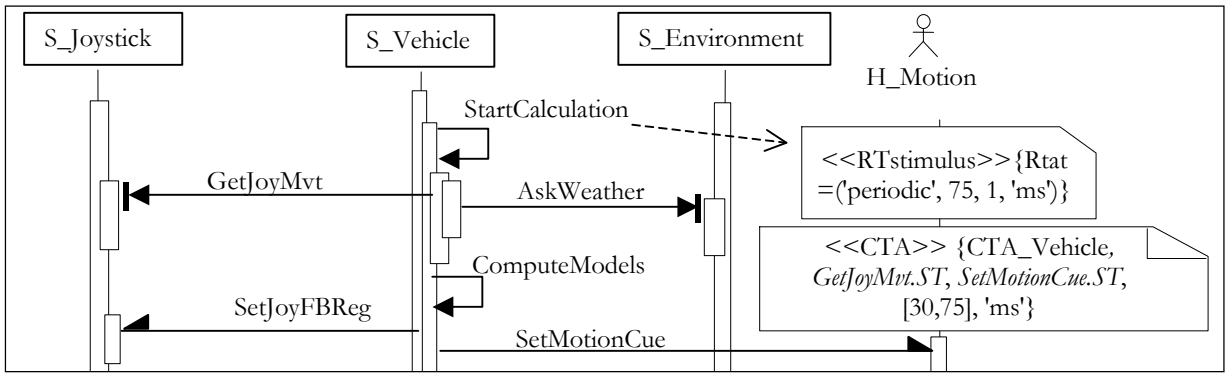


**Figure V-18 : CTA étudiée pour *S_Vehicle***

L'exemple du simulateur de vol, maintenant étudié, reprend les éléments abordés dans les

parties précédentes de ce mémoire. L'objectif principal est de vérifier l'une des CTA (*CTA_Vehicle*) du CO *S_Vehicle* du SRS (cf. Figure V-18). Afin de ne pas complexifier inutilement l'exemple, nous avons considéré que la commande de déplacement (*SetMotionCue*) est envoyée directement par le CO *S_Vehicle* à la cabine hydraulique (sans passer par le CO *S_Motion*). Sa modélisation se déroule dans une phase intermédiaire de la spécification. Les comportements internes et externes de *S_Vehicle* et de *S_Joystick* viennent d'être modélisés. Seul le comportement externe de *S_Environment* est connu.

### V.3.3.2. Diagrammes à RdP des CO étudiés

L'eBPN de *S_Vehicle* est donné afin de rappeler les principales communications de ce CO (cf. Figure V-19). Toutes les 75 ms, deux demandes (mouvement du joystick et données aérodynamiques) sont effectuées. Sur réception de ces données, le mouvement de la cabine est généré, ainsi que, parfois, une modification de la consigne de régulation du retour de force du joystick (d'où la présence de l'alternative *Choice1* sur Figure V-21).

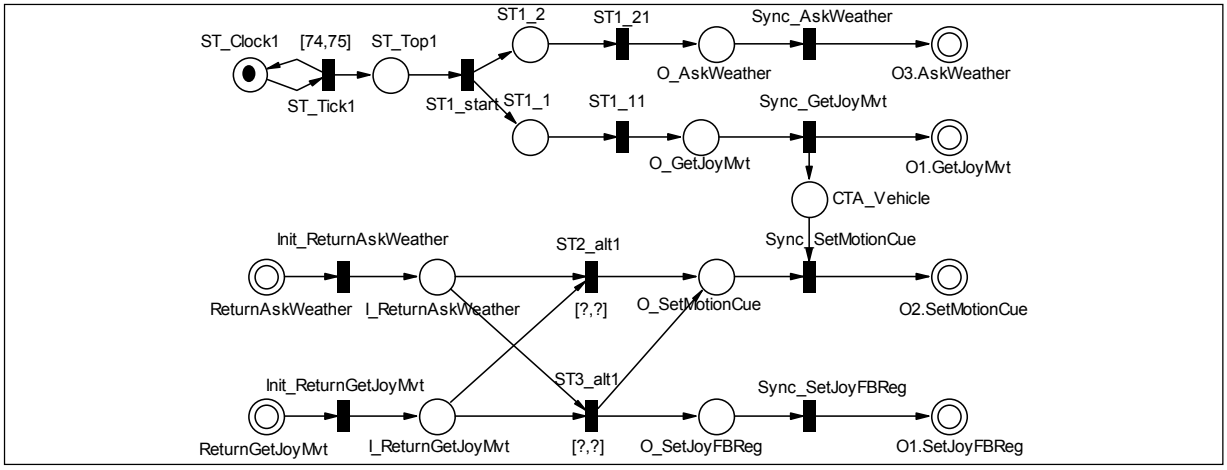


**Figure V-19 : eBPN de *S_Vehicle***

La Figure V-20 détaille l'iBPN complet de S_*Vehicle*. Une partie de ses activités composites a été décomposée, afin de modéliser explicitement les communications avec les autres objets.

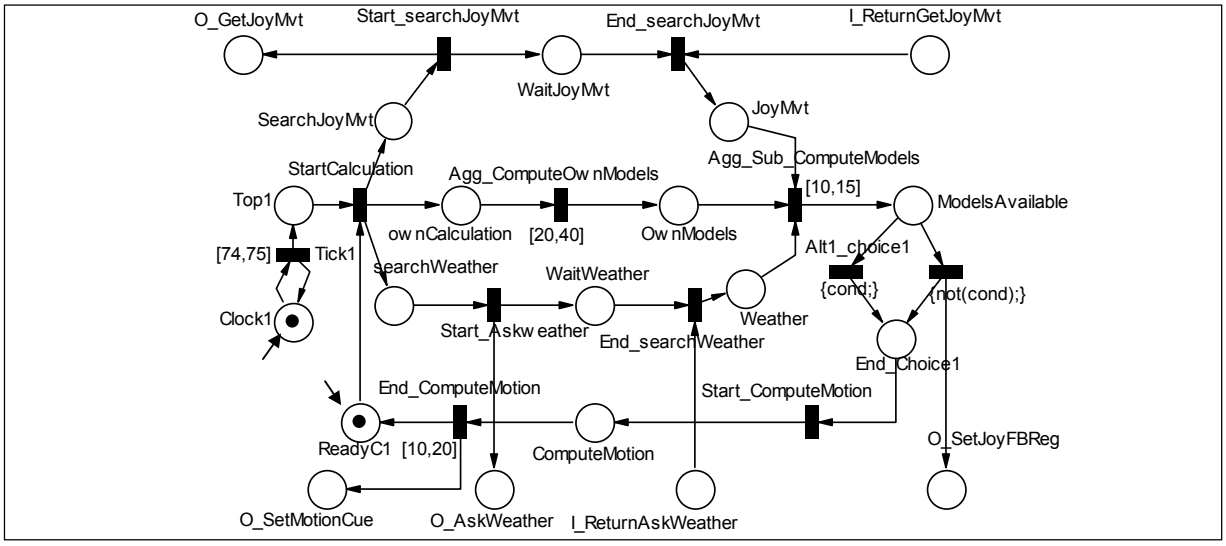


**Figure V-20 : Partie nominale de l'iBPN de *S_Vehicle***

Toutes ses CT internes sont connues. Le calcul interne de préparation des modèles mathématiques du véhicule (*Agg_Compute_own_models*) prend entre 20 et 40 ms. La durée de compilation et de synthèse de toutes les données reçues (mouvement du joystick, données

aérodynamiques…) est comprise dans l'intervalle de temps [10,15]. La génération de la commande (*MotionCue*), à envoyer à la cabine, dure entre 10 et 20 ms.

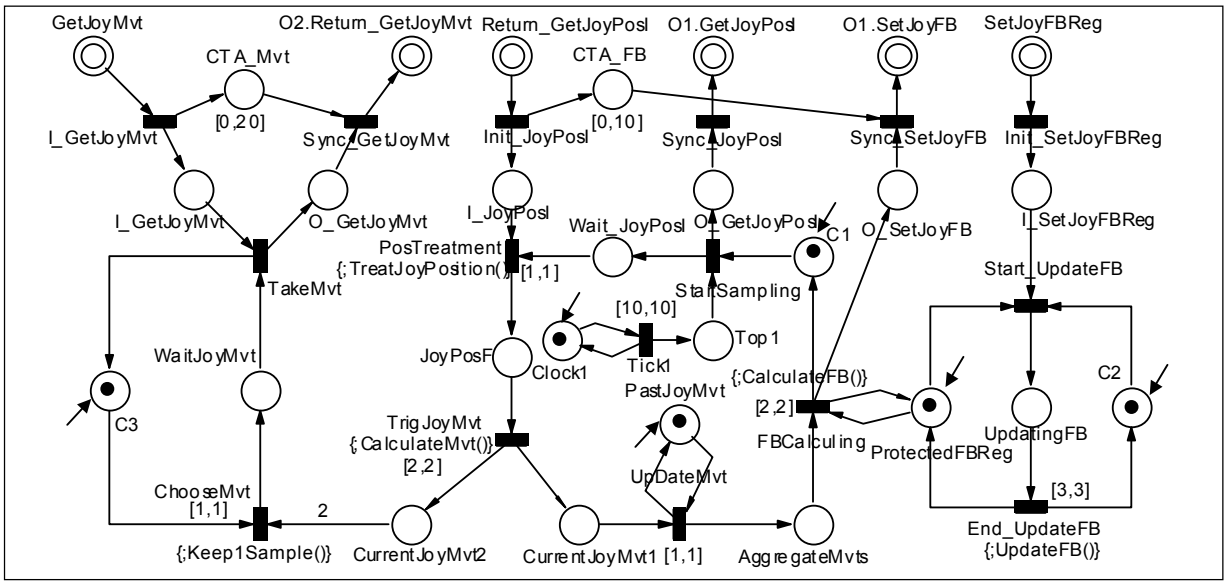


**Figure V-21 : Partie nominale du BPN complet de *S_Joystick***

Nous donnons le comportement complet (composition de son iBPN et de son eBPN) du Joystick en Figure V-21. Toutes ses CTP et CTA ont été définies.

### V.3.3.3. Diagramme à RdP global des CO étudiés

Si nous devions vérifier la CTA sans utiliser la notion de FPN, une partie du RdP global à analyser serait celui donné en Figure V-22. Ce RdP contient les comportements complets (définis à ce stade de la modélisation) des objets impliqués par le calcul de cette CTA.

La complexité de ce RdP et sa grande taille font que le calcul du graphe de classe comporte rapidement un très grand nombre d'états. Avec l'outil Tina de calcul de graphe de classe, nous avons directement été confrontés à une explosion combinatoire.

En outre, des interdépendances entre le joystick et le véhicule peuvent être constatées. La plus gênante est liée à la modification de la consigne de régulation (*Joy.Set_JoyFBReg*) par le véhicule. Cette modification peut alors retarder le prochain temps de calcul du mouvement du Joystick. En effet, la consigne de régulation est une ressource (*protectedFBReg*) utilisée pour le calcul du retour de force. Ce dernier est inclus dans le même cycle que celui du mouvement du joystick, dont la durée a une influence sur le calcul du mouvement de la cabine élaboré par le véhicule. La définition du marquage initial devient alors difficile afin de pouvoir traiter ces différentes situations.

La vérification de la *CTA_Vehicle* sur un RdP global prenant en compte tous les diagrammes à RdP de tous les objets, reste donc complexe, voire non envisageable (si une explosion combinatoire est atteinte). La génération du FPN va nous permettre de réduire la taille et la complexité du RdP à analyser. Un autre avantage de l'utilisation des eBPN est de différer le traitement de certaines interdépendances entre objets, ce qui permet un traitement progressif des problèmes rencontrés.

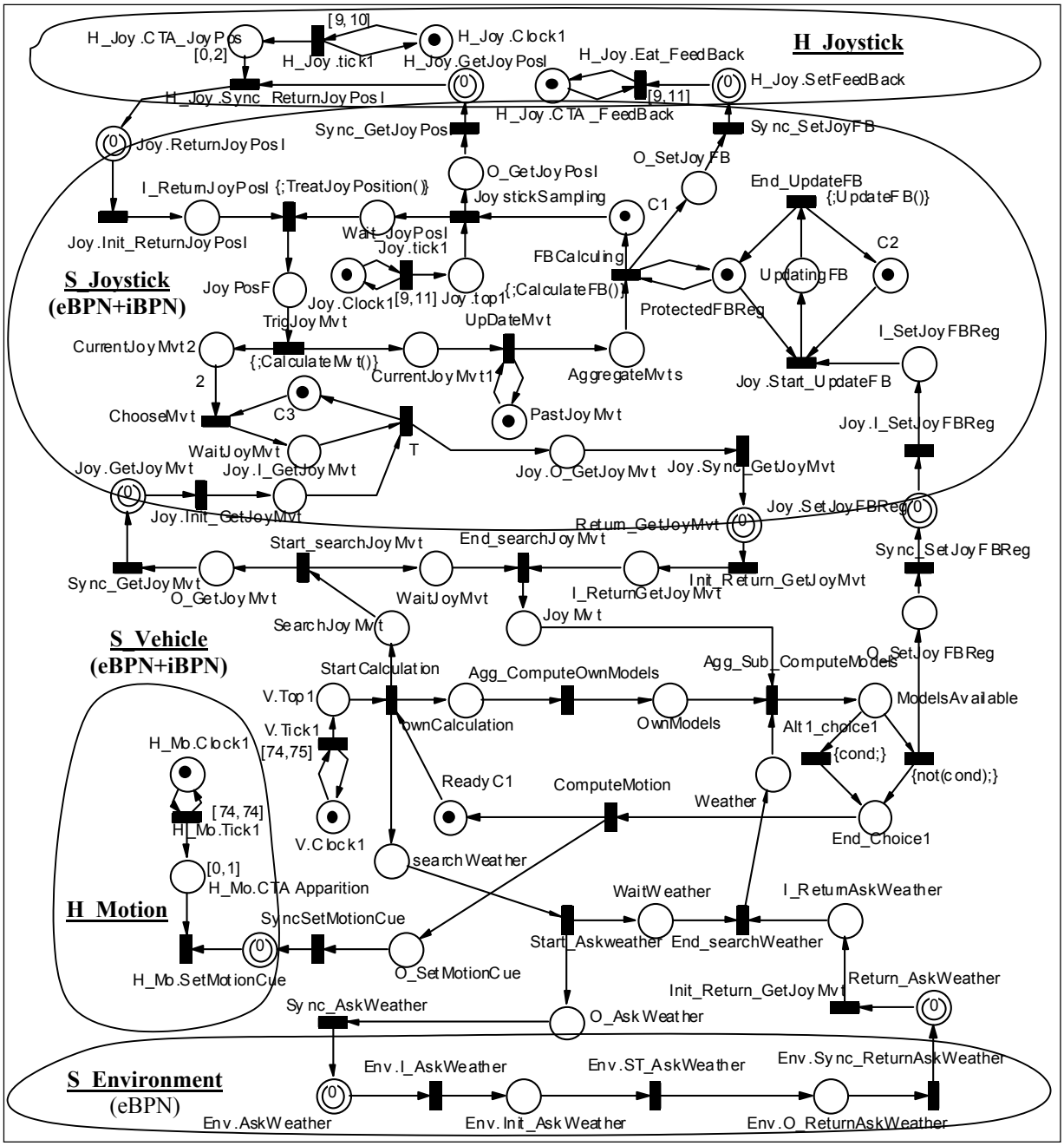


**Figure V-22 : Partie du RdP global du simulateur**

### V.3.3.4. Construction du FPN associé à la *CTA_Vehicle*

Le FPN lié à la CTA étudiée contient les diagrammes à RdP des objets participant à la génération de la réponse *SetMotionCue* (CO *S_Joystick*, CO *S_Environment* et l'acteur *H_Motion*) ainsi que le comportement complet (eBPN plus iBPN) de *S_Vehicle* (cf. Figure V-23).

Ce RdP est d'une taille inférieure à celle du RdP global présenté en Figure V-22.

Il peut être encore simplifié. En effet, en raison du principe de construction des eBPN, de nombreux traitements séquentiels (suites de places et de transitions) sont présents et peuvent être agrégés en une simple transition (cf. partie IV). Nous obtenons alors le FPN agrégé donné en Figure V-24.

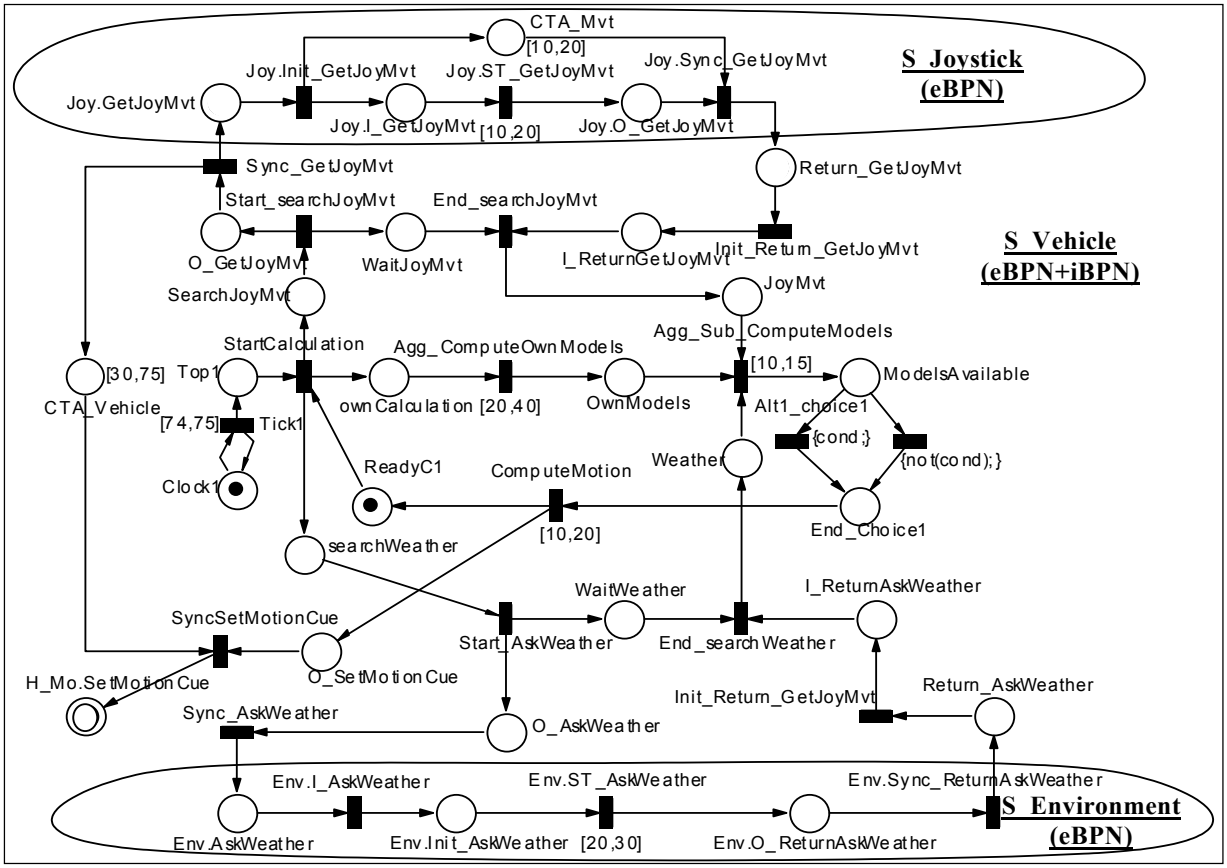


**Figure V-23 : FPN lié à la *CTA_Vehicle***

En outre, puisque nous ne travaillons que sur l'eBPN de *S_Joystick*, l'interdépendance entre le calcul du mouvement du joystick et la modification de la consigne de régulation n'apparaît plus.

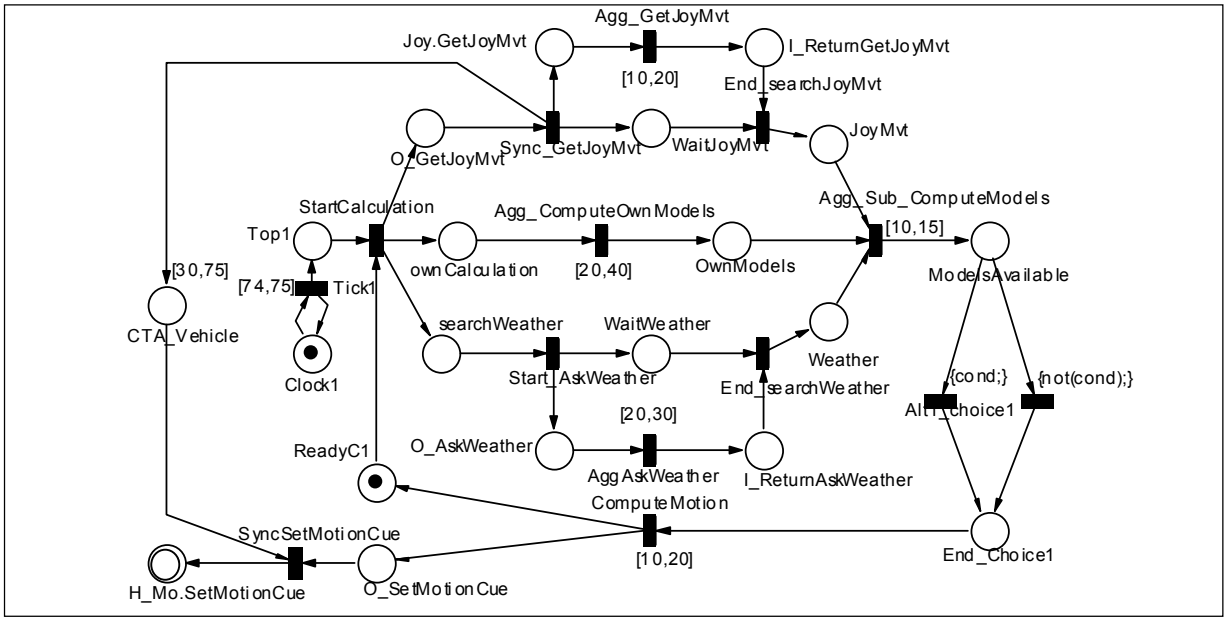


**Figure V-24 : FPN agrégé de *CTA_Vehicle***

En effet, la durée de la transition *Joy.ST_GetJoyMvt* est supposée correcte et la CTA (*CTA_Mvt*) qui lui est associée est respectée. Ce sera lors de la vérification de cette dernière que l'interdépendance sera prise en compte.

### V.3.3.5. Calcul du graphe de classe

Pour calculer le graphe de classe, il faut identifier les situations à vérifier, ce qui revient à définir le marquage initial du FPN. Dans le cas présent, la CTA à étudier est liée au cycle de calcul de *S_Vehicle*. Si cette CTA est respectée sur un cycle, alors elle sera toujours respectée. Nous pouvons donc n'étudier le FPN que sur un seul cycle. Le comportement périodique (les places et transitions horloge) peut alors être réduit à la seule place *Top1* de la Figure V-24. Le système est considéré en mode nominal et déjà initialisé (le marquage initial est défini comme indiqué en Figure V-24).

Le calcul du graphe de classe permet de vérifier le respect de la CTA. Sur le graphe obtenu (contenant 57 classes d'états et 92 transitions), différents chemins sont possibles, mais ils peuvent être rassemblés sous deux scénarii. Ils correspondent bien à un cycle de calcul de *S_Vehicle* où chaque transition du RdP est tirée une et une seule fois (à l'exception des alternatives *altX_choice1* qui sont mutuellement exclusive). Afin de comprendre les conditions de respect de la CTA, la logique linéaire est utilisée pour ces deux scénarii. L'expression symbolique obtenue est la même puisque leur différence concerne les transitions des alternatives qui ont la même durée.

### V.3.3.6. Quantification de la CTA et analyse des résultats

Pour des raisons de place, l'arbre de preuve de la logique linéaire pour l'évaluation du scénario n'est pas donné. L'expression algébrique de la durée de la CTA obtenue est la formule :

$Max(d_{Agg\_AskWeather} + d_{Start\_AskWeather} + d_{End\_SearchWeather}, d_{Agg\_ComputeOwnModels}, d_{Agg\_GetJoyMvt} + d_{Sync\_GetJoyMvt} + d_{End\_searchJoyMvt}) + d_{Agg\_Sub\_ComputeModels} + d_{Alt1\_Choice1} + d_{ComputeMotion} + d_{Start\_Calculation} + d_{SyncSetMotionCue}$

Cette formule peut encore être simplifiée par suppression des transitions immédiates. Nous obtenons alors l'expression :

$Max(d_{Agg\_AskWeather}, d_{Agg\_ComputeOwnModels}, d_{Agg\_GetJoyMvt}) + d_{Agg\_Sub\_ComputeModels} + d_{ComputeMotion}$.

Elle permet ici de mieux comprendre et d'identifier les conditions de respect de cette CTA. Ainsi, on constate que les temps de calcul du mouvement du joystick (*Agg_GetJoyMvt*) et des données aérodynamiques (*Agg_AskWeather*) n'interfèrent pas avec la CTA tant qu'ils restent inférieurs à la durée de calcul interne (*Agg_ComputeOwnModels*) de préparation des modèles mathématiques du véhicule.

Par ailleurs, d'autres calculs peuvent être effectués sur les sous-CTA (fictives ou réelles) présentes sur le FPN. Un exemple typique de CTA fictive est la quantification d'une CTP. Ainsi, dans le cas de la CTP de la transition *ST2_alt1* de l'eBPN du véhicule (cf. Figure V-19), l'intervalle de temps est inconnu. Il suffit alors de poser une CTA fictive (*CTA_virtual*) en relation avec cette transition et d'utiliser la logique linéaire pour quantifier son expression (cf. Figure V-25).

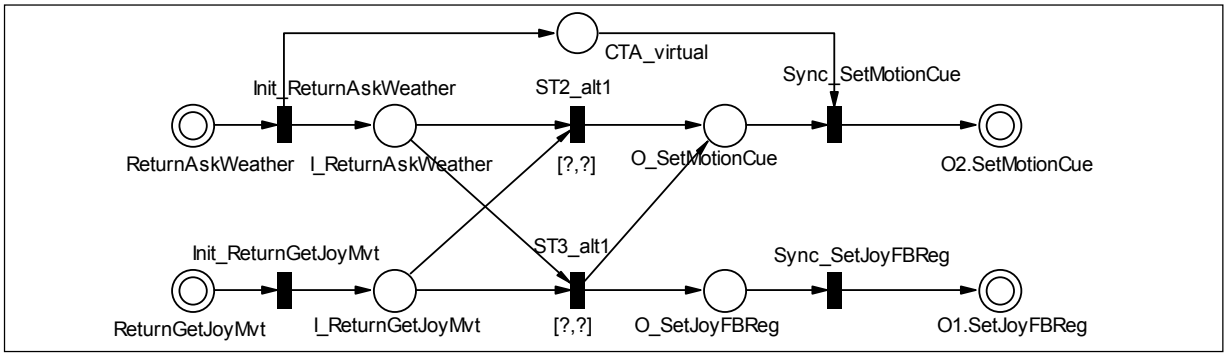


**Figure V-25 : Dérivation de CT**

Ce calcul ne peut être effectué que si cette CTA correspond à une sous-CTA en relation avec

un FPN existant (dans le cas contraire, les scénarii ne peuvent pas être établis à l'aide du graphe de classe). La durée de cette place CTA fictive est simplement la différence entre la date de production du jeton dans la place *SetMotionCue* (qui est la durée de la *CTA_Vehicle*) et la date de consommation du jeton de la place *ReturnAskWeathe*r, valant Max($d_{Agg\_AskWeather}$ , $d_{Agg\_ComputeOwnModels}$ , $d_{Agg\_GetJoyMvt}$). Cette différence est simplifiable et donne la formule $d_{Agg\_Sub\_ComputeModels} + d_{ComputeMotion}$. Le passage aux valeurs numériques est possible et donne un résultat exact. La CTP *ST2_alt1* vaut [20,35].

Naturellement, une utilisation manuelle de cette démarche de vérification, mêlant graphe de classe et logique linéaire, devient rapidement inenvisageable sur des cas plus complexes. Pour cette raison, nous avons commencé son implémentation dans un Atelier de Génie Logiciel (AGL).

# V.4. Un prototype d'AGL UML et RdP

## V.4.1. Motivation et choix

Afin d'aller plus loin dans la validation de l'approche UML/PNO, il nous est apparu indispensable de commencer son automatisation. En effet, que ce soit pour traiter des cas d'étude plus complexes ou pour tester la pertinence des aides proposées, l'utilisation d'un outil devient nécessaire. Bien que nous voulions, dans un premier temps, nous focaliser sur l'automatisation de la démarche de vérification des CT, nous avons choisi de travailler conjointement sur l'aspect UML et sur celui des RdP, afin de tester plus rapidement nos propositions concernant les apports mutuels offerts par ces deux formalismes. Il s'agissait donc de développer un AGL UML permettant l'utilisation des RdP.

Un tel outil n'existant pas, nous avons débuté son implémentation. Plutôt que de tout développer, nous avons recherché des AGL libres afin d'y ajouter les spécificités UML/PNO. Parmi ces derniers (Alma [Desnoix-2001], KUML [Groupe-2001a], Dia [Alla-2000], Umbrello [Groupe-2001b]…), ArgoUML [Robbins-1998] nous a semblé le plus intéressant, car :

- Il couvre la totalité des diagrammes UML.
- Il est basé sur un dépôt de modèles respectant le méta-modèle UML 1.3.
- Il dispose d'un système d'assistance, analysant en permanence la modélisation et proposant des conseils, sous forme de questions guidées, à l'analyste-concepteur. Ce système nous a semblé intéressant pour implémenter le processus de dérivation UML/PNO.
- Enfin, il propose un système de modules permettant d'ajouter des outils (essentiellement de génération de code). Nous pouvions l'utiliser pour ajouter simplement un éditeur à RdP ainsi que les outils de vérification associés.

## V.4.2. Le module ArgoPNO

Nous avons donc réalisé un module, nommé ArgoPNO [Delatour-2003], implémentant une partie de l'approche UML/PNO dans l'AGL ArgoUML. Ce module a été développé à l'Ecole Supérieure d'Electronique de l'Ouest avec l'aide de trois stagiaires (Florent De Lamotte, Frédéric Jouault et Florian Gardes).

### V.4.2.1. Les outils de vérification proposés dans ArgoPNO

La Figure V-26 est une capture d'écran d'ArgoPNO. On peut y reconnaître une partie du FPN de la *CTA_Vehicle* vérifiée dans le chapitre précédent. L'éditeur de diagrammes à RdP permet la modélisation des classes de RdP pt-temporel et un support pour les RdP colorés est en cours de développement. Différents formats de sauvegarde des RdP, basés sur XML, sont possibles. Cela nous permet d'utiliser des outils existants d'analyse de RdP.

Ainsi, lors de la vérification des CT, ArgoPNO construit automatiquement les FPN et les envoie aux outils (Romeo [Lime-2003] ou Tina [Berthomieu-2001]) de calcul des graphes de classe. A partir de ces derniers, il est alors possible de mener la quantification des scénarii dans ArgoPNO. En effet, nous avons réalisé un outil de calcul de l'arbre de preuve de la logique linéaire, intégré sous la forme d'un module. C'est d'ailleurs avec celui-ci que nous avons mené les vérifications de la *CTA_Vehicle* (cf. Figure V-26). Des fonctions de simplification des expressions symboliques ont été aussi implémentées.

Des développements sont en cours afin de pouvoir plus facilement visualiser et gérer les scénarii issus du graphe de classe. Des interfaces avec des outils de model-checking comme Aldebaran [Fernandez-1996] (afin de simplifier les graphes de classe obtenus) sont à l'étude.

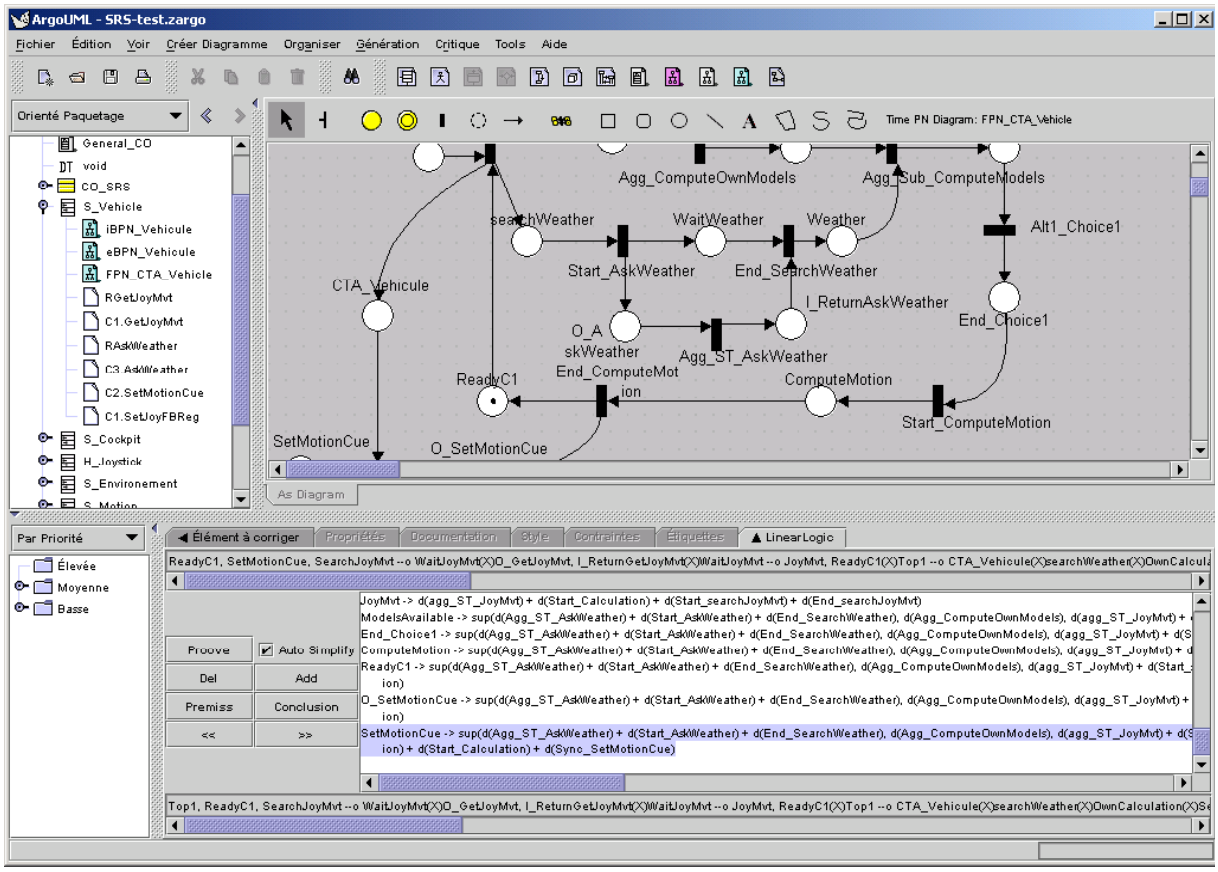


**Figure V-26 : Capture d'écran d'ArgoPNO**

### V.4.2.2. Le couplage UML et RdP

Nous avons commencé l'implémentation de la notion de CO et de CC, ainsi que des liens entre les diagrammes à RdP et la modélisation UML. En revanche, les dérivations d'UML vers les RdP ne sont pas encore abordées.

Ce développement a demandé la définition de méta-modèles pour les CO et pour les diagrammes à RdP, afin d'utiliser le dépôt de modèles d'ArgoUML. Le premier de ces méta-modèles n'est pas achevé et ne respecte pas totalement la syntaxe d'UML 1.3. Il spécialise bien les notions de sous-système et d'interface d'UML, mais ajoute un certain nombre de mécanismes non UML 1.3 (par exemple la notion d'instance de sous-système est non définie par la norme), mais présents dans les versions ultérieures d'UML.

Pour le second méta-modèle (celui des RdP), un certain nombre de compromis a été adopté. Nos travaux n'étant pas suffisamment avancés sur cet aspect d'une représentation des RdP dans le méta-modèle UML, nous étendons pour l'instant les RdP à partir de la classe *ModelElement*

(classe mère de tous les éléments du méta-modèle UML) sans utiliser d'autre notion UML d'état et d'action. Nous avons donc dû introduire des concepts externes à UML pour définir, par exemple, la notion de marquage. Pour toutes ces raisons, le méta-modèle de RdP actuellement implémenté dans ArgoPNO n'est pas compatible avec la notion de profil UML.

Néanmoins, afin que les diagrammes à RdP puissent se lier à des éléments UML, leurs places et transitions référencent des *Features* (des attributs et des opérations UML) provenant de l'entité (un *Classifier*) dont le RdP décrit le comportement. De ce fait, les diagrammes à RdP peuvent être utilisés dans ArgoPNO comme, par exemple, des diagrammes d'états/transitions.

# V.5. Résumé sur la vérification des contraintes temporelles

Une approche de vérification des CT, basée sur l'utilisation conjointe du graphe de classe et de la logique linéaire, a été présentée. Elle s'appuie sur une génération partielle des RdP à analyser (les RdP fonctionnels), ne prenant en compte que les parties ayant une incidence sur la CT en cours de vérification. De plus, cette génération peut s'effectuer sur les RdP de comportement externe, ce qui permet d'étudier des parties simplifiées ou en cours de modélisation. Cela offre l'avantage de travailler sur des RdP de tailles « adaptables » afin de mieux gérer l'inévitable explosion combinatoire des états.

Cette démarche de vérification peut aussi être utilisée comme une aide à la modélisation afin de dériver des CT : la logique linéaire, en fournissant des expressions symboliques, permet de mieux comprendre les relations d'ordre entre CT et de résoudre ainsi plus facilement les non-respects de CT.

Néanmoins, cette démarche de vérification présente encore des lacunes. La principale vient de la difficulté de son automatisation. En effet, bien que nous ayons implémenté dans un AGL UML une bonne partie de cette approche de vérification des CTA (calcul du graphe de classe, quantification des scénarii par la logique linéaire…), de nombreuses opérations doivent être guidées par un analyste-concepteur. Que ce soit pour la transformation des CTA et des comportements périodiques ou pour l'analyse des chemins du graphe de classe et leurs regroupements en scénarii, l'expertise d'un spécialiste en RdP est requise.

Une autre limite vient de la définition du marquage initial des RdP fonctionnels qui permet d'établir la situation à vérifier. En théorie, la modélisation du comportement des acteurs (les EPN) permet d'établir une partie de ce marquage initial en définissant la charge du système (en terme de flux de production des stimuli). Néanmoins, cela suppose que le développement soit suffisamment avancé pour que la modélisation des EPN soit effectuée. Même dans ce cas, cette modélisation de l'environnement est parfois insuffisante (le procédé est trop complexe ou mal connu) pour une détermination de la charge du système. En outre, l'activité du système est parfois indépendante de l'environnement, car initiée sur des comportements périodiques internes. Une aide à cette définition du marquage pourrait être apportée lors de la construction du FPN par chaînage arrière du marquage suivant la démarche proposée dans [Khalfaoui-2002]. D'autres possibilités d'améliorations sont envisagées et sont actuellement en cours d'étude. Elles sont présentées dans la partie prospective de ce mémoire.

# Partie VI Conclusion générale

## VI.1. Contributions

Dans ce mémoire, nous avons montré l'intérêt et la faisabilité de l'utilisation conjointe d'UML et des RdP pour la modélisation de STR. Ces deux formalismes sont très complémentaires. La notation UML est employée comme un langage semi-formel commun afin de faciliter le dialogue et la communication entre les différentes équipes intervenant sur le développement du système. Elle est utilisée pour la représentation de toute la partie statique du système et apporte des aides précieuses à la structuration, deux aspects pour lesquels les RdP sont peu adaptés. En revanche, l'utilisation des RdP comble une partie des lacunes d'UML, tant par leur richesse de description des aspects dynamiques (et tout particulièrement des contraintes temporelles) que par la formalisation de la modélisation. Il est alors possible de mener des activités de vérification et de validation.

### VI.1.1. Extensions proposées

Nous avons proposé des aménagements de la notation UML afin de l'adapter au domaine étudié. Les concepts d'objets et de classes composés (les CO et les CC) ainsi que leurs contrats ont été définis. Ils spécialisent les notions UML de sous-système et d'interface afin de les adapter à une approche orientée composant actif. Ils permettent la définition d'un composant en prenant en compte ses spécificités temps réel (protocoles de communication, contraintes temporelles, effets observables…).

Concernant l'expression des CT, une représentation unifiée a été proposée. Elle distingue les CT à vérifier (les CTA) des CT imposées (les CTP). Cette proposition est d'ailleurs compatible avec le profil récemment défini : « *UML Profile for Schedulability, Performance and Time* » [OMG-2002d].

Ces extensions d'UML (qui pourraient être réunies dans un profil « UML/PNO ») ont été déterminées afin de compléter la description des aspects comportementaux offerte par les RdP.

Ces derniers sont couplés à UML en tant que nouveau diagramme, nommé diagramme à RdP. Afin de différencier les CTA des CTP, ce dernier repose sur une classe de RdP à places et transitions temporelles. Trois utilisations différentes de ce diagramme ont été proposées :

- Les BPN (Behavioural Petri Net) modélisent le comportement d'une classe composée ou d'un objet composé. Ils proposent deux représentations différentes d'une même entité selon une vue externe (eBPN) ou interne (iBPN). Ils permettent ainsi d'obtenir des vues plus ou moins détaillées du comportement des CO et des CC.
- Les EPN (Environmental Petri Net) représentent une partie du comportement des entités de l'environnement (les acteurs au sens UML), ainsi que des médias de communication. Ils sont essentiellement utilisés à des fins de simulation et de vérification.
- Les FPN (Functional Petri Net) décrivent un ensemble de chemins possibles liés à une CTA. Ils sont utilisés à des fins de vérification temporelle et d'aides à la modélisation. Ils sont construits automatiquement à partir des BPN et des EPN.

### VI.1.2. Les dérivations UML vers les RdP

Afin de faciliter le couplage entre UML et les diagrammes à RdP, des règles de « traduction » entre ces deux formalismes ont été proposées. Elles portent sur les diagrammes de séquence, de collaboration et d'activités.

Nous avons préféré suivre un processus de dérivation semi-automatique. En effet, une représentation UML peut parfois être interprétée de diverses manières. Une dérivation automatique, choisissant systématiquement une traduction donnée, risquerait d'introduire de mauvaises interprétations des besoins. C'est la raison pour laquelle, lorsqu'il y a ambiguïté, l'intervention de l'analyste-concepteur est requise. De même, puisque tout passage d'une représentation semi-formelle vers une représentation formelle nécessite des précisions et des compléments d'informations (c'est d'ailleurs là un intérêt supplémentaire à l'emploi d'un modèle formel), l'analyste-concepteur doit compléter les informations manquantes.

De ce fait, le processus de dérivation peut être considéré comme un véritable assistant à la modélisation guidant, en quelque sorte, l'analyste-concepteur à l'aide d'un ensemble de questions lors de l'enrichissement de la modélisation.

## VI.1.3. Des aides à la validation et à la vérification

Au cours de ces dérivations, des vérifications de la cohérence des aspects dynamiques de la modélisation UML ont été effectuées. En effet, les RdP permettent de rassembler, sur une représentation complète et unifiée, les informations disséminées dans plusieurs types de diagrammes UML. Il est alors possible de détecter des incohérences entre des diagrammes UML contradictoires. De plus, la recherche des propriétés dynamiques et statiques des RdP offre des aides supplémentaires à l'analyste-concepteur pour juger du bien-fondé de sa modélisation. Toujours dans un contexte de validation, la simulation des RdP permet une meilleure compréhension de la modélisation UML. Une retranscription de la trace de ces simulations sous forme de diagrammes de séquence est d'ailleurs possible.

Concernant la vérification, nous proposons une approche couplant l'utilisation du graphe de classe et de la logique linéaire afin de vérifier le respect des CTA. L'utilisation de la logique linéaire permet de mieux comprendre les relations d'ordre entre CT dans le cas, par exemple, d'un non-respect des CT. En effet, des conditions symboliques (par quantification de la durée des scénarii redoutés) sur les durées de traitements peuvent être calculées. Par ailleurs, le fait de pouvoir disposer de vues partielles et simplifiées du système (grâce aux eBPN et à la notion de CO) permet de retarder l'inévitable explosion combinatoire des états. Grâce à l'identification des schémas de synchronisation des CT par la logique linéaire, des aides sont proposées concernant leurs compositions et décompositions.

## VI.1.4. Le prototype ArgoPNO

Une partie de l'approche UML/PNO a été implémentée sous forme d'un module, nommé ArgoPNO [Delatour-2003], dans l'AGL ArgoUML. Cet outil, encore à l'état de prototype, peut, d'ores et déjà, être utilisé pour spécifier un système suivant l'approche UML/PNO.

Une première implémentation (des CO, CC, liens « uses », « calls », « in » et « out ») bien qu'incomplète, permet déjà une structuration de la modélisation. Il est ainsi possible d'établir des liens entre la modélisation UML et les RdP. Par exemple, un diagramme à RdP peut décrire le comportement d'un CO/CC et faire référence aux opérations offertes par ce dernier.

Le support des diagrammes à RdP est, pour sa part, plus complet. Un éditeur de diagrammes à RdP, différents formats de sauvegarde vers des outils d'analyse des RdP et une automatisation partielle de la démarche de vérification (incluant une génération des FPN ainsi que la quantification symbolique de scénarii par la logique linéaire) permettent de mener des activités de vérification des CT.

Néanmoins, ArgoPNO est loin d'être achevé. Les efforts actuels de développement se portent sur les RdP et les techniques de vérification. Un support plus complet des graphes de classe (en particulier sur leur visualisation et sur la détermination des scénarii redoutés) se poursuit. Un

joueur à RdP pt-temporel est toujours en développement.

Les procédés de dérivation d'UML vers les RdP ainsi que les détections d'incohérences n'ont pas du tout été abordés. Nos travaux théoriques dans ce domaine ne sont pas assez avancés pour en permettre une automatisation. Cet aspect est d'ailleurs l'une de nos prochaines perspectives de travail.

# VI.2. Prospectives

Les travaux d'amélioration actuellement envisagés portent sur deux axes, dont les avancées potentielles seront répercutées dans le développement d'ArgoPNO.

## VI.2.1. Une utilisation plus grande de la logique linéaire

Le premier axe concerne les techniques de vérification des CT. La logique linéaire nous semble un outil très prometteur. Une coopération est en cours (co-encadrement de DEA) avec l'équipe Temps Réel de l'IRCCyN de Nantes afin de formaliser l'introduction du temps dans la logique linéaire. Nous espérons ainsi pouvoir traiter le problème de l'accessibilité des RdP dans le cas d'une sémantique forte. Nous n'aurions alors plus qu'un seul formalisme à manipuler (le recours aux graphes de classe ne serait plus nécessaire). Cela permettrait de travailler directement sur des expressions symboliques. Nous n'aurions plus à recalculer le graphe de classe après chaque modification des CT pour nous assurer de l'accessibilité des scénarii évalués.

Naturellement, il nous faudra aussi définir une autre approche que le graphe de classe pour déterminer les scénarii à étudier. L'équipe de Robert Valette et Brigitte Pradin-Chézalviel travaille sur cet aspect ; des résultats récents, toujours basés sur la logique linéaire, offrent d'ores et déjà des réponses. Ces travaux reposant sur une approche similaire à la nôtre, par construction d'une sorte de FPN, nous pensons pouvoir les adapter.

## VI.2.2. Vers la transformation de modèles comportementaux

Le second axe d'amélioration a trait à la dérivation des diagrammes UML vers les RdP. Avec l'approche MDA (*Model Driven Approach*) de l'OMG et l'appel à la définition d'un langage de transformation de modèles [OMG-2002b], des moteurs de transformation vont être bientôt disponibles. Une coopération (initiée par un co-encadrement de DEA) sur le développement de l'un de ces moteurs (et de son langage ATL) est d'ailleurs en cours avec l'équipe ATLAS de l'université de Nantes.

Nous aimerions utiliser ce langage pour décrire nos transformations et de ce fait offrir une implémentation de nos processus de dérivation. Toutefois, un problème subsiste : les transformations ainsi décrites sont, a priori, automatiques. Afin de pouvoir faire intervenir l'analyste-concepteur dans un processus semi-automatique, nous réfléchissons à la définition d'un modèle de traçabilité des transformations. La dérivation se ferait en plusieurs itérations. Les traces obtenues à chaque étape seraient utilisées pour l'identification des questions à poser. Les réponses de l'analyste-concepteur permettraient alors de paramétrer les transformations suivantes. Un autre avantage de la conservation des traces de dérivation est de pouvoir identifier les parties concernées lors de la correction d'incohérences. Actuellement, ce problème n'ayant pas été abordé dans UML/PNO, la correction d'un diagramme relance un processus de dérivation complet portant sur tous les diagrammes.

Naturellement, ce travail nécessite une définition exhaustive des règles de dérivation ainsi que d'un méta-modèle de RdP compatible avec UML (ce qui n'est pas encore le cas). Nous pensons effectuer ce travail sur UML 2.0. En effet, le grand nombre d'ambiguïtés et de problèmes identifiés dans ce mémoire devrait être réduit. De plus, le langage d'action (*action semantics*) ainsi que les profils temps réel devraient être mieux intégrés.

## VI.2.3. Perspectives à long terme

Nous aimerions incorporer des travaux existants sur l'aspect statique : soit par l'utilisation de langages formels plus appropriés, soit par l'extension des RdP utilisés (RdP algébriques [Biberstein-2001]). Un certain nombre de restrictions a été défini pour UML/PNO, en particulier sur les communications entre composants ou l'allocation et l'instanciation nécessairement statique des objets. Bien qu'elles ne nous aient pas gênés lors de la modélisation de système, nous aimerions les lever dans un futur proche.

Par ailleurs, nous espérons offrir une plus large couverture du cycle de vie avec UML/PNO. Cela implique un interfaçage avec des travaux existants sur les langages de description d'architecture (comme par exemple ceux de [Faucou-2002]) afin d'aborder les phases de conception détaillée et d'implémentation.

# Publications associées à UML/PNO

## Revues


[1] M. Paludetto, J. Delatour, *UML et les réseaux de Petri: Vers une sémantique des modèles dynamiques et une méthodologie de développement des systèmes temps réels*, L'objet, numéro spécial "Objet et méthodes formelles", vol. 5, n° 3-4, pp. 443-467, 1999

[2] M. Paludetto, J. Delatour, A. Benzina, *UML et réseaux de Petri: Vers une formalisation des besoins préliminaires des systèmes embarqués*, TSI, Techniques et Science Informatiques (accepté, actuellement en seconde lecture)

[3] M. Paludetto, J. Delatour, A. Benzina, *Validation and verification of temporal constraints within a component-based development of embedded real-time systems*, SoSyM, Journal of Software and System Simulation (revision en seconde lecture)


## Conférences internationales avec comité de lecture et actes


[4] A. Benzina, M. Paludetto et J. Delatour, *About the suitability of Petri nets for describing, validating and evaluating SA-RT specifications*, Conf. Proc. APSEC/ICSC'97 Asia Pacific Software Eng. Conf. & Int. Computer Science, IEEE Hong Kong, 2-5 Décembre 1997

[5] A. Benzina, M. Paludetto et J. Delatour, *Translation of Structured Specifications into Object-Oriented Specifications by Means of Petri Nets*, 2nd IMACS/IEEE Int. Conf. CESA'98 Computational Engineering in Systems Applications, Tunisia, 1998

[6] J. Delatour, M. Paludetto et A. Benzina, *Modélisation par la méthode HOOD/PNO (Objets à réseaux de Petri),* RTS 98, Real-Time Systems, Paris, 14-16 janvier 1998

[7] A. Benzina, M. Paludetto et J. Delatour, *Spécification à objets à partir d'une analyse de type SA-RT*, RTS 98, Real-Time Systems, Paris, 14-16 janvier 1998

[8] J. Delatour et M. Paludetto, *UML/PNO, a way to merge UML and Petri Net Objects for the analysis of real-time systems*, Object-Oriented Technology and Real Time Systems Workshop in ECOOP'98, LNCS 1543, pp.511-514, juillet 98

[9] R. Van Paassen, J. Delatour et C. Pronk, *Middleware for real-time distributed simulation systems*, Joint Modular Languages Conference, Zurich, June 2000

[10] R. Van Paassen, C. Pronk et J. Delatour, *DUECA - Data-drive activation in distributed real-time computation*, AIAA Modeling and Simulation Technologies Conference (American Institute of Aeronautics and Astronautics), Denver, CO, 2000

[11] F. Taiani, M. Paludetto et J. Delatour, T. Cros, *Vérification d'objets temps réel à l'aide des réseaux de Petri et de la logique linéaire*, RTS'2001, Real-Time Systems - 9th Edition, Paris, 6-8 mars 2001

[12] F. Taiani, M. Paludetto et J. Delatour, *Composing real-time objects: a case for Petri nets and Girard's linear logic*, the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2001, Magdeburg, Germany, May, 2 - 4, 2001

[13] J. Delatour et F. De Lamotte, *ArgoPN : A CASE Tool Merging UML and Petri Nets*, 1st International Workshop on Validation and Verification of software for Enterprise Information Systems in ICEIS 2003, Angers, april 2003


## Conférences nationales avec comité de lecture et actes


[14] J. Delatour et M. Paludetto, *De HOOD/PNO à UML/PNO : une méthode pour les systèmes temps réels basée UML et objets à réseau de Petri*, MSR'99, Modélisation des Systèmes Réactifs, édition Hermes, ISBN 2-7462-0017-1, Paris, mars 1999

# Références bibliographique


[Abadi-1996] M. Abadi et L. Cardelli, "A Theory of Object", Springer-Verlag, 1996.

[Aksit-1994] M. Aksit, J. Bosch, W. van der Sterren et L. Bergmans, "Real-time Specification Inheritance Anomalies and Real-time Filters", ECOOP'94, 1994.

[Alhir-1999] S. S. Alhir, "Extending the Unified Modeling Language", http://home.earthlink.net/~salhir, 1999.

[Alla-2000] Alla, "Dia manual", 0.92, http://www.lysator.liu.se/~alla/dia/, 2000.

[Allen-1983] J. F. Allen, "Maintaining Knowledge about Temporal Intervals", *Communications of the ACM*, vol. 26, n° 11, pp. 832-843, 1983.

[Alvarez-2001] J. Alvarez, A. Evans et P. Sammut, "MML and the Metamodel Architecture.", WTUML: Workshop on Transformations in UML in ETAPS 2001, Genova, Italy, 2001.

[America-1987] P. America, "Inheritance and Subtyping in a Parallel Object-Oriented Language", ECOOP'87, Paris, France, 1987.

[Andre-1995a] P. Andre, "Méthodes formelles et à objets pour le développement du logiciel : Etudes et propositions", PhD., Université de Rennes1, Rennes, 1995a.

[Andre-1995b] P. Andre, F. Barbier et J. C. Royer, "Une expérimentation de développement formel à objet", *Techniques et Sciences Informatiques*, vol. 24, n° 8, pp. 973-1005, 1995b.

[André-2000] P. André, A. Romanczuk, J.-C. Royer et A. Vascon-celos, "Checking the Consistency of UML Class Diagrams Using Larch Prover", 3 Rigorous Object-Oriented Methods Workshop in BCS eWics, 2000.

[Atamna-1994] Y. Atamna, "Réseaux de Petri Temporisés Stochastiques Classique et Bien Formés : Définition, Analyse et Application aux Systémes Distribués Temps Réels", PhD., Université Paul Sabatier, Toulouse, 1994.

[Babau-1998] J. P. Babau, "Modélisation par la méthode OMT et SDL", Real Time System'98, Paris, 1998.

[Bachatène-1995] H. Bachatène et H. Coriat, "PAM : a Petri net-based Abstract Machine for the Specification of Concurrent Systems", 1st W. on OO Programming and Models of Concurrency in the 16th Int. Conf. on Application and Theory of Petri Nets, Turin, 1995.

[Bako-1990] B. Bako, "Mise en oeuvre et simulation du niveau coordination de la commande des ateliers flexibles: une approche mixte réseaux de Petri et système de règles", PhD., Université Paul Sabatier, Toulouse, 1990.

[Baresi-2001a] L. Baresi et M. Pezzé, "Improving UML with Petri Net", *Electronic Notes in Theorical Computer Science*, vol. 44, n° , pp. 1-13, 2001a.

[Baresi-2001b] L. Baresi et M. Pezzè, "On Formalizing UML with High-Level Petri Nets", *Concurrent Object-Oriented Programming and Petri Nets*, vol. A special volume in the Advances in Petri Nets series, n° 2001 of Lecture Notes in Computer Science - Springer Verlag., pp. 271-300, 2001b.

[Baresi-2002] L. Baresi et M. Pezzè, "A Toolbox for Automating Visual Software Engineering", In Proceedings of 5th International Conference FASE 2002 (part of ETAPS 2002), Grenoble (Francia), 2002.

[Basten-1996] T. Basten et W. M. P. Van der aalst, "A process-Algebraic Approach to Life-Cycle Inheritance : Inheritance = Encapsulation + Abstraction", Eindhoven University of Technology, Dep. of Mathematics and Computing Science, Eindhoven, the Netherlands CSR 96/05, march 1996.

[Bastide-1995] R. Bastide, "Approaches in Unifying Petri Nets and the Object Oriented Approaches", 1st Workshop on Object Oriented Programming and Models of Concurrency, in the 16th International Conference on Application and Theory of Petri Nets, Turin, 1995.

[Battiston-1995] E. Battiston, A. Chizzoni et F. De Cindio, "Inheritance and Concurrency in CLOWN", Workshop on Object-Oriented Programming and Models of Concurrency, Torino, Italy, 1995.

[Battiston-1988] E. Battiston, F. De Cindio et G. Mauri, "OBJSA Nets", Springer Verlag, 1988.

[Beauvais-1999] J. R. Beauvais et Et_al, "Une modélisations de Statecharts et Activitycharts en Signal", Modélisation des Systémes Réactifs, Paris, 1999.

[Benveniste-1990] A. Benveniste et P. LeGuernic, "Hybrid dynamical system theory and the SIGNAL Language", *IEEE Transaction of Automatic Control*, vol. 35, n° 5, pp. 534-546, 1990.

[Benzina-1997] A. Benzina, M. Paludetto et J. Delatour, "About the suitability of Petri nets for describing, validating and evaluating SA-RT specifications.", Asia-Pacific Software Engineering Conf. and Int. Computer Science Conf., Hong Kong, 1997.

[Bernardi-2002] S. Bernardi, S. Donatelli et J. Merseguer, "From UML Sequence Diagrams and Statecharts to analysable Petri Net models", Third International Workshop on Software and Performance (WOSP 2002) in conjonction with ISSTA 2002 (International Symposium on Software Testing and Analysis), Rome, Italy, 2002.

[Berry-1992] G. Berry et G. Gonthier, "The ESTEREL Synchronous Programming Language : Design, Semantics, Implementation", *Science of Computer Programming*, vol. 19, pp. 87-152, 1992.

[Berthelot-1991] G. Berthelot, C. Johnen et L. Petrucci, "PAPETRI: Environment for the Analysis of Petri Nets", *LNCS, Springer Verlag*, vol. 531, 1991.

[Berthomieu-2001] B. Berthomieu, "La méthode des Classes d'états pour l'Analyse des Réseaux Temporels - Mise en Oeuvre et Extension à la multi-sensibilisation", Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR 2001), Toulouse, France, 2001.

[Biberstein-1997] O. Biberstein, "CO-OPN/2: An Object-Oriented Formalism for the Specification of Concurrent Systems", Thése de doctorat, University of Geneva, 1997.

[Biberstein-2001] O. Biberstein, D. Buchs et N. Guelfi, "Object-Oriented Nets with Algebraic Specifications: The CO-OPN/2 Formalism", Advances in Petri Nets on Object-Orientation, 2001.

[Blair-1998] L. Blair et G. Blair, "The Impact of Aspect-Oriented Programming on Formal Method", Computing Departement, Lancaster University, Lancaster MPG-98-08, May 1998.

[Boas-1998] P. v. E. Boas, "Formalizing UML: Misson Impossible?", OOPSLA'98 Workshop on Formalizing UML. Why? How?, 1998.

[Bock-1999] C. Bock, "Unified behavior models", *Jounal of Object Oriented Programming*, vol. 12, n° 5, 1999.

[Bodeveix-2002] J.-P. Bodeveix, T. Millan, C. Percebois, Christophe Le Camus, P. Bazex, L. Feraud et R. Sobek, "Extending OCL for verifying UML models consistency", Workshop on Consistency Problems in UML-based Software Development in UML 2002, 2002.

[Bolognesi-1987] T. Bolognesi et E. Brinksma, "Introduction to the ISO specification Language LOTOS", *Computer Networks and ISDN Systems*, vol. 14, n° 1, 1987.

[Bondavalli-1999] A. Bondavalli, I. Majzik et I. Mura, "Automated Dependability Analysis of UML Designs", Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC99), Saint-Malo, France, 1999.

[Booch-1994] G. Booch, " Analyse et Conception Orientées objets", 2nd ed. , Addison-Wesley, ISBN 2-87908-069-x, 1994.

[Booch-1998] G. Booch, J. Rumbaugh et I. Jacobson, "The Unified Modeling Language User Guide", Addison-Wesley, 1998.

[Börger-2000] E. Börger, A. Cavarra et E. Riccobene, "Modeling the Dynamics of UML State Machines", *Abstract State Machine. Theory and Application*, vol. 1912, 2000.

[Börger-2001] E. Börger, A. Cavarra et E. Riccobene, "Solving Conflicts in UML State Machines Concurrent States", Workshop on Concurrency Issues in UML (CIUML) at UML'2001, Toronto, 2001.

[Boucheneb-1999] H. Boucheneb, "Conception et analyse d'un modèle temporisé unificateur à base de réseaux de Petri de Haut Niveau", Thése d'état, Université des Sciences et de la technologie Houari Boumedienne, Alger, 1999.

[Boyer-2001] M. Boyer, "Contribution à la modélisation des systèmes à temps contraint et application au multimédia", PhD, Université Paul Sabatier, Toulouse, 2001.

[Breu-1997] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe et V. Thurner, "Towards a Formalization of the Unified Modeling Language", in *ECOOP'97 -- Object-Oriented Programming, 11th European Conference*, vol. 1241, *LNCS*, M. Aksit and S. Matsuoka, Springer, 1997.

[Briot-1996] J.-P. Briot, "Objets, parallélisme et répartition", *Techniques et Sciences Informatiques*, vol. 15, n° 6, pp. 765-799, 1996.

[Bruel-1998] J.-M. Bruel, "Transforming UML Models to Formal Specifications", OOPSLA'98, Workshop on Formalizing UML. Why? How?, 1998.

[Bruyn-1988] W. Bruyn, R. Jensen, D. Keskar et P. Ward, "ESML : An Extended Systems Modelling Languages", *ACM Software Engineering Notes*, vol. 13, n° 1, pp. 58-67, 1988.

[Bucci-1995] G. Bucci, M. Campanai et P. Nesi, "Tools for Specifying Real-Time Systems", *Real-Time Systems*, vol. 8, pp. 117-172, 1995.

[Burns-1994] A. Burns et A. Wellings, "HRT-HOOD : a Structured Design Method for Hard Real-Time Systems", *Real-Time Systems*, vol. 6, pp. 73-114, 1994.

[Calvez-1990] J. P. Calvez, "Spécification et conception des systèmes, une méthodologie", Masson, 1990.

[Carcagno-1995] L. Carcagno, " Spécification et conception de systèmes complexes temps réel strict distribués", PhD., Université Paul Sabatier, Toulouse, France, 1995.

[Cardoso-2001] J. Cardoso, C. Sibertin-Blanc et C. Soule-Dupuy, "Une sémantique formelle des diagrammes d'interaction d'UML via les réseaux de Petri", Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR'2001), Toulouse (France), 2001.

[Carreira-2000] P. Carreira et M. E. F. Costa, "Automatically verifying an object-oriented specification of the steam-boiler system", Proceedings of the 5th International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS'2000), 2000.

[Cassez-1995] F. Cassez et O. Roux, "Compilation of the ELECTRE reactive language into finite transition systems", *Theorical Computer Science*, vol. 146, n° 1-2, pp. 109-143, 1995.

[Chang-1993] S. K. Chang, A. Perkusich, J. De Figueiredo, B. Ehrenberger et M. J. Yu, "The Design of Real-Time Distributed Information System with Object-Oriented and Fault-Tolerant Characteristics", SEKE'93, The 5th International Conference on Software Engineering and Knowledge Engineering, 1993.

[Clark-2000] R. G. Clark et A. M. D. Moreira, "Use of E-LOTOS in Adding Formality to UML", *Journal of Universal Computer Science*, vol. 6, n° 11, pp. 1071-1087, 2000.

[Clarke-1996a] E. Clarke et M. Wing, "Formal methods: State of the Art and Future", *ACM Computing Surveys*, vol. 28, n° 4, pp. 626-643, 1996a.

[Clarke-1996b] E. M. Clarke et R. Kurshan, "Computer-Aided Verification", *IEEE Spectrum*, vol. 33, n° 6, pp. 61-67, 1996b.

[CNRS-1988] X. CNRS, "Le temps réel - Groupe de réflexion sur le temps réel", *Techniques et Sciences Informatiques*, vol. 8, n° 5, pp. 493-500, 1988.

[Cook-1990] W. Cook, W. Hill et P. Canning, "Inheritance is not subtyping", POP'90, San-Francisco, USA, 1990.

[Corsetti-1991] E. Corsetti et and al, " Dealing with Different Time Scales in Formal Specification", 6th IEEE Int. Workshop on Software Specification and Design, Como, Italy, 1991.

[Courtiat-2000] J.-P. Courtiat, C. A. S. Santos, C. Lohr et B. Outtaj, "Experience with RTLOTOS, a Temporal Extension of the LOTOS Formal Description Technique", *Computer Communications*, vol. 23, n° 12, pp. 1104-1123, 2000.

[Csertán-2002] G. Csertán, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza et D. Varró, "VIATRA - Visual Automated Transformations for Formal Verification of UML Models", ASE 2002, International Conference on Automated Software Engineering, Edinburgh, Scotland, 2002.

[Darvas-2002] A. Darvas, I. Majzik et B. Benyó, "Verification of UML Statechart Models of Embedded Systems", 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS 2002), Brno, Czech Republic, 2002.

[David-1992] R. David et H. Alla, "Du grafcet au reseaux de Petri", Hermes, 1992.

[De Hondt-1997] K. De Hondt, C. Lucas et P. Steyaert, "Reuse Contracts as Component Interface", Second International Workshop on Component-Oriented Programming, Jyvskyl, Finland, 1997.

[Delatour-2000] J. Delatour, "Introduction à la notation UML 1.3", LAAS-CNRS, Toulouse 2000.

[Delatour-2003] J. Delatour et F. De Lamotte, "ArgoPN : A CASE Tool Merging UML and Petri Nets", 1st International Workshop on Validation and Verification of software for Enterprise Information Systems in ICEIS 2003, Angers, 2003.

[Delatour-1998] J. Delatour et M. Paludetto, "UML/PNO: a Way to Merge UML and Petri Net Objects for the Analysis of Real-Time Systems", Object-Oriented Technology and Real Time Systems Workshop, ECOOP 98, Bruxelles, 1998.

[Desnoix-2001] G. Desnoix, "Manuel utilisateur d'Alma", http://www.memoire.com/guillaume-desnoix/, 2001.

[Di Stefano-1991] A. Di Stefano et O. Mirabella, "A Fast Sequence Control Device Based on Enhanced Petri Nets.", *Microprocessors and Microsystems*, vol. 15, n° 4, pp. 179-186, 1991.

[Dolev-1992] D. Dolev, S. Kramer et D. Malki, "Total ordering of messages in broadcast domains", The Hebrew University of Jerusalem CS-92-9, November 19921992.

[Dong-2001] Z. Dong et X. He, "Integrating UML Statechart and Collaboration Diagrams Using Hierarchical Predicate Transition Nets", *Lecture Notes in Informatics*, vol. P-7, pp. 99-112, 2001.

[Dorseuil-1991] A. Dorseuil et P. Pillot, "Le temps réel en milieu industriel", Dunod, 1991.

[Douglass-1998] B. P. Douglass, "Real-time UML : Developing Efficient Objects for Embedded Systems", Addison-Wesley, 0-201-32579-9, 1998.

[D'Souza-1998] D. D'Souza et A. Wills, "Objects, Components and Frameworks With UML: The Catalysis Approach", Addison-Wesley, 1998.

[Duke-1994] R. Duke, G. Rose et G. Smith, "ObjectZ: a Specification Language Advocated for the Description of Standards", Software Verification Research Center, University of Queensland, Australia, Queensland 94-95, December 941994.

[Dumas-2001] M. Dumas et A. t. Hofstede, "UML Activity Diagrams as a Workflow Specification Language", International Conference on the Unified Modeling Language (UML), Toronto, Canada, 2001.

[Dürr-1995] E. Dürr et S. Goldsack, "The Development of Concurrent and Real-Time Systems using VDM ++", Cap Volmac AFRO/CG:ED/COURSE/V1,1995.

[Ebert-1994] J. Ebert et G. Engels, "Observable or Invocable Behaviour - You Have to Choose", Department of Computer Science, Leiden University Technical Report 94-38, ftp://ftp.wi.LeidenUniv. nl/pub/CS/TechnicalReports/1994/tr94-38.ps.gz, 1994.

[Eijk-1989] P. V. Eijk, "Tools for LOTOS Specification Style Transformation", FORTE'89, 1989.

[Eirich-1991] T. Eirich et F. Hauck, "Inheritance by Aggregation", Computer Science Department, Erlangen-Nümberg, Germany TR-14-4-91, november 1991.

[Ek-1995] A. Ek, "The SOMT Method", Telelogic, Balbo, sweden, 19 september, 1995.

[Elkoutbi-2000] M. Elkoutbi et R. K. Keller, "User Interface Prototyping based on UML Scenarios and High-level Petri Nets", *Application and Theory of Petri Nets 2000 (Proc. of 21st Intl. Conf. on ATPN)*, vol. Springer-Verlag LNCS 1825, Aarhus, Denmark, June, pp. 166-186, 2000.

[Elloy-1989] J. P. Elloy, "Une approche intégrée des outils de conception des applications temps réels répartis", Habilitation à diriger des recherches, Université de Nantes, Nantes, 1989.

[Eshuis-2002] H. Eshuis, "Semantics and Verification of UML Activity Diagrams for Workflow Modelling", PhD, University of Twente, Twente, 2002.

[Eshuis-2001] R. Eshuis et R. Wieringa, "A Formal Semantics for UML Activity Diagrams", University of Twente, CTIT, Twente CTIT01-04, 2001.

[Esprit-1998] Esprit, "Esprit Project 27439 - HIDE High-level Integrated Design Environment for Dependability", 1998.

[Esprit-1999] Esprit, "Esprit Project 20897 - SACRES Safety Critical Embedded Systems: From Requirements to System Architecture", Munich,http://www.cordis.lu/esprit/src/20897.htm, 1999.

[Evans-1998] A. Evans, "Making UML Precise", OOPSLA'98, Workshop on Formalizing UML. Why? How?, 1998.

[Evans-1999] A. S. Evans, R. B. France, K. C. Lano et B. Rumpe, "Meta-modelling semantics of UML", in *Behavioural Specifications for Businesses and Systems*, H. Kilov, Ed.: Kluwer, 1999, chap 4.

[Faucou-2002] S. Faucou, "Description et construction d'architectures opérationnelles validées temporellement", PhD., Ecole Centrale de Nantes, Nantes, 2002.

[Feldbruge-1991] F. Feldbruge et K. Jensen, "Computer Tools for High-level Petri Net,", Springer Verlag, ISBN 0-387-54125-X., 1991.

[Fernandez-1996] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier et M. Sighireanu, "Cadp (cæsar/aldebaran development package): A protocol validation and verification toolbox", The 8th Conference on Computer-Aided Verification, New Brunswick, New Jersey, USA, 1996.

[Fernández-2001] J. L. Fernández et J. AmbrosioToval, "Seamless formalizing the UML semantics through metamodels", in *Unified Modeling Language: System Analysis, Desing and Development Issues*, U. o. N.-L. Dr. Keng Siau, Dr Terry Halpin,, Ed.: Microsoft Corporation, Idea Group, 2001.

[Fernández-2000] J. L. Fernández et A. Toval, "Can Intuition Become Rigorous? Foundations for UML Model Verification Tools", International Symposium on Software Reliability Engineering, San Jose, California, USA, 2000.

[Flake-2002a] S. Flake et W. Mueller, "An OCL Extension for Real-Time Constraints", in *Object Modeling with the OCL*, vol. LNCS 2263,, T. C. a. J. Warmer, Ed. Heidelberg, Germany: Springer-Verlag, 2002a.

[Flake-2002b] S. Flake et W. Mueller, "Specification of Real-Time Properties for UML Models", Hawai'i International Conference on System Sciences (HICSS-35), Big Island, Hawaï, USA, 2002b.

[Fleischhack-1993] H. Fleischhack et U. Lichtblau, "MOBY, a Tool for High Level Petri Nets with Objects", IEEE Int. Conf. on Systems, Man and Cybernetics, Le touquet, France, 1993.

[France-1997] R. France, J.-M. Bruel, M. Larrondo-Petrie et M. Shroff, "Exploring the Semantics of UML Type Structures with Z", Proc. 2nd IFIP Conf. Formal Methods for Open Object-Based Distributed Systems (FMOODS'97), 1997.

[France-1994] R. France et M. Larrondo-Petrie, "From Structured Analysis to Formal Specification : State of the Theory", Computer Science Conference, 1994.

[Fraser-1991] M. D. Fraser, K. Kumar et V. K. Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap", *IEEE Transaction on Software Engineering*, vol. 15, n° 5, pp. 454-466, 1991.

[Fraser-1994] M. D. Fraser, K. Kumar et V. K. Vaishnavi, " Strategies for Incorporating Formal Specification in Software Developmment", *Communication of the ACM*, vol. 37, n° 10, pp. 74-86, 1994.

[Gallon-1997] L. Gallon, "Le modèle réseaux de Petri temporisés stochastiques : extension et application", PhD., Université Paul Sabatier, Toulouse, 1997.

[Gamma-1995] E. Gamma, R. Helm, R. Johnson et J. Vlissides, "Design Patterns : Elements of Reusable Object-Oriented Software", , 1995.

[Gaubert-1999] S. Gaubert et J. Mairesse, "Asymptotic analysis of heaps of pieces and application to timed Petri Nets", 8th International Workshop on Petri Nets and Performance Models (PNPM'99), Zaragoza, Spain, 1999.

[Gehrke-1998] T. Gehrke, U. Goltz et H. Wehrheim, "The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process", Institut für Informatik, Universität Hildesheim, Hildesheim ISSN 0941-3014, september 1998.

[Gehrke-1999] T. Gehrke, U. Goltz et H. Wehrheim, "Zur semantischen Analyse der dynamischen Modelle von UML mit Petri-Netzen", Proceedings of The 6th Symposium on Development and Operation of Complex Automation Systems, 1999.

[Geisler-1998] R. Geisler, "Precise UML Semantics Through Formal Metamodeling", in *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.

[Genrich-1986] H. J. Genrich, "Predicate/Transition Nets", *Lecture Notes in Computer Science*, vol. 254, pp. 207-247, 1986.

[Ghezzi-1991] C. Ghezzi, M. Jazayeri, D. Mandrioli, "Fundamentals of Software Engineering", Prentice Hall, 1991.

[Giese-1999] H. Giese, J. Graf et G. Wirtz, "Closing the Gap between Object Oriented Modeling of Structure and Behavior", Institut für Informatic, Münster University, Münster, Germany 16/99, March 1999.

[Girard-1987] J. Y. Girard, "Linear Logic", *Theoritical Computer Science*, vol. 50, 1987.

[Girard-1990] J. Y. Girard, "La logique linéaire", *Pour la science*, vol. 150, avril, 1990.

[Girard-1991] J. Y. Girard, "Introduction à la logique linéaire", in *Logique et informatique : une introduction*, vol. 8, C. didactique, INRIA, 1991.

[Girault-1997] F. Girault, "Formalisation en logique linéaire du fonctionnement des réseaux de Petri", PhD., Université Paul Sabatier, Toulouse, France, 1997.

[Gogolla-1998] M. Gogolla et F. P. Presicce, "State Diagrams in UML: A Formal Semantics using Graph Transformations", in *Proceedings PSMT'98 Workshop on Precise Semantics for Modeling Techniques*, M. Broy, D. Coleman, T. S. E. Maibaum, and B. Rumpe, Eds.: Technische Universität München, TUM-I9803, 1998.

[Gogolla-2000] M. Gogolla, O. Radfelder, R. Kollmann et M. Richters, "Analysing Atomic Dynamic UML Notions by Surfing through the UML Metamodel", WORKSHOP Dynamic Behaviour in UML Models: Semantic Questions in <<UML>> 2000, 2000.

[Groupe-2001a] Groupe, "KUML", Hambourg, http://www.informatik.fh-hamburg.de/~kuml/, 2001a.

[Groupe-2001b] Groupe, "Umbrello", http://uml.sourceforge.net, 2001b.

[Guerraoui-1995] R. Guerraoui, "Modular Atomic Objects", *Theory and Practice of Object Systems (TAPOS)*, vol. 1, n° 2, November 1995, pp. 89-99, 1995.

[Guerraoui-1996] R. Guerraoui et A. Schiper, "Fault-Tolerance by Replication in Distributed Systems", Reliable Software Technologies - Ada-Europe'96, 1996.

[Halbwachs-1991] N. Halbwachs, P. Caspi, P. Ratmond et D. Pilaud, "The Synchronous Data Flow Programming Language LUSTRE", *Proceedings of the IEEE*, vol. 79, n° 9, pp. 1305-1321, 1991.

[Hamie-1998] A. Hamie, "A Formal Semantics for Checking and Analysing UML Models", in *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.

[Hanisch-1997] A. A. Hanisch et T. S. Dillon, "Object Oriented Behaviour Modelling for Real-Time Design", 3rd int. W. on OO Real Time Dependable Systems, Newport Beach, 1997.

[Harel-1987] D. Harel, "Statecharts : a Visual Formalism for Complex Systems", *Science of Computer Programming*, vol. 8, pp. 231-274, 1987.

[Harel-1985] D. Harel et A. Pnuelli, "On the development of Reactive Systems, in Logic and Models of Concurrent Systems", *NATO ASI in Computer Science*, pp. 447-498, 1985.

[Hay-1998] D. Hay, "UML Misses the Boat", in *Workshop on Formalizing UML. Why? How? in OOPSLA'98*, L. Andrade, A. Moreira, A. Deshpande, and S. Kent, Eds., 1998.

[He-2000] X. He, "Formalizing Class Diagrams Using Hierarchical Predicate Transition Nets", the 24th International Computer Software and Application Conference (COMPSAC'2000), Taiwan, 2000.

[Hernalsteen-1998] C. Hernalsteen, "Specification, Validation and Verification of Real-Time Systems in ET-LOTOS", PhD, Université libre de Bruxelle, Bruxelle, 1998.

[Heymer-2000] S. Heymer, "A Semantics for MSC based on Petri Nets Components", Lübek university, Lübek SIIM-TR-A-00-12, June 2000.

[Ho-1999] W. Ho, J. M. Jézéquel, A. Le Guenec et F. Pennaneach, "UMLAUT : an Extensible UML Transformation Framework", Automated Software Engineering, ASE'99, Florida, 1999.

[Hoare-1978] C. A. R. Hoare, "Communicating Sequential Processes", *Communication of ACM*, vol. 21, n° 8, pp. 666-677, 1978.

[Hoare-1987] C. A. R. Hoare, "An Overview of Some Formal Methods for Programm Design", *IEEE Computer*, vol. 20, n° 9, pp. 85-91, 1987.

[Holvoet-1995] T. Holvoet et P. Verbaeten, "PN-TOX : a Paradigm and Develpment Environment for Object Concurrency Specifications", 1st W. on OO Programming and Models of Concurrency in the 16th Int. Conf. on Application and Theory of Petri Nets, Turin, 1995.

[Holzmann-1997] G. J. Holzmann, "The Model Checker SPIN", *IEEE Transactions on Software Engineering*, vol. may, n° Special issues on Formal Methods in Software Practice, 1997.

[Hoogeboom-1991] B. Hoogeboom et W. A. Halang, " The Concept of Time in Software Engineering for Real-Time Systems", Proc. 3rd Int. Conf. On Software engineering for Real-Time systems, 1991.

[HRM-1989] HRM, "HOOD Reference Manual, version 3.0", European Space Agency WME/89-173JB, 1989.

[Huszerl-2001] G. Huszerl et I. Majzik, "Modelling and Analysis of Redundancy Management in Distributed Object-Oriented Systems by Using UML Statecharts", 27th Euromicro Conference, Warsaw, Poland, 2001.

[Huszerl-2002] G. Huszerl, I. Majzik, A. Pataricza, K. Kosmidis et M. D. Cin, "Quantitative Analysis of UML Statechart Models of Dependable Systems", *The Computer Journal*, vol. 45, n° 3, pp. 260-277, 2002.

[Information Society Technologies-1999] Information Society Technologies, "AIT-WOODDES", http://wooddes.intranet.gr/, 1999.

[Information Society Technologies-2001] Information Society Technologies, "IST-2000-25434 - DepAuDE Dependability for embedded automation systems in dynamic environment with intra-site and inter-site distribution aspects", Milan,http://www.depaude.org/, 2001.

[Irit-2001] IRIT, "NEPTUNE (Nice Environment with a Process and Tools Using Norms (UML, XML and XMI) and Examples)", IRIT, Toulouse,http://neptune.irit.fr, 2001.

[ITEA-1999] ITEA, "ITEA 99012- DESS (Software Development Process for Real-Time Embedded Software Systems)", Eindhoven1999.

[ITU-TS-1996] ITU-TS, "Z120 - Message Sequence Chart", Recommendation, ITU, 1996.

[Jahanian-1986] F. Jahanian et A. K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems", *IEEE Transaction on Software Engineering*, vol. 12, n° 9, pp. 890-904, 1986.

[Jahanian-1988] F. Jahanian et D. A. Stuart, "A Method for Verifying Properties of Modechart Specifications", Proceedings of 9th IEEE Real-Time Systems Symposium, Huntsville, Alabama, USA, 1988.

[Jensen-1981] K. Jensen, "Coloured Petri Nets and the Invariant Method", *Theoretical Computer Science*, vol. 14, pp. 317-336, 1981.

[Jones-1993] C. B. Jones, "VDM, une méthode rigoureuse pour le développement du logiciel", Prentice-Hall, 1993.

[Jørgensen-2002] J. B. Jørgensen, "Coloured Petri Nets in UML-Based Software Development - Designing Middleware for Pervasive Healthcare", Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, 2002.

[Kappel-1991] G. Kappel et M. Schrefl, "Using an OO Diagram Technique for the Design of Information Systems", in *Dynamic Modelling of Information Systems*, V. Hee, Ed. North Holland: Elsevier Science, pp. 121-164, 1991.

[Kappel-1994] G. Kappel et M. Schrefl, "Inheritance of Object Behavior - Consistent Extension of Object Life Cycles", East/West Database Workshop, Bonn, Germany, 1994.

[Keller-1994] R. K. Keller, X. Shen et G. Bochman, "Macronet-a Simple, Yet, Expressive and Flexible Formalism for Business Modeling", WCSCW in 15th Int. Conf. on Application and theory of Petri Nets, Zaragoza, Spain, 1994.

[Kemmerer-1990] R. A. Kemmerer, "Integrating Formal Methods into the Development Process", *IEEE Transaction on Software Engineering*, vol. 15, n° 10, pp. 37-50, 1990.

[Khalfaoui-2002] S. Khalfaoui, E. Guilhem, H. Demmou et R. Valette, "Une méthode pour obtenir des scénarios critiques dans les systèmes mécatroniques", ESREL 2002, 13e colloque Européen de sûreté de fonctionnement, Lyon, 2002.

[Khansa-1997] W. Khansa, "Réseaux de Petri p-temporels : contribution à l'étude des systèmes à événements discrets", Ph.D, Université de Savoie, Annecy, France, 1997.

[King-1999] P. King et R. Pooley, "Using UML to Derive Stochastic Petri Nets Models", Fifteenth UK Performance Engineering Workshop (UKPEW '99), University of Bristol, 1999.

[Kopetz-1983] H. Kopetz, " Real-time in Distributed Real-Time Systems", Real-time in distributed real-time systems.Proc. 5th IFAC workshop on Distributed Computer Control Systems, Oxford, UK, 1983.

[Kuzniarz-2002] L. Kuzniarz, G. Reggio, J.-L. Sourrouille et Z. Huzar, "Workshop on Consistency Problems in UML-based Software Development", Research Report 2002:06: Blekinge Institute Of Technology, pp. 168, 2002.

[Lakos-1995a] C. Lakos, "Pragmatic Inheritance Issues for Object Petri Nets", TOOLS'95 Pacific conference,

Melbourne, Australia, 1995a.

[Lakos-1995b] C. A. Lakos, "From Coloured Petri nets to Object Petri nets", in *Application and Theory of Petri Nets*, vol. 935, L. N. i. C. Science, Ed.: Springer, 1995b, pp. 278--297.

[Lakos-1997] C. A. Lakos, "Object Oriented Modelling with Object Petri nets", Advances in Petri nets, Berlin, 1997.

[Lakos-1990 ] C. A. Lakos et C. D. Keen, "LOOPN --- Language for Object-Oriented Petri Nets. ", *Simulation Series; Proceedings of the SCS Multiconference on Object-Oriented Simulation, 1991, Anaheim, USA* , vol. 23, n° 3, pp. 22-30 , 1990 .

[Lamport-1978] L. Lamport, " Time, cloks, and the ordering of events in a distributed system", *Communication of the ACM*, vol. 21, n° 7, 1978.

[Lamport-1994] L. Lamport, "The Temporal Logic of Action", *ACM Transaction on Programming Languages and Systems*, vol. 16, pp. 768-775, 1994.

[Lano-1991] K. Lano, "an Object-Oriented Extensions to Z", Z user Workshop, Workshop in Computing, Oxford, 1991.

[Lano-1998] K. Lano et J. Bicarregui, "Formalising the UML in Structured Temporal Theories", in *Proceedings Second ECOOP Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications)*, H. Kilov and B. Rumpe, Eds.: Technische Universität München, TUM-I9813, 1998, pp. 105--121.

[Lano-1997] K. Lano et S. Goldsack, "Formalising Real-time System Design", ECOOP'97, Workshop on Real-Time System, 1997.

[Laplante-1991] P. A. Laplante, "Real-Time Systems Design and Analysis, an Engineer's handbook", IEEE Computer Society Press, 0-8186-3107-4, 1991.

[Laprie-1995] J. C. Laprie, "Guide de la sureté de fonctionnement", Cépaduès edition, 2-85428-382-1, 1995.

[Lavazza-2001] L. Lavazza, G. Quaroni et M. Venturelli, "Combining UML and Formal Notations for Modeling Real-Time Systems", Joint 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering, Wien, 2001.

[Le Guennec-2001] A. Le Guennec, "Génie Logiciel et Méthodes Formelles avec UML : Spécification, Validation et Génération de tests", PhD., Université de Rennes 1, Rennes, 2001.

[Le Guernic-1997] P. Le Guernic, "Projet EP-ATR : Environnement de programmation d'applications temps réels", IRISA, Rennes Rapport d'activités scientifiques,1997.

[Le Lann-1994] G. Le Lann, P. Minet et D. Powell, "Systémes répartis", *Informatique tolérante aux fautes, ARAGO*, vol. 15, pp. 55-71, 1994.

[Leblanc-1996] P. Leblanc et V. Encontre, "Object-Oriented Real-Time Techniques : Method Guidelines", Verilog version 1.0, 1996.

[Ledang-2002] H. Ledang, "Traduction Systématique de Spécifications UML en B", PhD., Université de Nancy 2, Nancy, 2002.

[L'Her-1997] D. L'Her, "Modélisation du Grafcet temporisé et vérification de propriétés temporelles", PhD., Université de Rennes I, Rennes, 1997.

[Lilius-1998] J. Lilius et I. P. Paltor, "Digital Sound Recorder: A Case Study On Designing Embedded Systems Using the UML Notation", TUCS - Turku Centre for Computer Science TUCS-TR-234,,http://www.tucs.fi/publications/techreports/TR234.html, 1998.

[Lilius-1999a] J. Lilius et I. Porres Paltor, "The Semantics of UML State Machines", TUCS, Turku Centre for Computer Science Tecnical report 273, ISBN 952-12-0446-1, June, www.tucs.abo.fi\publications\techreports\TR273.html, 1999a.

[Lilius-1999b] J. Lilius et I. Porres Paltor, "vUML, a Tools for Verifying UML Models", TUCS, Turku Centre for Computer Science Tecnical report 272, ISBN 952-12-0445-1, May, www.tucs.abo.fi\publications\techreports\TR272.html, 1999b.

[Lime-2003] D. Lime et O. H. Roux, "State class Timed Automaton of a Time Petri Net", The 10th International Workshop on Petri Nets and Performance Models, (PNPM'03:), Urbana, USA, 2003.

[Liskov-1994] B. H. Liskov et J. M. Wing, "A Behavioral Notion of Subtyping", *ACM Transaction on Programming Langages and Systems*, vol. 16, n° 6, pp. 1811-1841, 1994.

[Liskov-1999] B. H. Liskov et J. M. Wing, "Behavioural Subtyping Using Invariants and Constraints", MIT Lab, Cambridge CMU-CS-99-156, july 1999.

[Löwe-1995] M. Löwe, D. Wikarski et Y. Han, "Higher-Order Object Nets and their Application to Workflow Modelling", FB Informatiks, Technical University Berlin 95-34, 1995.

[Machado-2001] R. Machado, J. Fernandes et H. Santos, "A Methodology for Complex Embedded Systems Design: Petri Nets within an UML Approach", in *Architecture and Design of Distributed Embedded Systems*, ISBN 0-7923-7345-6, B. Kleinjohann, Ed. Boston, U.S.A: Kluwer Academic, 2001, pp. 1-10.

[Maier-1997] C. Maier et D. Moldt, "Object Coloured Petri Nets : a Formal Techniques for OO Modelling", in *Petri Nets in System Engineering, Modelling, Verification and Validation*, M. O. Steher, Ed. University of Hamburg, 1997, pp. 11-19.

[Majzik-1998] I. Majzik et A. Bondavalli, "Automatic Dependability Modelling of Systems Described in UML", 9th International Symposium on Software Reliability Engineering (ISSRE-98), Paderborn, Germany, 1998.

[Mandrioli-1992] D. Mandrioli, "The Specification of Real-Time Systems: a Logical Object Oriented Approach", TOOLS'92, 1992.

[Marti-Oliet-1991] M. Marti-Oliet et J. Meseguer, "From Petri Nets to Linear Logic Through Catagories : a Survey", *International Journal of Foundation of Computer Science*, vol. 2, n° 2, pp. 297-399, 1991.

[Matsuoka-1990] S. Matsuoka, K. Wakita et A. Yonezawa, "Synchronization Constraints with Inheritance: What is not Possible - so What is ?", Dept. of Information Science, the University of Tokyo,, Tokyo Technical Report 10, http://citeseer.nj.nec.com/matsuoka90synchronization.html, 1990.

[Matsuoka-1993] S. Matsuoka et A. Yonezawa, "Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages", in *Research Directions in Object-Based Concurrency*, vol. chapter 1, G. A. W. a. A. Y. editors, Ed.: MIT Press, 1993, pp. 107-150.

[Mc Gregor-1993] J. D. Mc Gregor et D. M. Dyer, "A Note on Inheritance and State Machines", *ACM SIGSOFT Software Engineering Notes*, vol. 18, n° 4, pp. 61-69, 1993.

[Menasche-1982] M. Menasche, "Analyse des réseaux de Petri temporisés et applications aux systémes distribués", Thèse de doctorat, Université Paul Sabatier, Toulouse, 1982.

[Merlin-1974] P. Merlin, "A Study of the Recoverability of Computer System", Dep. Comput. Sci., Univ. California, Irvine, 1974.

[Merlin-1976] P. M. Merlin et D. J. Farber, "Recoverability of Communication Protocols - Implication of a Theoretical Study", *IEEE Transactions on Communications*, vol. 24, n° 9, pp. 1036-1043, 1976.

[Merseguer-2002] J. Merseguer, S. Bernardi, J. Campos et S. Donatelli, "A Compositional Semantics for UML State Machines Aimed at Performance Evaluation", the 6th International Workshop on Discrete Event Systems, Zaragoza, Spain, 2002.

[Merseguer-2001] J. Merseguer, J. Campos et E. Mena, "Performance Analysis of Internet Based Software Retrieval Systems Using Petri Nets", 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems, within the 7th International Conference on Mobile Computing and Networking, Rome, 2001.

[Meyer-1988] Meyer, "Object-Oriented Software Construction", Prentice Hall, 1988.

[Meyer-1999] B. Meyer, "The Significance of Components", *Software development magazine*, november, http://www.sdmagazine.com/, 1999.

[Meyer-2000] B. Meyer, "Contracts for Components", *Software development magazine*, July, http://www.sdmagazine.com/, 2000.

[Meyer-2001] E. Meyer, "Développements formels par objets : Utilisation Conjointes de B et d'UML", PhD., Université de Nancy 2, Nancy, 2001.

[Mikhajlov-1997] L. Mikhajlov et E. Sekerinski, "The Fragile Base Class Problem and its solution", Turku center for computer Science, Turku TUCS TR 117, june 19971997.

[Milner-1980] R. Milner, "A Calculus of Communicating System", LNCS 92, New York, 1980.

[Mitchell-1996] S. E. Mitchell et A. J. Wellings, "Synchronisation, Concurrent Object-Oriented Programming and the Inheritance Anomaly", *Computer Languages*, vol. 22, n° 115-26, 1996.

[Motus-1992] L. Motus, " Time Concepts in Real-Time Software", IFAC/IFIP Int. Workshop on real-time programming, Bruges, Belgium, 1992.

[Muller-1997] P. A. Muller, " Modélisation objet avec UML" Eyrolles, 2-212-08966-X, 1997.

[OMG-1997a] OMG, "Object Meta Object Facility (MOF) Specification", OMG, Framingham AD/97-08-14, Framingham,1997a.

[OMG-1997b] OMG, " UML Notation Guide version 1,1", www.rational.com, 1997b.

[OMG-1998] OMG, "Action Semantics for the UML, Request For Proposal", Request for Proposal ad/98-11-01, November 1998.

[OMG-1999a] OMG, " UML Semantic Guide version 1.3", Object Management Group, www.omg.org, 1999a.

[OMG-1999b] OMG, " UML version 1.3", Object Management Group, www.omg.org, 1999b.

[OMG-2000] OMG, "UML Action Semantics V1.0", OMG, the Action Semantics Consortium 31 may 2000.

[OMG-2002a] OMG, "Action Semantics for the UML", OMG http://www.omg.org/cgi-bin/doc?ptc/02-01-09, february2002a.

[OMG-2002b] OMG, "MOF 2.0 Query / Views /Transformations RFP", OMG **ad/02-04-10**, 2002b.

[OMG-2002c] OMG, "UML 1.5 (Recommended Available Specification - Action Semantics FTF outcome)", OMG http://www.omg.org/cgi-bin/doc?ptc/2002-09-02, 09/02, 2002c.

[OMG-2002d] OMG, "UML Profile for Scheduling final adopted specification", OMG http://www.omg.org/cgi-bin/doc?ptc/2002-03-02, 2002, 2002d.

[Ostroff-1987] J. S. Ostroff et W. Wonham, "Modeling and Verifying Real-Time Embedded Computer Systems", IEEE Real-Time Symposium, 1987.

[Paech-1994] B. Paech et B. Rumpe, "A new Concept of Refinement used for Behaviour Modelling with Automata", Formal Methods Europe'94, 1994.

[Palsberg-1992] J. Palsberg et M. Schwartzbach, "Three Discussions on Object-Oriented Typing", *ACM SIGPLAN OOPS Messenger*, vol. 3, n° 2, pp. 31-38, 1992.

[Paludetto-1991] M. Paludetto, "Sur la commande de procédés industriels : une méthodologie Orientée Objets et Réseaux de Petri", Thèse de doctorat, Université Paul Sabatier, Toulouse, 1991.

[Paludetto-1999] M. Paludetto et J. Delatour, "UML et les réseaux de Petri : vers une sémantique des modèles dynamiques et une méthodologie de développement des systèmes temps réel", *L'objet*, vol. 5, n° 3-4, pp. 443-467, 1999.

[Paludetto-1993] M. Paludetto et S. Raymond, "A Methodology based on Objects and Petri Nets for the Development of Real-Time Software", IEEE International Conference on Systems, Man and Cybernetics, Le Touquet-France, 1993.

[Papathomas-1997] M. Papathomas, J. Hernandez, J. M. Murillo et F. Sanchez, "Inheritance and Expressive power in Concurrent Object-Oriented Programming", Proceedings of Langages et Modèles à Objets '97, Roscoff, France, 1997.

[Pennaneac'h-2001] F. Pennaneac'h, "UML : de l'action à la réflexion", PhD, Université de rennes 1, Rennes, 2001.

[Petri-1962] C. A. Petri, "*Kommunikation mit Automate*", *Bonn: Institut für Instrumentelle, Mathematik, Schriften des IIM Nr. 3* , 1962.

[Pettit-2000] R. G. Pettit et H. Gomaa, "Validation of Dynamic Behavior in UML Using Colored Petri Nets", Workshop on Dynamic Behaviour in UML Models: Semantic Questions, UML 2000 Conference, York, England, 2000.

[Pettit-2001] R. G. Pettit et H. Gomaa, "Modeling State-Dependent Objects using Coloured Petri Nets", MOCA'01, Workshop on Modelling of Objects, Components, and Agents, Aarhus, Denmark, 2001.

[Pomello-1992] L. Pomello, G. Rozenberg et e. al, "A Survey of Equivalence Notions of Net Based Systems", in *Advances in Petri Net 92*, Rozenberg, Ed. Berlin: Springer-Verlag, 1992, pp. 410-472.

[Pooley-1998] R. J. Pooley et C. Kabajunga, "Simulation of UML Sequence Diagrams", UK PEW '98 - 14th UK Performance Engineering Workshop, Edinburgh, 1998.

[Pradin-chézalviel-1999a] B. Pradin-chézalviel, L. A. Künzle, F. Girault et R. Valette, "Evaluation temporelle de

scénario de réseau de Petri incluant du parallélisme", Congrès sur la Modélisation des Systèmes Réactifs (MSR'99), Paris, France, 1999a.

[Pradin-Chézalviel-2000] B. Pradin-Chézalviel et R. Valette, "Accessibilité de marquage et logique linéaire dans un réseau de Petri t-temporel", Journées Formalisation des Activités Concurrentes, FAC'2000, CERT-IRIT-LAAS, Toulouse, 2000.

[Pradin-Chézalviel-2003] B. Pradin-Chézalviel et R. Valette, "Réseaux de Petri et Logique Linéaire", in *Vérification et mise en oeuvre des réseaux de Petri*, S. l. d. d. M. Diaz, Editions Hermès, 2003, chapitre 6, p.209-229.

[Pradin-chézalviel-1999b] B. Pradin-chézalviel, R. Valette et L. A. Künzle, "Formalisation de scénarios, réseaux de Petri et logique linéaire", FAC 99, Toulouse, LAAS-CNRS, 1999b.

[Pradin-chézalviel-1999c] B. Pradin-chézalviel, R. Valette et L. A. Künzle, "Scenario duration characterization of t-timed Petri nets using linear logic", EEE PNPM'99, 8th International Workshop on Petri Nets and Performance Models, Zaragoza, Spain, September 6-10, 1999c.

[Ramchandani-1974] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems using Petri Nets", Laboratory for Computer Science, MIT, Cambridge 120,1974.

[Reggio-2001] G. Reggio, M. Cerioli et E. Astesiano., "Towards a Rigorous Semantics of UML Supporting its Multiview Approach", *In Proc. FASE 2001 (H. Hussmann Editor). Lecture Notes in Computer Science*, vol. 2029, Berlin, Springer Verlag, 2001.

[Reggio-1999] G. Reggio et R. J. Wierenga, "Thirty One Problems in the Semantics of UML 1.3 Dynamics", Workshop "Rigourous Modeling and Analysis of the UML Challenges and Limitations" in OOPSLA'99, Denver, Colorado, 1999.

[Reitzner-1996] S. Reitzner, "Synchronizing Classes: Splitting Inheritance Concepts", Univ. of Erlangen-Nuernberg, IMMD IV TR-I4-96-04,http://citeseer.nj.nec.com/142394.html, 1996.

[Rivière-2001] N. Rivière, B. Pradin-Chézalviel et R. Valette, " Reachability and temporal conflicts in t-time Petri nets", IEEE 9th International Workshop on Petri Nets and Performance Models (PNPM'01), Aachen (Germany), 2001.

[Robbins-1998] J. Robbins, "ArgoUML", http://argouml.tigris.org/, 1998.

[Roubtsova-2001] E. E. Roubtsova, J. v. Katwijk et W.J.Toetenel, "Transformation of UML Specification to XTG", the Andrei Ershov Fourth International Conference, Novosibirsk, 2001.

[Saldhana-2001] J. Saldhana, S. M. Shatz et Z. Hu, "Formalization of Object Behavior and Interactions From UML Models", *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 2001.

[Saldhana-2000] J. A. Saldhana et S. M.Shatz, "UML diagrams to Object Petri Net Models: An Approach for Modeling and Analysis", International Conference on Software Engineering and Knowledge Engineering, 2000.

[Saqui-Sannes-2001] P. d. Saqui-Sannes, L. Apvrille, C. Lohr, P. Sénac et J.-P. Courtiat, "UML et RT-LOTOS : Vers une intégration informel/formel au service de la validation de systèmes temps réel", Colloque Francophone sur la Modélisation des Systèmes Réactifs (MSR'2001), Toulouse (France), 2001.

[Saqui-Sannes-2000] P. d. Saqui-Sannes, J.-P. Courtiat, C. Lohr et P. Sampaio, "TURTLE: A Timed UML and RTLotos Environment", Workshop on Formal Methods for Real Time UML, UML'2000 conference, York, UK, 2000.

[Schäfer-2001] T. Schäfer, A. Knapp et S. Merz, "Model checking UML State Machines and Collaborations", *Electronics Notes in Theorical Computer Science*, vol. 47, n° 2001, pp. 1-13, 2001.

[Schmiedel-1998] A. Schmiedel, "Temporal Constraints Network", Université technologique, Berlin 69,1998.

[Schöf-1995] S. Schöf, M. Sonnenschein et R. Wieting, "Efficient Simulaton of THOR Nets", 16th Int. Conf. on Application and Theory of Petri Nets, Torino, Italy, 1995.

[Schrefl-1995] M. Schrefl et M. Stumptner, "Behavior Consistent Extension of Object Life Cycles", in *Object-Oriented and Entity-Relationship Modelling*, 1995, pp. 133-145.

[Selic-1994] B. Selic, G. Gullekson et P. Ward, "Real-Time Object Oriented Modelling", John Willey and sons, 1994.

[Selic-1998] B. Selic et J. Rumbaugh, "Using UML for Modeling Complex Real-Time Systems", ObjecTime Limited, 1998.

[Sénac-1996] P. Sénac, "Contribution à la modélisation des systèmes multimédias et hypermédias", PhD., Université Paul Sabatier, Toulouse, 1996.

[Sendall-2001] S. Sendall et A. Strohmeier, "Specifying Concurrent System Behavior and Timing Constraints Using OCL and UML", UML 2001 - The Unified Modeling Language: Modeling Languages, Concepts and Tools, Fourth International Conference, Toronto, Canada, 2001.

[Shin-1994] K. Shin et P. Ramanathan, "Real-time computing : A new Discipline of computer science and Engineering", *Proceedings of the IEEE*, vol. 28, n° 1, 1994.

[Sibertin-Blanc-1985] C. Sibertin-Blanc, "High-Level Petri Nets with Data Structure", 6th European Workshop on Petri Net and Application, Espoo (Finland), 1985.

[Sibertin-Blanc-1991] C. Sibertin-Blanc et R. Bastide, "Object-Oriented Structuring using High-Level Petri Nets", *Lecture Notes in Computer Science, Advances in Petri Nets'91*, pp. 1-24, 1991.

[Sifakis-1977] J. Sifakis, "Etude du comportement permanent des réseaux de Petri Temporisés", Journées AFCET sur les réseaux de Petri, Paris, 1977.

[Sifakis-1979] J. Sifakis, "Performance evaluation of systems using nets", *Net theory and applications : Lecture Notes in Computer Science*, vol. 84, pp. 307-319, 1979.

[Simons-2000] A. Simons, "On the Compositional Properties of UML Statecharts Diagrams", Electronic Workshops in Computing: Rigorous Object-Oriented Methods 2000, 2000.

[Simons-1998] A. Simons et I. M. Graham, "37 Things That Don't Work in Object Modelling with UML", in *Proc. 2nd. ECOOP Workshop on Precise Behavioural Semantics*, vol. http://www.dcs.shef.ac.uk/ajhs/abstracts.html#uml, H. Kilov and B. Rumpe, 1998.

[Simons-1999] A. J. H. Simons et I. Graham, "30 things that go wrong in object modelling with UML 1.3", in *Precise Behavioral Specification of Businesses and Systems*, Chap. 17, R. B. a. S. I. Kilov H, Ed.: Kluwer Academic Publishers, 1999, pp. 237-257.

[Sourrouille-1998] J.-L. Sourrouille, "Modélisation par la méthode OMT-2 avec UML", Real-Time System'98, Paris, 1998.

[Spivey-1989] J. M. Spivey, "The Z Notation : a Reference Manual", Prentice Hall, 1989.

[Stankovic-1988] J. A. Stankovic, " Misconception about Real-Time Systems - a serious problem for next generation systems", *IEEE Computer*, vol. October, pp. 10-19, 1988.

[Stein-1987] L. A. Stein, "Delegation is inheritance", OOPSLA'87, 1987.

[Sunyé-2002] G. Sunyé, A. LeGuennec et J.-M. Jézéquel, "Using UML action semantics for model execution and transformation", *Information Systems Elsevier*, vol. 27, n° 6, July 2002, pp. 445--457, 2002.

[Szyperski-1998] C. Szyperski, "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, 1998.

[Telelogic-1999] Telelogic, "Real-time SDL/UML revolution", *Signals, Telelogic news letter*, vol. 3, 1999.

[Toetenel-2001] H. Toetenel, E. Roubtsova et J. v. Katwijk, "A Timed Automata Semantics for Real-Time UML Specifications", IEEE Symposia on Human-Centric Computing Languages and Environments (HCC'01), Visual Languages and Formal Methods (VLFM'01), Stresa, Italy, 2001.

[Tsai-1995] J. J. P. Tsai, S. J. Yang et Y.-H. Chang, "Timing constraint Petri Nets and their application to schedulability analysis of real-time system specification", *IEEE Transactions on Software Engineering*, vol. 21, n° 1, pp. 32-49, 1995.

[Union-1992] I. T. Union, "Specification and Description Language", International Telecommunication Union Recommendation Z.100, 1992.

[Valette-1979] R. Valette, "Analysis of Petri Nets by Stepwise Refinements", *Journal of Computer and System Sciences*, vol. 18, n° 1, pp. 35-46, 1979.

[Valette-2000] R. Valette, "Cours sur les réseaux de Petri", LAAS, Toulouse,www.laas.fr/~robert, 2000.

[Valette-1985] R. Valette, V. Thomas et S. Bachmann, " SEDRIC, un simulateur à événements discrets basés sur les réseaux de Petri", *RAIRO/APII*, vol. 19, n° 5, pp. 423-436, 1985.

[Valk-1996] R. Valk, "On Processes of Object Petri Net", University of Hamburg FBI-HH-B-185/96,1996.

[Valk-2000] R. Valk, "Relating Different Semantics for Object Petri nets", TGI - Theoretical Foundations of

Computer Science Group, Computer Science, University of Hamburg, Hamburg B-226-00, june 2000.

[Van der Aalst-2002] W. M. P. Van der Aalst, "Inheritance of Dynamic Behaviour in UML", MOCA'02 , Second Workshop on Modelling of Objects, Components, and Agents, Aarhus, Denmark, 2002.

[Van der Aalst-1997] W. M. P. Van der Aalst et T. Basten, "Life-Cycle Inheritance : A Petri Net Based Approach", 18th ICATPN, Application and Theory of Petri Nets, Toulouse, France, 1997.

[Verkoulen-1998] P. A. Verkoulen, "Integrated Information systems Design", University of Eindhoven, Eindhoven RT_EN_98,1998.

[Weck-1997] W. Weck, "Inheritance Using Contracts & Object Composition", International Workshop on Component-Oriented Programming WCOOP'97, Turku Centre for Computer Science, 1997.

[Wegner-1988] P. Wegner et S. B. Zdonik, "Inheritance as an Incremental Modification Mechanism or What like is and isn't like", ECOOP'88, Oslo, Norway, 1988.

[Wilson-1988] B. Wilson, "Systems : Concepts, Methodologies and Applications", John Wiley & Sons, 1988.

[Wirsing-1993] M. Wirsing, "Développement de logiciel et spécification formelles", *Techniques et Sciences Informatiques*, vol. 12, n° 4, pp. 413-431, 1993.

[Wirtz-2000] G. Wirtz et H. Giese, "Using UML and Object-Coordination-Nets for Workflow Specification", IEEE International Conference on Systems, Man, and Cybernetics (SMC'2000), Nashville, TN, USA, 2000.

[Yu-2002] H. Yu, X. He, Y. Deng et L. Mo, "A Formal Method for Analyzing Software Architecture Models in SAM", COMPSAC2002, Oxford, U.K, 2002.

[Zapf-1999] M. Zapf et A. Heinzl, "Techniques for integrating Petri Nets and Object-Oriented Concepts", University of Bayreuth, Bayreuth, Germany Working papers, 4 august 1999,http://wi.oec.uni-bayreuth.de/mitarbeiter/zapf/, 1999.

# Partie VII Annexe : règles de dérivation des diagrammes de séquence



Cette annexe présente le processus de dérivation et la liste des règles de traduction des éléments UML présents dans un diagramme de séquence vers un diagramme à RdP de comportement externe.

# VII.1. Processus de passage

Un diagramme de séquence décrit une histoire particulière impliquant différents objets. C'est donc le comportement externe de ces objets qui est décrit et non leur fonctionnement interne. Par conséquent, le diagramme à RdP qu'il est possible de générer (ou de compléter), à partir de diagrammes de séquence, est un diagramme à RdP de comportement externe (eBPN).

Chacun des CO actifs du diagramme est traité en reportant directement les informations sur son RdP de comportement externe.

Pour chaque ligne de vie du diagramme de séquence, quatre types d'information sont distingués, se rapportant :

1. Aux communications entrantes et sortantes.
2. Aux effets observables des invocations.
3. Aux contraintes temporelles.
4. Aux alternatives et boucles.

Chaque type d'information sera traité dans l'ordre donné ci-dessus lors du processus de traduction.

Rappelons que ce processus de dérivation est semi-automatique et doit être plutôt vu comme un guide de traduction pour un analyste-concepteur. En effet, lorsqu'il y a ambiguïté ou une demande d'information supplémentaire, ce sera à l'analyste-concepteur d'y répondre. Par ailleurs, le RdP produit à partir de ces règles de dérivation n'est pas toujours complet et devra être enrichi manuellement si nécessaire.

# VII.2. Les communications entrantes et sortantes

Toute communication entrante ou sortante sur la ligne de vie d'un objet est traduite par un ensemble de places et de transitions sur le RdP de comportement externe de l'objet.

## VII.2.1. Règles de dérivation des communications entrantes

Une communication entrante (opération mX[62](args_mX)) est traduite :

- Par une place partagée symbolisant l'invocation (nommée *mX*),
- Une transition interne d'acceptation de la communication (nommée *Init_mX*) et
- Une place modélisant la prise en compte par l'eBPN de l'invocation (nommée *I_mX*).

Si la communication est asynchrone, la consommation du jeton est modélisée par une transition puits (*Eat_mX*) de traitement supposé. Cette transition est complétée par d'autres places et transitions si cette communication à des effets observables.

Si la communication est synchrone, l'acquittement (pour une communication synchrone faible) ou la réponse (pour une communication synchrone forte) doivent être modélisés. Une communication asynchrone (opération m1) et des communications synchrones faible (*m2*) et forte (*m3*) sont données en Figure VII.1, .

La dénomination des différentes places et transitions générées automatiquement dans ce processus de traduction suit toujours les mêmes règles. C'est par ces conventions de nommage que sont effectués les liens entre les places et les transitions du diagramme à RdP durant les différentes étapes de traduction.

[62] Où X représente un numéro.

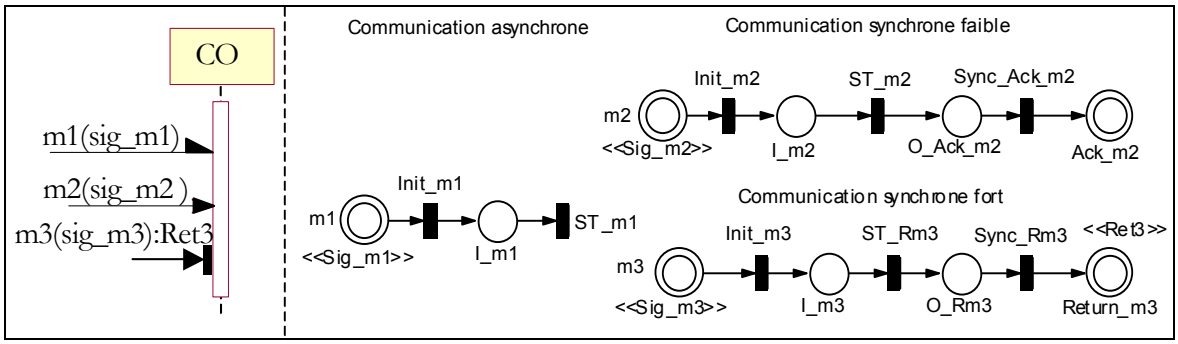


**Figure VII.1 : Dérivation des communications entrantes**

Notons qu'il est possible de représenter les signatures des opérations, ainsi que le type des valeurs retournées. Cela se fait en indiquant sur les places partagées quels types de jetons sont acceptés. Cela implique alors d'utiliser des RdP de haut niveau. Par exemple, pour la communication *m1* en Figure VII.1, nous avons indiqué sur la place *m1* que les jetons reçus étaient de type *sig_m1*.

## VII.2.2. Règles de dérivation des communications sortantes

Les communications sortantes sont traduites suivant les mêmes principes que pour les communications entrantes. Les correspondances avec les communications sortantes sont données en Figure VII.2. Notons les places d'attente (acquittement ou retour) pour les invocations d'opérations synchrones (*Wait_Ack* ou *Wait_Ret*). Ces places, outre le fait de modéliser l'attente de l'objet appelant, conservent l'historique des invocations afin de s'assurer, lors du retour ou de l'acquittement, de la correspondance des messages.

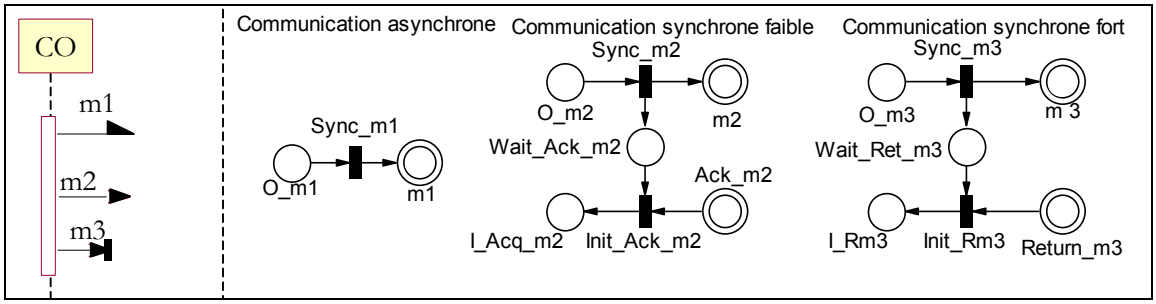


**Figure VII.2 : Dérivation des communications sortantes**

Comme pour les communications entrantes, il est tout à fait possible de représenter les signatures des opérations. Afin de ne pas complexifier les schémas et les explications, elles ne sont pas ici modélisées.

## VII.2.3. Le cas particulier des communications sortantes avec attente limitée

La représentation des communications avec attente limitée est complexe. Détaillons les difficultés rencontrées :

- Une même opération peut être invoquée au sein d'un CO avec ou sans attente limitée. La représentation de l'activation ou non de la garde temporelle lors de l'invocation d'une opération est délicate à représenter sur l'eBPN. Ce sont, en effet, des informations concernant le comportement interne du CO. Par ailleurs, l'alternative lors de l'expiration du délai d'attente peut dépendre, elle aussi, du comportement interne du CO. L'analyste-concepteur doit pouvoir faire le choix de ne pas représenter cette attente limitée. Elle est alors considéré comme une

communication synchrone simple, avec présence d'alternatives (cas sans ou avec expiration de l'attente). Dans des cas particuliers, l'analyste-concepteur peut vouloir représenter cette communication avec attente limitée sur l'eBPN. Au minimum, il faut prévoir un mécanisme prenant en compte l'arrivée trop tardive du retour d'une invocation alors que l'alternative a été lancée. La difficulté ne vient pas tant de la modélisation de ce mécanisme que du moment de son activation.

- La description de l'alternative (par exemple l'expiration) ne peut se faire que sur un autre diagramme de séquence UML (cf. I.4.2.2a), mais il est difficile d'indiquer alors, et ce de manière automatique sans ajouter d'information supplémentaire, quelle partie précise d'un diagramme de séquence donné décrit l'alternative.

Nous donnons en Figure VII.3 les mécanismes minimaux modélisant la traduction sur un eBPN d'une communication avec attente limitée. Nous ne faisons pas de différence entre une communication de type « *balking* » (invocation m4 sur la Figure VII.3) ou d'attente limitée. La première est considérée comme une communication avec une attente limitée d'une durée infinitésimale. Notons que, dans le cas d'un message à attente limité, le chien de garde n'est modélisé que sur le client (et non sur le serveur[63]).

Le chien de garde dans une communication avec attente limitée est modélisé par des places et des transitions de traitement supposé. Détaillons leurs rôles respectifs :

- Une transition temporelle *ST_WD* (pour *Supposed Treatment Watch Dog*) modélise l'expiration de l'attente.
- La place *ST_Acti_WD_M5* modélise l'activation du délai d'attente.
- La place *ST_Alt_WD_M5* représente le fait qu'une alternative doive être effectuée.
- La place *ST_LateRetm5* et la transition puits *ST_Ignore_Retm5* permettent de ne pas tenir compte des réponses trop tardives de l'invocation m5.

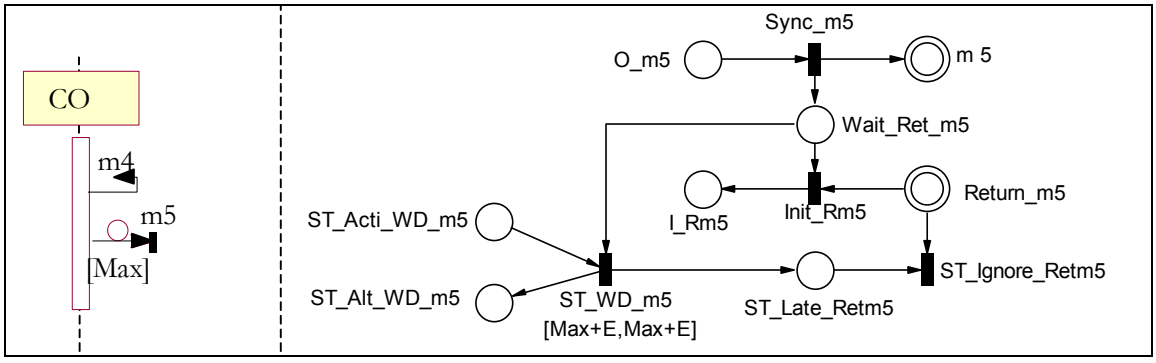


**Figure VII.3 : Dérivation d'une communication sortante avec attente limitée**

A partir des informations sur la communication sortante, il n'est pas possible de décrire l'activation ou non du chien de garde, ni même de décrire l'alternative à mettre en place. Ces informations seront complétées lorsque nous décrirons la traduction des alternatives.

[63] En effet, poser un chien de garde (correspondant à l'attente limitée de l'appelant) sur le serveur est déconseillé dans un système distribué où les communications prennent du temps. Si le chien de garde est posé sur le serveur (entre la réception du message et l'envoi de la réponse), il doit alors prendre en compte les temps de communication et les déduire. Un tel chien de garde sur le serveur ne sera pas établi automatiquement par la construction du diagramme à RdP.

# VII.3. Les effets observables des invocations

## VII.3.1. Principes

Les effets observables d'un eBPN sont les réactions observables de l'objet à l'invocation d'une méthode. Il s'agit, dans la plupart des cas, de l'invocation d'une méthode d'un autre CO. Ces effets sont représentés par des places et transitions de traitement supposé sur l'eBPN.

Le report de ces effets observables à partir des diagrammes de séquence consiste à analyser les lignes de vie des objets. Pour chaque invocation sur une ligne de vie, nous vérifions s'il n'y a pas une requête associée. Cette vérification se fait en posant la question à l'analyste-concepteur.

## VII.3.2. Ambiguïtés d'interprétation des effets observables

Mais quelques difficultés, dues aux différentes représentations possibles de ces effets observables sur le diagramme de séquence, sont rencontrées.

Nous avons représenté, en Figure VII.4, le fait que l'invocation de l'opération Op1 a pour conséquence l'appel de l'opération Op2. Dans UML 1.3, deux représentations sont possibles[64]. Cependant, seule la représentation de droite avec les barres d'activation (*activation bar*) est non-ambiguë. En effet, le diagramme de gauche sur la Figure VII.4 peut simplement traduire une relation d'ordre posant une contrainte de causalité : l'Op2 ne peut être appelée que si l'Op1 a déjà été invoquée. Cette interprétation diffère d'un effet observable, car l'invocation de l'Op1 n'entraîne pas automatiquement la requête de Op2. En outre, une autre interprétation du diagramme de gauche est possible : considérer qu'il n'y a aucune relation d'ordre entre Op1 et Op2, puisque, dans un diagramme de séquence, les relations entre les communications décrivent un ordre partiel et non total.

Face à ces ambiguïtés d'interprétation, deux stratégies de traduction sont possibles :

- Lorsqu'un diagramme de séquence est traduit, considérer que l'analyste concepteur représentera toujours les effets observables avec des barres d'activation.
- Ou bien, au contraire, penser que les ambiguïtés sont possibles. Il faut alors les lever en demandant à l'analyste-concepteur s'il y a un effet observable entre les invocations et les requêtes d'opérations.

La seconde stratégie consiste, en quelque sorte, à d'abord lever les ambiguïtés en questionnant l'analyste-concepteur puis, en fonction des réponses apportées, à ensuite enrichir le diagramme de séquence avec des barres d'activation. Nous nous ramenons ainsi à la première stratégie.

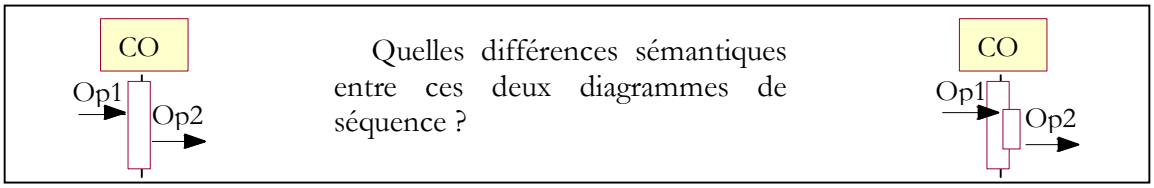


**Figure VII.4 : Une représentation ambiguë des diagrammes de séquence**

## VII.3.3. Règles de dérivation des effets observables

Un effet observable est traduit par un ensemble de places et de transitions de traitement supposé sur l'eBPN. Dans le cas du déclenchement d'une requête d'opération Op2 suite à une invocation Op1, l'effet observable est modélisé par une transition de traitement supposé, nommée *ST_Op1->Op2*. La Figure VII.5 modélise ce type de situation. Remarquons que la

[64] L'utilisation des barres d'activation dans UML 1.3 reste optionnelle.

transition puits modélisant la consommation du jeton (transition *eat_Op1* voir Figure VII.1) de l'opération asynchrone Op1 est alors remplacée par une transition de traitement supposé *ST_Op1->Op2* (cf. Figure VII.5). Ce nommage de la transition permet de garder une trace de la relation de causalité.

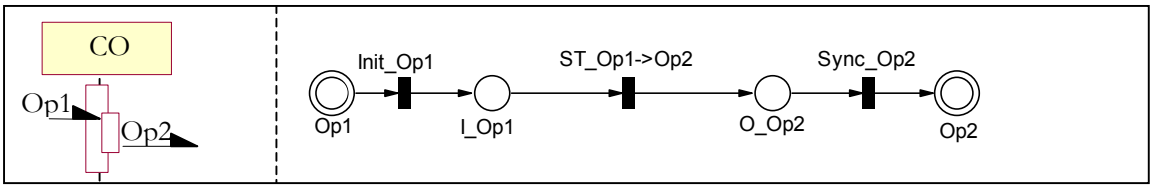


**Figure VII.5 : Principe de dérivation des effets observables d'une invocation**

D'autres cas peuvent être modélisés lorsque les effets observables sont plus complexes. Par exemple dans le cas de boucles, de requêtes périodiques ou d'invocations suite à une expiration temporelle, le comportement de l'eBPN devra être modélisé par un ensemble de places et de transitions.

# VII.4. Les contraintes temporelles

## VII.4.1. Principe

Une contrainte temporelle modélise une relation d'ordre entre deux événements. Dans un diagramme de séquence, ces événements ne peuvent être que des réceptions ou des envois de messages.

Lorsque les contraintes temporelles sont quantitatives, elles doivent être exprimées par des notes respectant le format défini dans UML/PNO.

Lorsque ces CT portent sur des CO, leur report sur des BPN reste très simple. Pour les CTA, il s'agit simplement de préciser leurs intervalles temporels représentant les relations d'ordres entre les communications. Pour les CTP, elles sont liées à des effets observables et sont représentées sur des transitions de traitement supposé de l'eBPN.

Toutefois, pour des relations d'ordre qualitatives, nous sommes confrontés, lors de la traduction, aux ambiguïtés des diagrammes de séquence, principalement par le fait que l'ordre des messages échangés entre les lignes de vie n'est que partiel.

## VII.4.2. Ambiguïtés d'interprétation des CT qualitatives

Nous avons représenté en Figure VII.6a) une partie d'un diagramme de séquence où des relations d'ordre semblent être possibles entre la communication entrante "m1" et les communications sortantes "m2" et "m3".

Différentes interprétations (en supposant qu'il y a bien au moins une relation d'ordre) peuvent être données. Nous avons choisi de distinguer graphiquement ces différentes interprétations. Naturellement, ces représentations ne respectent plus la norme UML. Elles sont seulement utilisées ici dans un but didactique. Dans le cas des vues b) et c) de la Figure VII.6, nous supposons qu'il y a une relation entre le message "m1" et les deux messages "m2" et "m3". Dans la vue d), en revanche, nous supposons que seule une des deux communications sortantes est en relation avec la communication "m1".

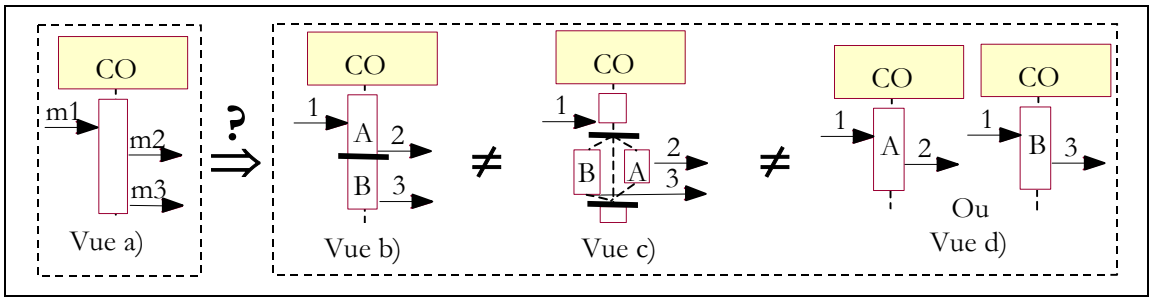


**Figure VII.6 : Représentation des relations d'ordre et ambiguïtés à lever**

La différence d'interprétation ente les vues b) et c) consiste à considérer, pour la première, qu'il y a une relation d'ordre entres les deux communications sortantes ("m3" ne peut être généré qu'après "m2"). En revanche, pour la seconde, aucune supposition n'est faite sur un ordre entre les messages "m2" et "m3". Un autre diagramme de séquence pourrait très bien montrer un ordre différent (la communication "m3" avant "m2" suite à la réception de "m1"). Les messages sortants sont donc considérés comme élaborés en parallèle.

Face à ces ambiguïtés d'interprétation, nous nous retrouvons dans la même situation qu'en VII.3.2 et deux stratégies de traduction sont possibles :

- Nous considérons que l'analyste-concepteur indiquera toujours quelles sont les relations d'ordre, et ce en utilisant des notes UML exprimant des CTA d'une durée non définie (l'intervalle temporel de ces CTA étant alors [0,+∞[)
- Aucune supposition ne peut être faite et des questions explicites sont posées à l'analyste-concepteur. En reprenant l'exemple de la Figure VII.6a), il faut alors demander s'il existe des relations d'ordre entre m1 et m2, entre m1 et m3 et entre m2 et m3.

Là encore, comme en VII.3.2, nous nous ramènerons toujours à la première stratégie, en enrichissant le diagramme de séquence par des notes UML représentant explicitement les relations d'ordre. De cette manière, nous ne faisons plus de différences entre le traitement des CT qualitatives et quantitatives.

## VII.4.3. Règles de dérivation des contraintes temporelles

Nous illustrons ces règles de passage en nous appuyant sur l'exemple de la Figure VII.6a).

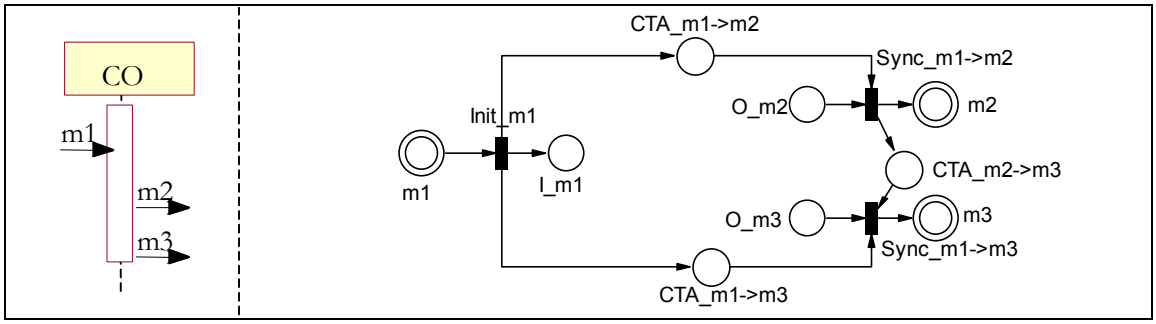


**Figure VII.7 : Dérivation d'une relation d'ordre entre un message entrant et deux messages sortants**

Si nous prenons l'interprétation donnée par la Figure VII.6b), nous supposons qu'il y a une relation entre m2 et m3, m3 devant être générée après m2. Cette relation est modélisée par une place temporelle, nommée "CTA_m2→m3" (cf. Figure VII.7).

Sinon, si nous supposons qu'il n'y a pas de relation d'ordre, la place CTA n'est pas représentée (interprétation de la vue c) de la Figure VII.6). Les deux interprétations b) et c) sont donc

similaires à l'exception d'une CTA qui définit la relation d'ordre entre les messages m2 et m3.

L'interprétation de la vue d) de la Figure VII.6 est un cas différent car il s'agit de la description d'une alternative : suivant l'état du CO ou du message m1, ce dernier peut entretenir une relation d'ordre avec m2 ou avec m3. Nous traitons ce cas en VII.5.

## VII.4.4. Les contraintes temporelles périodiques

Nous ne représentons les CT périodiques sur l'eBPN que lorsqu'elles décrivent une activation interne du CO ayant des effets observables (invocation d'opérations d'autres CO). En outre, un comportement périodique peut être en relation avec plusieurs CT : une CTP liée aux places et transitions horloge, mais aussi des CTA (généralement d'occurrence) sur la production de stimuli ou la réception de réponses périodiques.

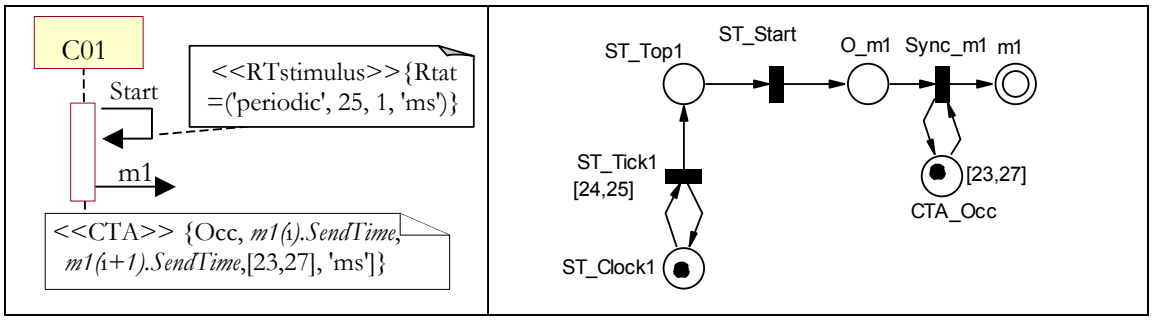


**Figure VII.8 : Dérivation des comportements périodiques**

Un exemple de dérivation d'un comportement périodique ayant une CTA d'occurrence associée est donné Figure VII.8.

## VII.4.5. Les CTA d'expressions complexes

Certaines CTA peuvent porter sur des conditions complexes faisant intervenir plusieurs occurrences de stimuli pour une réponse. Ainsi, la formule <<CTA>> {val, *m2([1-5],i=1).receiveTime*, *m3.sendTime()*, [10,50], 'ms'} exprime une CTA entre la première de cinq occurrences successives d'un stimulus m2 et d'une réponse m3.

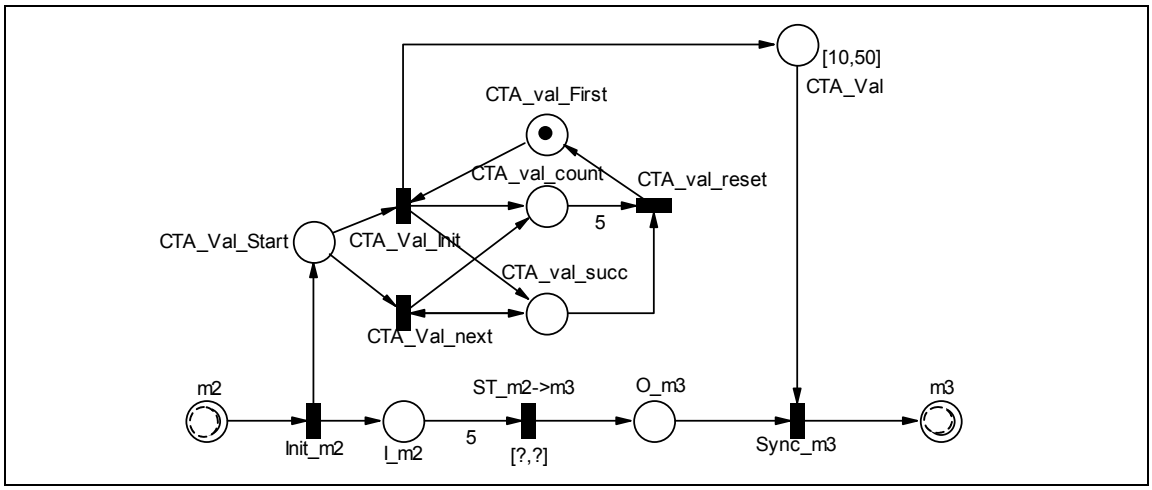


**Figure VII.9 : Dérivation de CTA complexes**

Sa traduction implique des mécanismes d'activation de la place CTA plus complexes que dans les cas simples faisant intervenir une seule occurrence. Il faut mettre en place un système de compteur afin de différencier la première (celle qui active la place CTA) des cinq occurrences successives (cf. Figure VII.9).

Afin de ne pas complexifier inutilement le RdP, nous proposons une représentation simplifiée de ce type de mécanisme d'activation de la place CTA. Nous parlons alors de places modulo. Par conséquent, le RdP de la Figure VII.9 peut être remplacé par celui donné en Figure VII.10.

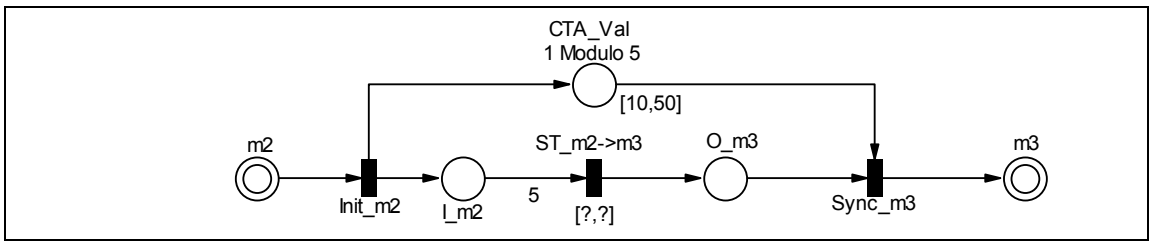


**Figure VII.10 : Représentation d'une place CTA modulo**

Cependant, d'autres CT sont rencontrées dans les diagrammes de séquence et ne sont pas représentables sur les eBPN des CO, car elles ne concernent pas le comportement des CO. Ce sont des CT :

- Décrivant le comportement temporel des acteurs du diagramme de séquence.
- Modélisant des temps de communications entre des CO.

Détaillons leurs traductions.

## VII.4.6. Les contraintes temporelles concernant les acteurs

Nous ne représentons que les informations de l'environnement qui imposent des contraintes temporelles sur le système. Nous avons identifié des comportements standards des acteurs (production périodique, apériodique ou sporadique de stimuli ou réception de réponses).

La Figure VII.11 propose différentes représentations courantes d'un stimulus ou d'une réponse suivant sa fréquence d'apparition, le protocole de communication utilisé (synchrone/asynchrone), le fait qu'une CTA soit demandée ou non…

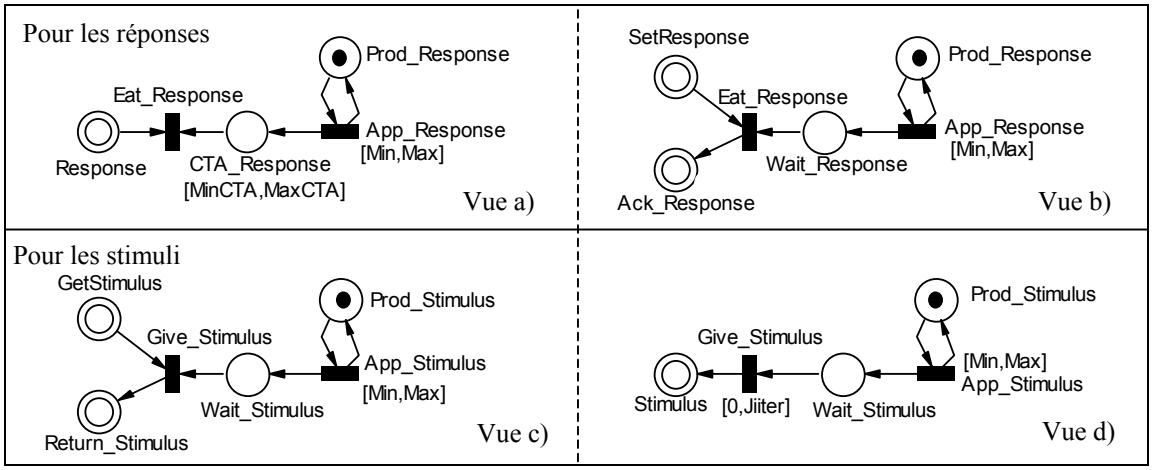


**Figure VII.11 : Représentations des stimuli et réponses dans un EPN**

Les bornes de l'intervalle temporel de la transition d'apparition du stimulus (*App_Stimulus*) ou de la réponse (*App_Response*) dépendent :

- De la périodicité (ou non) du comportement.
- De la référence temporelle utilisée (temps global ou relatif à l'occurrence précédente).

Dans le cas d'un stimulus périodique capturé par scrutation, exprimé dans un temps global, c'est la représentation de la vue c) qui est choisie, les bornes de la transition *App_Stimulus* sont

alors égales à la période moins sa gigue. Si la détection du non-respect de la CT associée était demandée, la place *Wait_Stimulus* serait remplacée par une place *CTA_Stimulus* avec un intervalle temporel couvant la gigue (comme pour la vue a)).

## VII.4.7. Les temps de communication entre CO

Les temps de communication entre deux CO sont représentés, lorsqu'ils ne sont pas négligeables, sur un objet **médium**. Cet objet est en quelque sorte représentatif d'une partie du médium de communication du système (d'où son nom). Sa représentation en UML est faite en définissant un objet stéréotypé <<medium>>.

Un objet médium ne fait que préciser les relations <<uses>> entre les CO. Il ne contient que des références sur les opérations des CO serveurs. Un EPN est associé à cet objet, il modélise les CT de communication.

Nous proposons, à la Figure VII.12, la représentation par EPN d'un objet médium, suivant le type de communication (synchrone ou asynchrone). Cet objet (« Med ») modélise l'invocation d'une opération Op1 entre un client et un serveur.

Les temps de communication sont indiqués sur les transitions de l'EPN de l'objet médium. Pour l'opération asynchrone, nous avons associé, à titre de rappel, une loi de tir fDC() à l'intervalle représentant le délai de communication. Cette loi indique la probabilité de tir de la transition entre les temps Min et Max de l'intervalle. Cela permet, par exemple, de modéliser des temps moyens de communication.

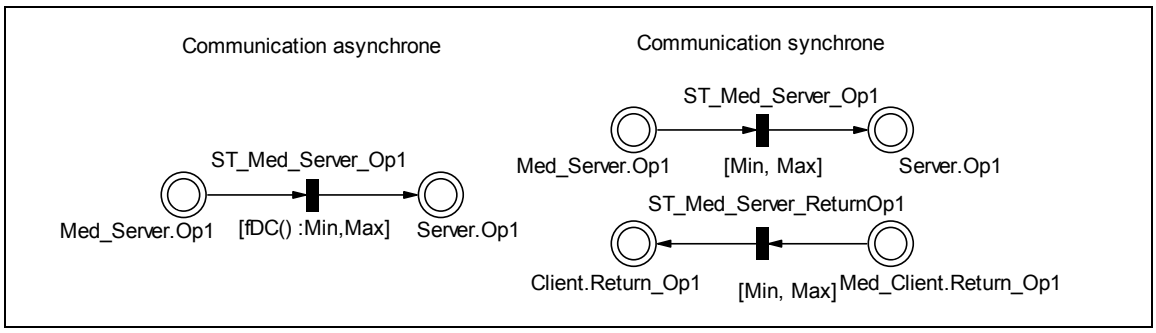


**Figure VII.12 : EPN d'un objet média**

Figure VII.13, nous donnons un exemple de traduction de temps de communication entre un objet client et un objet serveur. On suppose que l'objet <<medium>> est appelé « M_Média1 ».

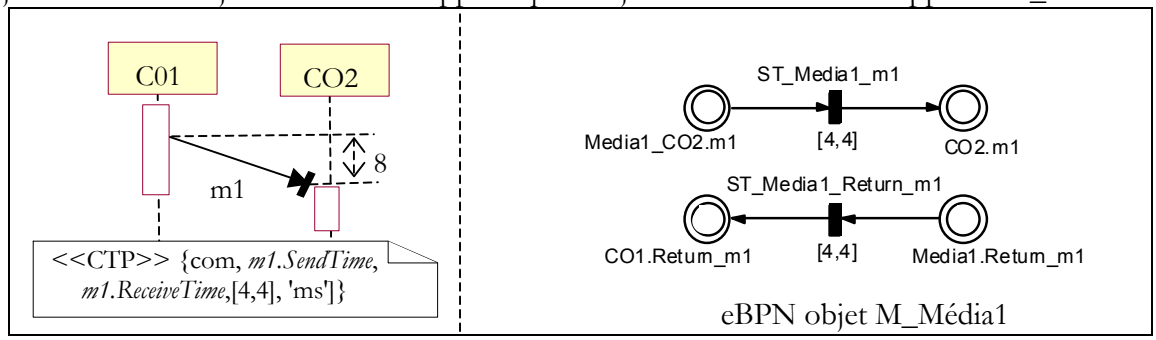


**Figure VII.13 : Dérivation des temps de communications**

Signalons l'une des ambiguïtés possibles d'interprétation si les notes temporelles ne sont pas utilisées. Graphiquement, un temps de 8 a été indiqué, mais rien ne nous permet de savoir si ce temps de communication vaut pour l'aller, pour le retour ou pour l'aller-retour. Dans le cas

présent, nous demanderons à l'analyste-concepteur de préciser le temps de communication (sur la figure, seule une note a été représentée pour l'aller).

# VII.5. Les alternatives

## VII.5.1. Principe

Une alternative peut être modélisée de deux façons avec des diagrammes de séquence, soit à l'aide de deux diagrammes de séquence présentant chacune des alternatives, soit sur un seul diagramme de séquence en indiquant des gardes sur les invocations.

La Figure VII.14 représente l'une de ces situations où, en fonction d'un même message en entrée, le CO1 va avoir un comportement différent et renvoie deux messages différents. Les vues a) et b) ont presque la même signification[65] : la première utilise deux diagrammes et la seconde un seul.

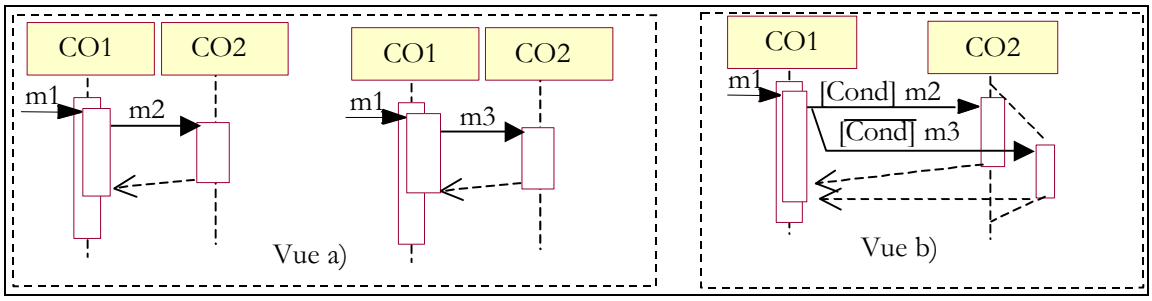


**Figure VII.14 : Représentation des alternatives en UML**

## VII.5.2. Les ambiguïtés d'interprétation des alternatives

Deux difficultés de traduction sont rencontrées :

- La première concerne l'identification de l'alternative, lorsque celle-ci est traduite par deux diagrammes de séquence.
- La seconde concerne la représentation de la condition.

Détaillons ces deux points.

### VII.5.2.1. Identification de l'alternative

Dans le cas où l'alternative est représentée sur un deuxième diagramme de séquence, il nous faut comprendre que c'en est une et identifier l'endroit où elle commence sur la ligne de vie du CO. Une détection automatique est possible dans les cas suivants :

- Lorsque l'alternative est liée a un effet observable (c'est le cas de la Figure VII.14).
- Lorsqu'il y a une relation d'ordre entre des communications.

Dans les deux cas, la détection n'est faite que lors de la traduction du second diagramme. En effet, lorsque nous reportons les informations liées à la relation d'ordre ou à l'effet observable, nous constatons qu'il existe déjà un ensemble de places et de transitions traduisant des informations (issues du report des éléments de modélisation du premier diagramme) en relation avec ces communications. Ce constat est possible grâce à la convention de nommage interne des éléments du RdP que nous utilisons. Si cet ensemble diffère des informations que nous voulions

[65] Dans la vue a), la condition de l'alternative n'est pas explicite, bien qu'elle puisse figurer sur une note du diagramme de séquence

reporter, il suffit alors de demander à l'analyste-concepteur s'il s'agit bien d'une alternative ou d'une incohérence.

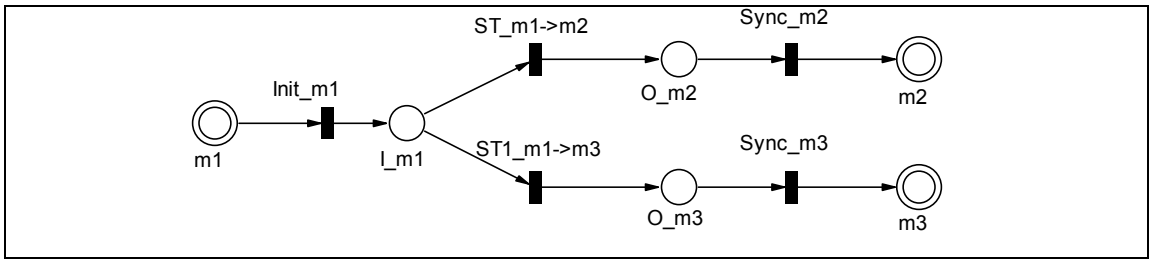


**Figure VII.15 : Représentation d'une alternative sur un eBPN**

Détaillons cette situation avec l'exemple de la Figure VII.14, traduit sur l'eBPN de la Figure VII.15 (vue a)). Lors du report du premier diagramme de séquence, le traitement supposé « ST_m1->m2 » est représenté. C'est en traduisant le deuxième diagramme, lorsqu'il s'agit de traduire l'effet invocable par une transition « ST_m1->m3 », que l'on s'aperçoit qu'une transition, portant un autre nom, part de la place I_m1. Cela est alors signe : soit d'une incohérence, soit d'une alternative. Une question est alors posée à l'analyste-concepteur afin qu'il lève l'ambiguïté.

### VII.5.2.2. Représentation de la condition

La deuxième difficulté de la traduction des alternatives vient de l'expression de la condition.

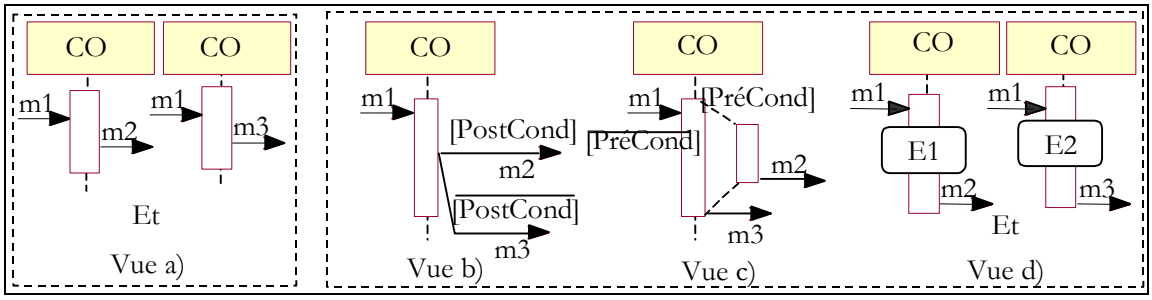


**Figure VII.16 : Différentes interprétations pour la condition d'une alternative en UML**

A partir de la Figure VII.16a), nous représentons une situation où, en fonction d'un même message en entrée, le CO va avoir un comportement différent et renvoie deux messages différents. Il nous semble que là encore réside une ambiguïté (devant être levée par l'analyste-concepteur) qui est de savoir quand cet indéterminisme sera résolu par l'objet. A l'aide de représentations UML non-standards (seulement utilisées dans un but didactique), nous donnons les trois interprétations possibles. La vue b) suppose que ce n'est qu'à la fin du traitement que l'indéterminisme est résolue (post-condition). La vue c) suppose que c'est dès la réception du message m1. La vue d) suppose que cela dépend de l'état interne de l'objet. Ainsi, plutôt que de différencier ces différents cas, nous préférons considérer que l'eBPN ne peut distinguer ces différentes situations. Nous modélisons donc les deux alternatives sans préciser sur l'eBPN comment l'indéterminisme est levé. C'est à l'analyste-concepteur d'enrichir manuellement, s'il le juge nécessaire, l'eBPN, en ajoutant, par exemple, des conditions sur le franchissement des transitions modélisant les alternatives.

## VII.5.3. Règles de dérivation des alternatives concernant les eBPN

Une fois qu'une alternative est clairement identifiée (grâce à un ensemble de questions posées à l'analyste-concepteur), chacune des branches de l'alternative est représentée par une transition

de traitement supposé. Ce principe a déjà été présenté à la Figure VII.15, qui est une traduction de la vue a) de la Figure VII.14.

# VII.6. Les boucles

## VII.6.1. Principe

Sur un diagramme UML, l'appel répétitif d'une opération peut être représenté par une garde précédant le nom de l'opération invoquée. Cette garde s'exprime sous la forme « *[i=1..n] », où n représente le nombre d'appels. Cette notation dénote des appels séquentiels de l'opération (cf. Figure VII.17). UML permet une représentation d'appels concurrents (par diffusion) par la notation « *//[i=1..n] ». Dans UML/PNO, rappelons que nous nous ramenons toujours à des communications peer to peer (et non par diffusion). De ce fait, une telle représentation sera interprétée comme une suite d'appels séquentiels.

De même, il est possible de représenter la notion de boucle « tant que » sur un diagramme de séquence. Il n'y a alors pas besoin de spécifier le nombre exact de répétitions : il suffit de donner la condition définissant l'arrêt de la répétition. Cela se fait par la garde « *[condition] » (cf. Figure VII.18). La traduction de cette garde rejoint le problème des alternatives et la difficulté de représenter une condition, surtout si celle-ci porte sur des informations internes du CO. Nous procéderons alors avec le même principe que celui des alternatives, en modélisant les deux traitements sans préciser comment l'indéterminisme est levé. Ce sera à l'analyste-concepteur de spécifier les conditions des transitions concernées de l'eBPN.

## VII.6.2. Règles de dérivation des boucles

### VII.6.2.1. Pour une boucle de type « pour »

Une boucle de type « pour », où n représente le nombre d'itérations, est traduite par un ensemble de places et de transitions (cf. Figure VII.17) tel que :

- Deux transitions de traitement supposé modélisent : l'une l'initialisation et l'autre l'itération elle-même. Sur la Figure VII.17, il s'agit respectivement des transitions *ST_m1->m2* et *ST_For_m1->m2*.
- Deux places de traitement supposé tiennent une comptabilité des itérations déjà effectuées et à effectuer. En Figure VII.17, il s'agit respectivement des places *ST_Start_m1->m2* et *ST_End_m1->m2*.

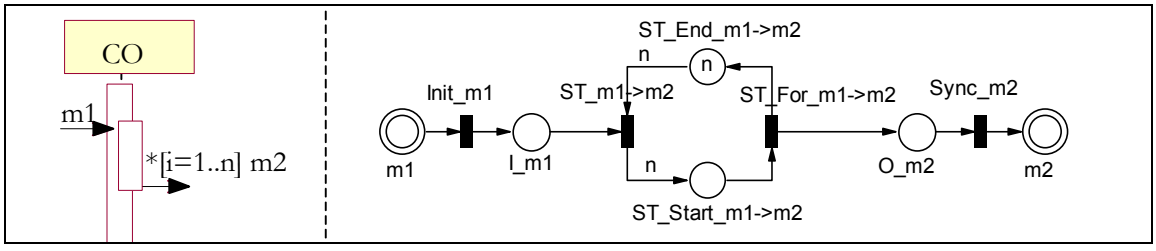


**Figure VII.17 : Dérivation d'une boucle de type « pour »**

Remarquons que l'initialisation de l'itération (transition *ST_m1->m2*) fait passer n jetons de la place *ST_End_m1->m2* à la place *ST_Start_m1->m2*, les arcs de cette transition reliant ces places étant de poids n.

### VII.6.2.2. Pour une boucle de type « tant que »

Une boucle de type « tant que » est traduite sur un eBPN par :

- Deux transitions de traitement supposé modélisant chacune des alternatives : l'une l'itération, commençant par les lettres « *ST_While* », et l'autre sa fin.
- Une transition de traitement supposé initialisant la boucle.
- Une place modélisant le point de décision, commençant par les lettres « *ST_Start* ».

La traduction d'une boucle effectuant une requête de l'opération m2 est donnée en Figure VII.18 . La fin de l'itération déclenche alors une requête d'une opération m3. Dans cet exemple, nous supposons que la condition porte sur des paramètres fournis par l'opération m1. Dès lors, la condition peut être portée sur les transitions *ST_While_m1->m2* et *ST_m1->m3*.

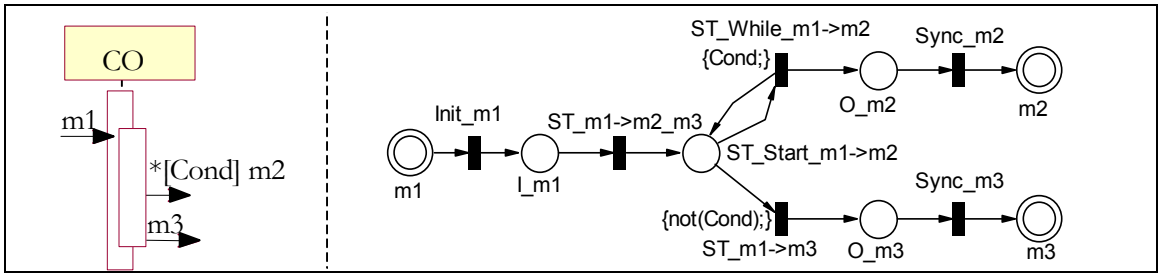


**Figure VII.18 : Dérivation d'une boucle de type « tant que »**

# VII.7. Les informations non traduites des diagrammes de séquence

Des informations présentes sur les diagrammes de séquence ne sont pas reportées sur l'eBPN :

- Il s'agit de la représentation des messages réflexifs non périodiques, car ce type d'information n'a pas à figurer, par définition, sur le RdP de comportement externe, mais sur le RdP de comportement interne.
- La création et la destruction des CO ne sont pas représentées puisque dans UML/PNO, l'allocation dynamique n'est pas permise. Toutes les allocations sont statiques, comme c'est souvent le cas dans la modélisation de STR fermes ou stricts.

## Contribution à la spécification des systèmes temps réels : l'approche UML/PNO

Ce travail présente l'approche UML/PNO (Unified Modelling Language with Petri Net Objects) pour la spécification de systèmes temps réel. La méthode propose d'enrichir la description semi-formelle UML du système par une modélisation formelle basée sur les réseaux de Petri. Les concepts UML de sous-système et d'interface ont été étendus afin d'améliorer la description de la vue système en termes de structuration, gestion de projet et organisation de la modélisation. L'objectif est également d'adapter la méthode de modélisation système à une approche « orientée composant » pour le temps réel. Le concept « d'objet composé » permet d'intégrer des spécificités temps réel au sein du composant (protocole de communication, contrainte temporelle, effet observable).

Le comportement des objets est spécifié à l'aide des réseaux de Petri à places et transitions stochastiques temporelles afin de permettre la validation et la vérification du système en cours de spécification. L'approche de validation propose des traductions semi-automatiques des diagrammes UML en réseaux de Petri. Les techniques classiques de simulation et d'évaluation de performances peuvent alors être appliquées. Ces traductions présentent l'avantage de rassembler, sur un même modèle à réseau de Petri, tous les aspects dynamiques du composant apparaissant dans différents diagrammes UML et de vérifier ainsi la cohérence de son comportement et de son utilisation. La vérification utilise les techniques d'analyse formelle, basées sur l'utilisation conjointe du graphe de classes et de la logique linéaire. Une approche de vérification quantitative des contraintes temporelles est proposée au niveau de l'analyse des besoins et des exigences du système. Une partie de ce travail a été concrétisée par l'intégration d'un module (ArgoPNO) dans l'atelier de génie logiciel ArgoUML. A ce jour, une première automatisation de la démarche de vérification des contraintes temporelles est opérationnelle sur cet outil.

Mots-clefs : UML temps réel, Réseaux de Petri, contraintes temporelles, analyse des besoins, spécification, validation et vérification.

## Towards the analysis of real-time systems: the UML/PNO approach

This work focuses on the UML/PNO (Unified Modelling Language with Petri Net Objects) approach for the specification of real-time systems. It enhances the UML dynamic diagrams with the Petri Nets (PN) formal language. The concept of real-time components, called "Compound Object", has been developed as an extension of the UML sub-systems. The aim is to improve the system modelling and structuring.

The behaviour of a Compound Object is described by means of time PN. The timing intervals are assigned to PN places and transitions. Thus, formal verification of timing constraints can be done. The validation approach relies upon on the semi-automatic translation of the UML dynamic diagrams into Petri Nets. During such a derivation, some UML diagrams inconsistencies are automatically detected. Then, existing PN tools can be used for simulation and performance analysis. The verification process uses a model-checking analysis based on linear state class graph. When a non-respected timing constraint occurs, a theorem proving approach based on linear logic is used in order to capture the causality relationships. A part of the UML/PNO approach has been integrated in a CASE Tool as an "ArgoPNO" module. It is now available in the ArgoUML environment.

Keywords: real-time UML, Petri nets, timing constraints, requirement analysis, specification, validation and verification.