

**République Algérienne Démocratique et Populaire**  
**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**  
**Université Mentouri Constantine**  
**Faculté Des Sciences de l'Ingénieur**  
**Département d'Informatique**

**Thèse de Doctorat en Sciences en Informatique**

**Thème :**

**Modélisation et Vérification des processus métiers dans  
les entreprises virtuelles : Une approche basée sur  
la transformation de graphes**

**Présentée par : Raida EIMansouri**

### **Composition du Jury**

|                                    |   |                   |
|------------------------------------|---|-------------------|
| <b>Benmohammed Mohamed</b>         | <b>Professeur, Université de Constantine</b>            | <b>Président</b>  |
| <b>Boufaida Zizette</b>            | <b>Professeur, Université de Constantine</b>            | <b>Rapporteur</b> |
| <b>Kholladi Mohamed Khireddine</b> | <b>Maître de Conférences, Université de Constantine</b> | <b>Examineur</b>  |
| <b>Meslati Djamel</b>              | <b>Maître de Conférences, Université de Annaba</b>      | <b>Examineur</b>  |
| <b>Benchikha Fouzia</b>            | <b>Maître de Conférences, Université de Skikda</b>      | <b>Examineur</b>  |

## Remerciements

J'adresse toute ma reconnaissance au Professeur Boufaïda Zizette qui a dirigé ce travail. Je la remercie pour sa patience et sa compréhension durant toutes ces années de thèse.

Ma profonde reconnaissance au Docteur Chaoui Allaoua sans qui, ce travail n'aurait jamais vu le jour. Je le remercie infiniment pour son apport scientifique incontestable, mais aussi pour toutes ses qualités humaines, sa compréhension, sa tolérance, sa patience et son soutien.

Mes remerciements vont aussi à Mr Kerkouche El Hilali pour sa coopération et son esprit d'équipe.

Je tiens à remercier le Professeur Benmohammed Mohamed de m'avoir fait l'honneur d'accepter de présider mon jury de thèse.

Je remercie également Dr Khoukhi Mohamed Khireddine, Dr Meslati Djamel et Dr Benchikha Fouzia d'avoir accepté de juger ce travail.

Je tiens à remercier tous mes collègues et amis qui m'ont soutenue toutes ses années et en particulier Mr Med Redha Bahri pour son aide et sa disponibilité.

ET MA FAMILLE (ma mère, ma sœur, mes frères et tous les autres) pour tout ce qu'ils ont fait et ce qu'ils font encore pour moi.

# Table des matières

|   |    |
|---|----|
| <b>Introduction</b>   | 1  |
| <b>1. Modélisation des entreprises</b>                                  | 5  |
| 1.1 Introduction .....  | 5  |
| 1.2 Modélisation des entreprises .....                                  | 5  |
| 1.3. Evolution de l'approche processus dans les organisations .....     | 6  |
| 1.4. Types de processus .....   | 7  |
| 1.5. Définition de processus .....                                      | 9  |
| 1.6.1 Modélisation des processus .....                                  | 12 |
| 1.6.2 Objectifs de la modélisation .....                                | 13 |
| 1.6.3 Critères de modélisation .....                                    | 14 |
| 1.7 Techniques de modélisation des processus métiers .....              | 15 |
| 1.7.1 Organigrammes (Flowcharts) .....                                  | 15 |
| 1.7.2 Les diagrammes des flux de données DFD (Data Flow Diagrams) ..... | 16 |
| 1.7.3 IDEF .....  | 18 |
| 1.7.3.1 IDEF0 .....   | 18 |
| 1.7.3.2 IDEF1 .....   | 19 |
| 1.7.3.3 IDEF2 .....   | 19 |
| 1.7.3.4 IDEF3 .....   | 20 |
| 1.7.4 Réseau de Pétri (RdP) .....                                       | 20 |
| 1.7.5 La méthode GRAI .....   | 20 |
| 1.7.6 CIMOSA .....  | 22 |
| 1.7.7 PERA .....  | 24 |
| 1.7.8 GERAM .....   | 25 |
| 1.7.9 Diagrammes d'enchaînement de processus d'ARIS .....               | 26 |
| 1.7.10. La Chaîne de Processus Événementielle (CPE) .....               | 26 |
| 1.7.11 BPMN .....   | 28 |
| 1.7.11.1 Notions de base .....  | 28 |
| 1.7.12. Unified Modelling Langage: UML .....                            | 30 |
| 1.7.12.1. Les Diagrammes UML .....                                      | 31 |
| 1.8. Evolution des organisations .....                                  | 33 |

|  |           |
|--|-----------|
| 1.8.1. Les organisations classiques .....                            | 34        |
| 1.8.2. Les nouvelles structures d'organisation .....                 | 35        |
| 1.8.2.1. Entreprise réseau .....                                     | 35        |
| 1.8.2.2. Entreprise virtuelle .....                                  | 36        |
| 1.8.2.3. Entreprise étendue .....                                    | 36        |
| 1.8.2.4. Réseau d'entreprises .....                                  | 37        |
| 1.9. Processus métiers et entreprises virtuelles .....               | 37        |
| 1.10. Conclusion .....   | 38        |
| <br>   |           |
| <b>2. Méthodes Formelles pour la vérification des systèmes</b> ..... | <b>39</b> |
| 2.1 Introduction .....   | 39        |
| 2.2 Critères d'un bon modèle .....                                   | 39        |
| 2.3 Techniques d'analyse .....                                       | 40        |
| 2.3.1. Vérification .....  | 40        |
| 2.3.2. Validation .....  | 40        |
| 2.3.3 Qualification .....  | 41        |
| 2.3.4. Certification .....   | 41        |
| 2.4 Vérification et validation .....                                 | 41        |
| 2.5 Les méthodes formelles .....                                     | 42        |
| 2.5.1 Les mythes autour des méthodes formelles .....                 | 43        |
| 2.5.2 Classification des méthodes formelles .....                    | 44        |
| 2.6 Techniques de vérification formelle .....                        | 45        |
| 2.6.1 Vérification de modèles (Model Checking) .....                 | 45        |
| 2.6.2 Preuve de théorèmes (Theorem proving) .....                    | 45        |
| 2.7 Réseaux de Petri (RdPs) .....                                    | 46        |
| 2.7.1 Systèmes à événements discrets .....                           | 47        |
| 2.7.2 Concepts de base des réseaux de Petri ordinaires .....         | 48        |
| 2.7.2.1 Définitions informelles .....                                | 48        |
| 2.7.2.2 Définitions formelles .....                                  | 50        |
| 2.7.3 Exécution d'un réseau de Petri : Notion de marquage .....      | 51        |
| 2.7.4 Propriétés des Réseaux de Petri Places/Transitions .....       | 53        |
| 2.7.5 Méthodes d'analyse des réseaux de Petri .....                  | 57        |
| 2.7.6 Outils d'analyse des RdPs.....                                 | 59        |
| 2.8 Extensions des réseaux de Petri .....                            | 61        |

|  |           |
|--|-----------|
| 2.9 Présentation des ECATNets .....  | 62        |
| 2.9.1 Formes des Règles de Réécriture .....  | 63        |
| 2.9.2 ECATNets et Maude .....  | 65        |
| 2.9.3 Méthodes et outils d'analyse des ECATNets .....  | 67        |
| 2.10 Conclusion .....  | 67        |
| <br>   |           |
| <b>3. Transformations des modèles à l'aide de transformation de graphes</b> .....  | <b>69</b> |
| 3.1 Introduction .....   | 69        |
| 3.2. Architecture. dirigée par les modèles .....   | 69        |
| 3.2.1 Notion de modèle .....   | 69        |
| 3.2.2. La transformation des modèles .....   | 71        |
| 3.2.3. Méta-modélisation et transformation .....   | 72        |
| 3.2.4 Classification des approches de transformation .....   | 75        |
| 3.2.4.1 Transformations de type Modèle vers code.....  | 75        |
| 3.2.4.2 Transformations de type modèle vers modèle .....   | 76        |
| 3.3 Les transformations de graphes .....   | 79        |
| 3.3.1 Notion de graphe .....   | 79        |
| 3.3.1.1 Graphes non orientés .....   | 79        |
| 3.3.1.2 Graphes orientés .....   | 80        |
| 3.3.2. Grammaires de graphes .....   | 80        |
| 3.3.2.1 Le principe de règles .....  | 82        |
| 3.3.3.2. Application des règles.....   | 82        |
| 3.3.2.3 Système de transformation de graphes .....   | 83        |
| 3.3.3. outils de transformation de graphes .....   | 83        |
| 3.3.4. ATOM3 : présentation générale .....   | 83        |
| 3.4 Conclusion .....   | 84        |
| <br>   |           |
| <b>4 Une approche automatique de transformation des processus métiers vers les réseaux de Petri basée sur la transformation de graphes</b> ..... | <b>85</b> |
| 4.1. Introduction .....  | 85        |
| 4.2 Modèle de processus métiers .....  | 85        |
| 4. 3. Réseaux de Petri .....   | 86        |
| 4.4. Formalisation des processus métiers par les réseaux de Petri .....  | 86        |

|  |     |
|--|-----|
| 4.5 Transformation des processus métier vers les réseaux de Petri .....                                      | 87  |
| 4.5.1 Méta-Modélisation des Processus Métiers et des réseaux de Petri .....                                  | 87  |
| 4.5.1.1. Méta-Modélisation des Processus Métiers .....   | 88  |
| 4.5.1.2. Méta-Modélisation des réseaux de Petri .....  | 88  |
| 4.5.2. Génération de l'outil des Processus métiers .....   | 89  |
| 4.5.3. Grammaire de graphes pour la transformation des processus<br>métiers vers les réseaux de Petri .....  | 90  |
| 4.5.4 Exemple .....  | 95  |
| 4.6. Utilisation de l'analyseur INA .....  | 96  |
| 4.6.1 Méta -modélisation des RdPs .....  | 96  |
| 4.6.2 Génération de la spécification INA .....   | 97  |
| 4.6.3. Exemple .....   | 99  |
| 4.7. Etude de cas : Détection de la situation d'interblocage dans un réseau de Petri...                      | 101 |
| 4.8. Conclusion.....   | 104 |
| <br>   |     |
| <b>5 Une approche de transformation des modèles de processus métiers vers<br/>les réseaux de Petri</b> ..... | 105 |
| 5.1 Introduction .....   | 105 |
| 5.2 Les processus métiers dans l'EV .....  | 105 |
| 5.3 Modélisation des processus de L'EV .....   | 106 |
| 5.4 Description de l'approche .....  | 107 |
| 5.4.1 Modèle des cas d'utilisation .....   | 107 |
| 5.4.2 Modèle d'analyse .....   | 108 |
| 5.4.3 Modèle de conception .....   | 108 |
| 5.4.4 Transformation des diagrammes UML vers les ECATNets .....  | 109 |
| 5.4.5 Modèle d'Implémentation .....  | 109 |
| 5.4.6 Modèle de test .....   | 109 |
| 5.5 Techniques formelles appliquées à UML .....  | 111 |
| 5.5.1 Génération de tests .....  | 111 |
| 5.5.2 Approches traductionnelles de la formalisation d'UML .....   | 111 |
| 5.5.3 Les approches "méta" .....   | 112 |

|  |     |
|--|-----|
| 5.5.4 Utilisation de Réseaux de Petri.....   | 112 |
| 5.5.5 UML et la transformation de graphes .....  | 113 |
| 5.5.6 Diagrammes de séquence et réseaux de Petri .....   | 114 |
| 5.5.7 Discussion .....   | 115 |
| 5.6. Transformation des diagrammes de séquence vers les ECATNets .....                                   | 115 |
| 5.6.1. Méta-modélisation des diagrammes de séquences et des ECATNets..                                   | 116 |
| 5.6.1.1. Méta-modélisation des diagrammes de séquences.....  | 116 |
| 5.6.1.2. Méta-modèle des ECATNets .....  | 118 |
| 5.6.2. Grammaire de graphes pour la transformation des diagrammes<br>de séquences vers les ECATNets..... | 119 |
| 5.7. Analyse des diagrammes de séquences .....   | 120 |
| 5.8. Exemples de transformation d'un diagramme de séquence vers un<br>ECATNets.....                      | 120 |
| 5.8.1.Exemple 1 .....  | 120 |
| 5.8.2.Exemple 2.....   | 123 |
| 5.9. Conclusion .....  | 124 |
| <b>Conclusion générale</b>   | 126 |
| <b>Bibliographie</b>   | 128 |

## Liste des figures

|   |    |
|---|----|
| <b>Figure 1.1</b> : Typologie des processus .....   | 8  |
| <b>Figure 1.2</b> : (a) Processus de gestion des commandes (b) Exemples de processus .....                        | 11 |
| <b>Figure 1.3</b> : (a) l'entreprise vue par ses processus (b) Entreprise vue par ses processus .....             | 12 |
| <b>Figure 1.4</b> : Modélisation des entreprises .....  | 13 |
| <b>Figure 1.5</b> : Exemple d'organigrammes .....   | 16 |
| <b>Figure 1.6</b> : Symboles de base des DFDs .....   | 17 |
| <b>Figure 1.7</b> : Traitement des commandes de la clientèle représenté par un DFD .....                          | 17 |
| <b>Figure 1.8</b> : Le modèle SADT/IDEF0 .....  | 19 |
| <b>Figure 1.9</b> : La grille de GRAI .....   | 21 |
| <b>Figure 1.10</b> : Réseau GRAI, activité de décision et activité d'exécution .....                              | 22 |
| <b>Figure 1.11</b> : Le cadre de modélisation de CIMOSA .....   | 23 |
| <b>Figure 1.12</b> : Chaîne de processus de pilote d'événement. ....  | 27 |
| <b>Figure 1.13</b> : Symbolisation BPMN .....   | 29 |
| <b>Figure 1.14</b> : Un segment d'un processus .....  | 30 |
| <b>Figure 1.15</b> : Evolution de UML .....   | 31 |
| <b>Figure 1.16</b> : Diagrammes UML .....   | 33 |
| <b>Figure 2.1</b> : Méthode générale de modélisation et d'analyse basée sur les réseaux de Petri .....            | 46 |
| <b>Figure 2.2</b> : Exemple de réseau de Petri marqué .....   | 49 |
| <b>Figure 2.3</b> : (a) RdP marqué avant franchissement de T1 (b) RdP après franchissement de T1 .....            | 52 |
| <b>Figure 2.4</b> : Matrice d'incidence et vecteur de marquage du RdP de la figure 2.2 .....                      | 53 |
| <b>Figure 2.5</b> : RdP borné .....   | 54 |
| <b>Figure 2.6</b> : RdP quasi-vivant, pseudo-vivant .....   | 55 |
| <b>Figure 2.7</b> : RdP vivant .....  | 56 |
| <b>Figure 2.8</b> : RdP réversible .....  | 57 |
| <b>Figure 2.9</b> : (a) Réseau de Petri (b) Graphe des marquages fini .....                                       | 59 |
| <b>Figure 2.10</b> : (a) Réseau de Petri (b) Graphe des marquages infini .....                                    | 59 |
| <b>Figure 2.11</b> : Trois programmeurs partageant deux terminaux .....   | 60 |
| <b>Figure 2.12</b> : La spécification INA de l'exemple de la figure 2.11 .....                                    | 61 |
| <b>Figure 2.13</b> : Un ECATNets générique .....  | 62 |
| <b>Figure 2.14</b> : ECATNets modélisant le problème du routeur .....   | 65 |
| <b>Figure 2.15</b> : ECATNets du problème du routeur dans l'outil de transformation des ECATNets vers Maude ..... | 66 |
| <b>Figure 2.16</b> : Spécification Maude de l'exemple du routeur .....  | 67 |
| <b>Figure 3.1</b> : Aperçu globale du processus de transformations de modèles de l'approche MDA .....             | 71 |
| <b>Figure 3.2</b> : Niveaux d'abstraction de la méta-modélisation. ....   | 72 |
| <b>Figure 3.3</b> : (a) Modèle AFD des nombre binaires pairs (b) Méta-modèle : Formalisme AFD                     |    |



|  |     |
|--|-----|
| (Diagramme de classe UML + OCL) .....  | 73  |
| <b>Figure 3.4</b> : Concepts de base de la transformation de modèles. ....   | 74  |
| <b>Figure 3.5</b> : Graphe non orienté .....   | 79  |
| <b>Figure 3.6</b> : (a) graphe G (b) sous-graphe de G .....  | 79  |
| <b>Figure 3.7</b> : Graphe orienté .....   | 80  |
| <b>Figure 3.8</b> : Graphe orienté étiqueté .....  | 80  |
| <b>Figure 3.9</b> : (a) Partie gauche d'une règle (b) Partie droite d'une règle .....  | 81  |
| <b>Figure 4.1</b> : Un exemple de processus métier. ....   | 86  |
| <b>Figure 4.2</b> : Un réseau de Petri représentant le processus métier de la Figure 4.1   | 86  |
| <b>Figure 4.3</b> Le Meta-Model des processus métiers .....  | 88  |
| <b>Figure 4.4</b> : Méta- Model des réseaux de Petri .....   | 89  |
| <b>Figure 4.5</b> : Outil généré pour les processus métiers .....  | 89  |
| <b>Figure 4.6</b> : (a) LHS de la règle 1 (b) RHS de la règle 1 .....  | 90  |
| <b>Figure 4.7</b> : (a) LHS de la règle 2 (b) RHS de la règle 2 .....  | 90  |
| <b>Figure 4.8</b> : (a) LHS de la règle 3 (b) RHS de la règle 3 .....  | 91  |
| <b>Figure 4.9</b> : (a) LHS de la règle 4 (b) RHS de la règle 4 .....  | 91  |
| <b>Figure 4.10</b> : (a) LHS de la règle 5 (b) RHS de la règle 5 .....   | 92  |
| <b>Figure 4.11</b> : (a) LHS de la règle 6 (b) RHS de la règle 6 .....   | 92  |
| <b>Figure 4.12</b> : (a) LHS de la règle 7 (b) RHS de la règle 7 .....   | 93  |
| <b>Figure 4.13</b> : (a) LHS de la règle 8 (b) RHS de la règle 8 .....   | 93  |
| <b>Figure 4.14</b> : (a) LHS de la règle 9 (b) RHS de la règle 9 .....   | 94  |
| <b>Figure 4.15</b> : (a) LHS de la règle 10 (b) RHS de la règle 10 .....   | 94  |
| <b>Figure 4.16</b> : (a) LHS de la règle 11 (b) RHS de la règle 11 .....   | 94  |
| <b>Figure 4.17</b> : Exemple de processus métier .....   | 95  |
| <b>Figure 4.18</b> : Le réseau de Petri généré du processus métier de la figure 4.17 .....   | 95  |
| <b>Figure 4.19</b> : Environnement graphique généré pour les modèles RdPs .....  | 97  |
| <b>Figure 4.20</b> : grammaire de graphes pour la génération la spécification INA à partir d'une représentation<br>graphique. .... | 99  |
| <b>Figure 4.21</b> : représentation graphique de l'exemple crée par notre outil. ....  | 100 |
| <b>Figure 4.22</b> : Spécification INA de l'exemple de la figure 4.21. ....  | 100 |
| <b>Figure 4.23</b> : Spécification INA du modèle RdP de la figure 4.2 .....  | 101 |
| <b>Figure 4.24</b> : Résultat de l'application de l'outil INA sur le fichier de la figure 4.21 .....                               | 103 |
| <b>Figure 5.1</b> : Approche de modélisation des processus métiers dans les EVs. ....  | 110 |
| <b>Figure 5.2</b> : Méta-modèle des diagrammes de séquences. ....  | 116 |
| <b>Figure 5.3</b> : Outil de modélisation des diagrammes de séquences. ....  | 117 |
| <b>Figure 5.4</b> : Méta-modèle des ECATNets. ....   | 118 |
| <b>Figure 5.5</b> : Outil de modélisation des ECATNets .....   | 119 |
| <b>Figure 5.6</b> : Exemple d'un diagramme de séquences. ....  | 120 |
| <b>Figure 5.7</b> : Lancement de l'exécution de la grammaire. ....   | 121 |
| <b>Figure 5.8</b> : graphe intermédiaire obtenu après l'exécution de la règle 1. ....  | 121 |

|  |     |
|--|-----|
| <b>Figure 5.9 :</b> ECATNets résultat de la transformation. ....   | 122 |
| <b>Figure 5.10 :</b> outil de génération des spécifications Maude à partir des ECATNets .....                          | 122 |
| <b>Figure 5.11:</b> Spécification Maude équivalente à l' ECATNets présenté dans la figure 5.9 .....                    | 123 |
| <b>Figure 5.12 :</b> diagramme de séquences. ....  | 124 |
| <b>Figure 5.13:</b> Modèle ECATNets résultat de la transformation du diagramme de séquences<br>de la figure 5.12. .... | 124 |

## **Introduction générale**

Les entreprises actuelles connaissent des changements multiples aussi bien dans les formes d'organisation que dans leurs façons de concevoir et de produire. Ceci est dû aux nouveaux critères de compétitivité imposés par un marché en évolution continue conduisant à une concurrence de plus en plus sauvage.

La mondialisation du marché, les cycles de vie des produits de plus en plus courts, le besoin accru de flexibilité et des changements fréquents de techniques et de technologie ont obligé les entreprises à se préoccuper plus sérieusement de leurs modélisation afin de faire face à tous ces facteurs complexes.

La modélisation des entreprises est une tâche complexe qui concerne la représentation et la spécification des différents aspects des opérations des entreprises; fonctionnels informationnels, ressources et organisationnels.

Durant les dernières années, la notion de "processus" s'est largement installée car on s'est rendu compte que pour être conforme avec la nouvelle philosophie des affaires, l'organisation doit être conçue pour fournir le flot vertical et horizontal des informations nécessaires à la réalisation des objectifs globaux de l'organisation.

L'approche processus consiste à décrire de façon méthodique une organisation ou un ensemble d'activités en processus, de façon à organiser sa contribution à la satisfaction des clients.

Il existe plusieurs types de processus (de pilotage, métier et de support). Ce qui nous intéresse dans le cadre de cette thèse ce sont les processus métiers qui contribuent directement à la réalisation du produit, de la détection du besoin client jusqu'à sa satisfaction.

Donc un intérêt particulier doit être porté à la modélisation de ces processus métiers. La modélisation de processus a pour but la production d'abstraction du processus qui servira de base pour une définition détaillée, une étude et un réengineering possible afin d'éliminer les activités n'ayant pas de valeur ajoutée. La modélisation de processus doit permettre une compréhension claire et transparente des activités considérées, les dépendances entre les activités et les rôles (gens, machines, informations,...) nécessaires pour le processus.

La capacité de représenter les comportements tels que la concurrence et le choix, augmente les chances de définir des modèles logiquement incorrects avec des erreurs de contrôle de flux (interblocage, vivacité...).

Ce risque d'erreur est d'autant plus grand lorsqu'il s'agit de systèmes complexes telles que les entreprises virtuelles.

En effet, dû aux menaces du marché contemporain, de nouvelles formes d'organisations ont vu le jour, telles que les entreprises étendues, réseaux et virtuelles. Ces dernières présentent un paradigme prometteur pour l'entreprise moderne. Leur caractéristique essentielle est l'exécution distribuée et parallèle des processus métiers par les entités de chaque membre de l'EV. D'où le besoin d'une modélisation efficace des processus métiers avec un modèle strict d'analyse.

Plusieurs modèles de processus métiers sont proposés dans la littérature. Or pour la plupart, il y a un manque apparent de moyens d'analyse et de vérification.

Des techniques diverses d'analyse et de vérification existent. Parmi lesquelles on peut citer les réseaux de Petri. Ces derniers ont été choisis dans notre thèse pour la vérification de nos modèles. Ceci se justifie par leur base mathématique solide, la large gamme d'outils d'analyse qu'ils possèdent, par leur simplicité et leur capacité de représentation graphique des modèles.

Du fait que les modèles des processus métiers sont des graphes et que les réseaux de Petri le sont aussi, Il semblait très naturel d'utiliser la transformation de graphes pour passer d'un modèle à un autre.

Dans ce contexte nous proposons une approche totalement automatisée basée sur la méta-modélisation et les grammaires de graphes pour générer de manière totalement automatique des modèles réseaux de Petri pour les processus métiers en utilisant AToM<sup>3</sup>.

Pour analyser les RdPs obtenus, on a choisi l'outil d'analyse INA. Mais comme il utilise une spécification textuelle, une autre partie de notre travail est consacrée à la génération automatique d'un environnement graphique pour cet outil.

La deuxième contribution dans cette thèse consiste en la proposition d'une approche intégrée UML-ECATNets pour la modélisation des processus métiers dans les entreprises virtuelles.

L'approche proposée suit le processus UP et utilise les diagrammes UML pour la modélisation des processus métiers dans les entreprises virtuelles. Pour l'analyse des diagrammes obtenus, nous recourons aux ECATNets, une catégorie des réseaux de Petri algébriques de haut niveau. La puissance des ECATNets réside dans leur sémantique basée sur la logique de réécriture et son langage de spécification Maude. Du moment qu'un ECATNets peut être transformé en une spécification Maude, le langage Maude peut être alors

utilisé comme moyen pour l'analyse des systèmes modélisés par des ECATNets. En plus de l'utilisation de Maude, beaucoup d'autres outils d'analyse des ECATNets ont été proposés.

Plus précisément, nous proposons une transformation automatique des *diagrammes de séquence* vers les *ECATNets* basée sur les grammaires de graphes et réalisée à l'aide de l'outil AToM3. Ensuite Le langage Maude est utilisé pour analyser les modèles obtenus.

## Organisation du mémoire

Le reste de ce mémoire est organisé comme suit.

*Le chapitre 1* donne un aperçu sur la modélisation des entreprises et leur évolution. Nous mettrons l'accent sur la notion de processus, l'évolution de l'approche processus dans les organisations, ainsi que les types de processus. Nous présentons ensuite les processus métiers et leurs différents modèles. Nous commencerons par rappeler la modélisation d'un processus, les objectifs de modélisation, les critères de modélisation, et nous terminerons ce chapitre par les différentes techniques de modélisation des processus métiers.

*Le chapitre 2* sera consacré aux méthodes formelles pour la vérification des systèmes. Un rappel sur les différents concepts des réseaux de Petri et sur les ECATNets sera donné. Les ECATNets sont une catégorie de réseaux de Petri de haut niveau dont la sémantique est donnée dans la logique de réécriture.

*Le chapitre 3* sera dédié à la transformation de modèles à l'aide de transformation de graphes. Nous donnerons un rappel sur L'architecture dirigée par les modèles ou MDA ("Model Driven Architecture") qui est une démarche de réalisation de logiciel, proposée et soutenue par l'OMG. MDA est une variante particulière de l'ingénierie dirigée par les modèles. Puis la transformation de graphes comme outil de transformation de modèles sera présentée.

Nous proposerons dans *le chapitre 4* notre première contribution qui consiste en une approche totalement automatique basée sur la transformation de graphe pour transformer les modèles des processus métiers vers les réseaux de Petri sous forme de spécification de l'outil INA (Integrated Net Analyzer). Cette approche est réalisée en deux étapes. Dans la première étape, nous proposerons deux méta-modèles (un pour les modèles processus métiers et un pour les graphes des réseaux de Petri) et une grammaire de graphe permettant de transformer un modèle d'un processus métier vers son réseau de Petri équivalent sous forme d'un graphe. Dans la deuxième étape nous proposerons une grammaire qui nous permettra de générer automatiquement la spécification INA (forme textuelle) du réseau de Petri (sous forme graphique) obtenu par la première étape. L'objectif de cette

transformation est d'utiliser l'outil INA pour la vérification des propriétés telles que l'interblocage, la répétition multiple, etc ...

*Le chapitre 5* constituera notre deuxième contribution à savoir la proposition d'une approche intégrée UML/ECATNets pour la modélisation et la vérification des processus métiers dans les entreprises virtuelles. Notre approche proposée suit le processus de développement UP. UML est utilisé pour la modélisation tandis que les ECATNets sont utilisés pour l'analyse et la vérification des diagrammes obtenus. Plus précisément, nous avons proposé une transformation automatique des *diagrammes de séquence* vers la catégorie des réseaux de Petri appelée *ECATNets* basée sur les grammaires de graphes et réalisée à l'aide de l'outil AToM3. Ensuite, le langage Maude est utilisé pour analyser les modèles obtenus.

*Une conclusion générale et des perspectives* de ce travail de recherche termineront cette thèse.

# Chapitre 1

## Modélisation des entreprises

### 1.1 Introduction

De nos jours, les entreprises opèrent dans un milieu en perpétuelle évolution. La mondialisation du marché et la concurrence farouche, les changements fréquents de techniques et de technologies, les clients de plus en plus puissants, les réglementations gouvernementales, et les responsabilités éthiques sont quelques exemples des pressions que subit l'entreprise actuelle.

Pour faire face à tous ces facteurs complexes, la modélisation des entreprises est devenue une préoccupation primordiale depuis le milieu du 20<sup>ième</sup> siècle.

De nouveaux paradigmes et de nouvelles formes d'organisations sont apparus pour essayer de répondre aux exigences des entreprises imposées par les changements continus de l'environnement.

L'approche processus a connu, les dernières années, un intérêt remarquable dans la modélisation des entreprises ce qui a conduit à l'apparition d'un grand nombre de modèles et d'approches de modélisation.

L'organisation classique des entreprises cède la place aux nouvelles formes d'organisations telles que l'entreprise étendue et virtuelle pour pouvoir s'adapter à la révolution technologique actuelle.

Le but de ce chapitre est donc de donner un aperçu sur les entreprises, leur modélisation et leur évolution, ainsi que sur les processus métiers et leurs différents modèles.

### 1.2 Modélisation des entreprises

Une entreprise selon [Vernadat96] est une entité complexe composée de gens et de processus et fournissant des produits et des services à des clients.

Un modèle est une abstraction de la réalité. Cette représentation est construite, vérifiée, analysée et manipulée pour maîtriser la réalité et mieux la comprendre. Certains modèles sont mentaux, d'autres sont codifiés. " Tous les modèles sont faux, mais certains sont utiles" [box79].

[LeMoigne90] définit la modélisation comme *"l'élaboration et la construction intentionnelle par composition de symboles, de modèles susceptibles de rendre intelligible un phénomène perçu complexe, et d'amplifier le raisonnement de l'acteur projetant une intention délibérée au sein du phénomène"*. Un modèle possède alors une syntaxe définie et chaque élément de modèle véhicule une sémantique particulière.

[Zaidat05] ajoute que la modélisation est basée sur des techniques appropriées : *"le processus de modélisation dans les entreprises est une tâche très complexe. Il consiste à décrire un agencement d'un nombre d'éléments important dont la nature est différente. La réalisation de ce processus requiert l'utilisation de techniques de modélisation appropriées. C'est dans ce sens que les architectures de référence ont proposé des cadres de modélisation pour la conduite du processus de modélisation"*.

"La modélisation des entreprises concerne la représentation et la spécification des différents aspects des opérations des entreprises. L'aspect *fonctionnel* décrit ce qu'on doit faire et dans quel ordre. L'aspect *informationnel* décrit quels sont les objets utilisés ou traités. L'aspect *ressource* décrit qui "fait" les choses et selon quelle politique. Et enfin, l'aspect *organisationnel* décrit la structure organisationnelle dans laquelle les choses seront-elles faites" [Verdanat66].

### **1.3. Evolution de l'approche processus dans les organisations**

Durant les dernières cinquante années, la vision processus a joué un rôle grandissant dans les théories des organisations, comme dans le domaine système d'information où la modélisation des processus est reconnue comme un élément clé de la représentation de la dynamique [Curtis92]. Les deux aspects sont fortement liés : un processus système d'information peut être considéré comme une vue d'un processus métier centrée sur les informations [Alter99], et inversement, la modélisation des processus métiers est considérée comme un préalable nécessaire à la conception d'un système d'information organisationnel [Butler99]. La littérature abondante sur la gestion des processus métiers ([Davenport93], [Hammer93], [McCormack01], [Burlton01], [Harmon03]) suggère que les organisations peuvent nettement améliorer leurs performances en adoptant une vue orientée processus des affaires.

En effet, face à la mondialisation et aux menaces ou aux opportunités environnementales, on s'est rendu compte que le **"business process"** est le paradigme le plus important dans la gestion. L'idée de l'organisation processus devient plus forte. L'option "processus" est devenue



un besoin obligatoire [Levi02] ce qui veut dire que pour être conforme avec la nouvelle philosophie des affaires, l'organisation doit être conçue pour fournir le flot vertical et horizontal des informations nécessaires à la réalisation des objectifs globaux de l'organisation. [Daft04].

L'approche processus consiste à décrire de façon méthodique une organisation ou un ensemble d'activités en processus, de façon à organiser sa contribution à la satisfaction des clients. Ainsi la production d'un produit est un processus qui fait intervenir différentes entités d'une organisation. L'importance de l'approche qualité et de la satisfaction du client ont mené à cette définition de processus "orientés client" par nature inter-fonctionnelle, donc transversale à l'organisation et soutenus par la mise en place des technologies de l'information et de la communication [Gaubert-Macon06].

#### 1.4. Types de processus

Quand on parle de processus, il faut bien préciser de quel "type" il s'agit. La norme AFNOR de juin 2000 sur le management des processus [Catton00] propose trois grandes familles de processus :

- *les processus métiers* (de réalisation ou opérationnels) : ils contribuent directement à la réalisation du produit, depuis la détection du besoin client jusqu'à sa satisfaction. Ils regroupent des activités liées au cycle de vie du produit : recherche de nouveaux produits, conception, achats et approvisionnements, logistique, production, commercialisation, etc.
- *les processus de support* (dits aussi processus de soutien) : ils contribuent au bon déroulement des processus de réalisation en leur apportant les ressources nécessaires. Bien que ne créant pas de valeur directement perceptible par le client, ils sont nécessaires au fonctionnement permanent de l'organisation et de sa pérennité. Selon l'activité de l'organisation et sa stratégie, les processus de support peuvent être considérés comme des processus de réalisation et réciproquement. C'est le cas, par exemple, de la gestion des ressources humaines, des achats/approvisionnements, de la logistique, etc.

Dans [Brandenburg05], il est mentionné que l'on retrouve toujours trois types génériques de processus de support qui fournissent ou surveillent trois types de moyens : moyens humains, matériels et financiers ; soit les "M" des anglo-saxons (*Men, Means and Money*).

- **les processus de direction** (dits aussi processus de management) : ils contribuent à la détermination de la politique et au déploiement des objectifs dans l'organisation. Sous la responsabilité totale de l'équipe dirigeante, ils permettent d'orienter et d'assurer la cohérence des processus de réalisation et de support. Parmi les processus de direction on peut citer :
  - L'élaboration de la stratégie de l'organisation,
  - Le management de la qualité de l'organisation, et
  - La communication interne et mobilisation du personnel.

Le schéma suivant donne une vision globale des types de processus et de leurs interactions :

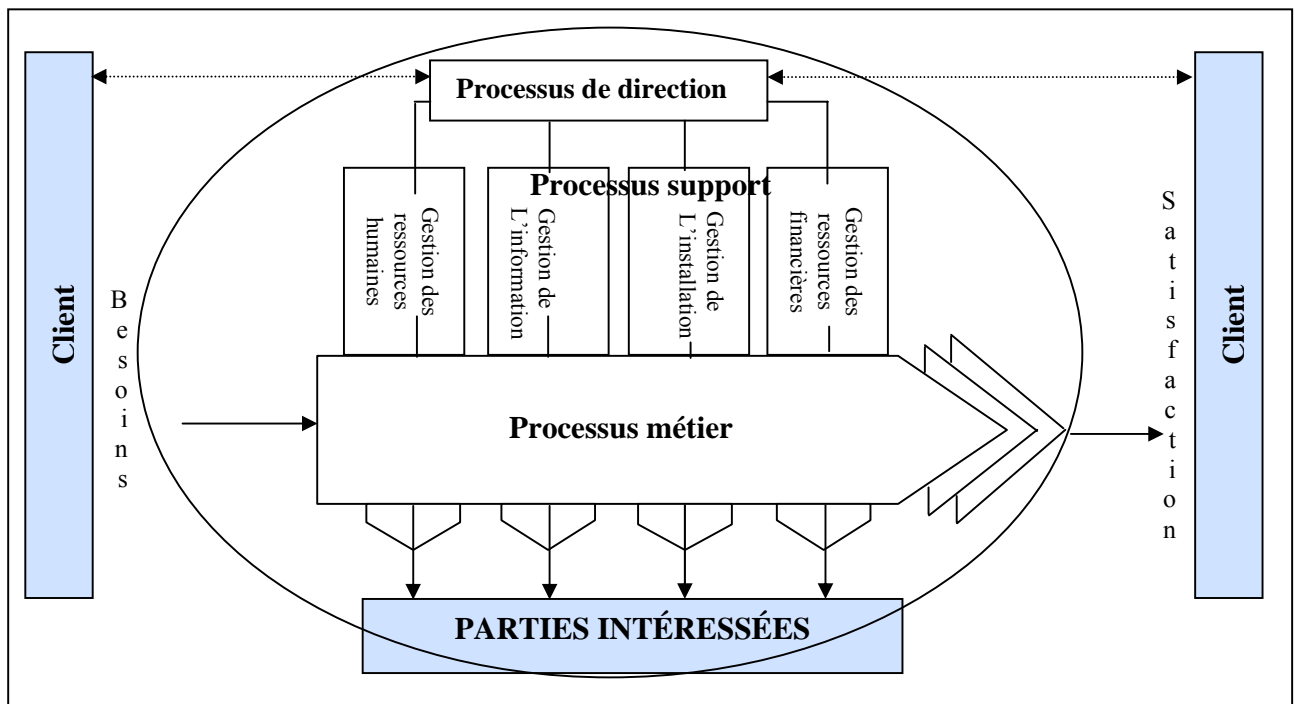


Figure 1.1 : Typologie des processus.

### 1.5. Définition de processus

- Une définition utile dit qu' : *"Un processus est un ensemble partiellement ordonné d'étapes exécutées en vue de réaliser au moins un objectif"* [Vernadat99].
- La définition officielle de la norme ISO 9001 (2000) [ISO00] voit le processus comme : *"Un ensemble d'activités corrélées ou interactives qui transforment des éléments d'entrée en éléments de sortie"*.
- Dans [Morley02] on s'intéresse plus à la structure d'un processus : *"Un processus représente l'organisation d'un ensemble finalisé d'activités effectuées par des acteurs et mettant en jeu des entités"*.
- Une définition plus complète provient de [Théroude02], *"Un processus est défini comme un enchaînement partiellement ordonné d'exécution d'activités qui, à l'aide de **moyens techniques et humains**, transforme des éléments d'entrée en éléments de sortie en vue de réaliser un **objectif** dans le cadre d'une **stratégie** donnée"*.

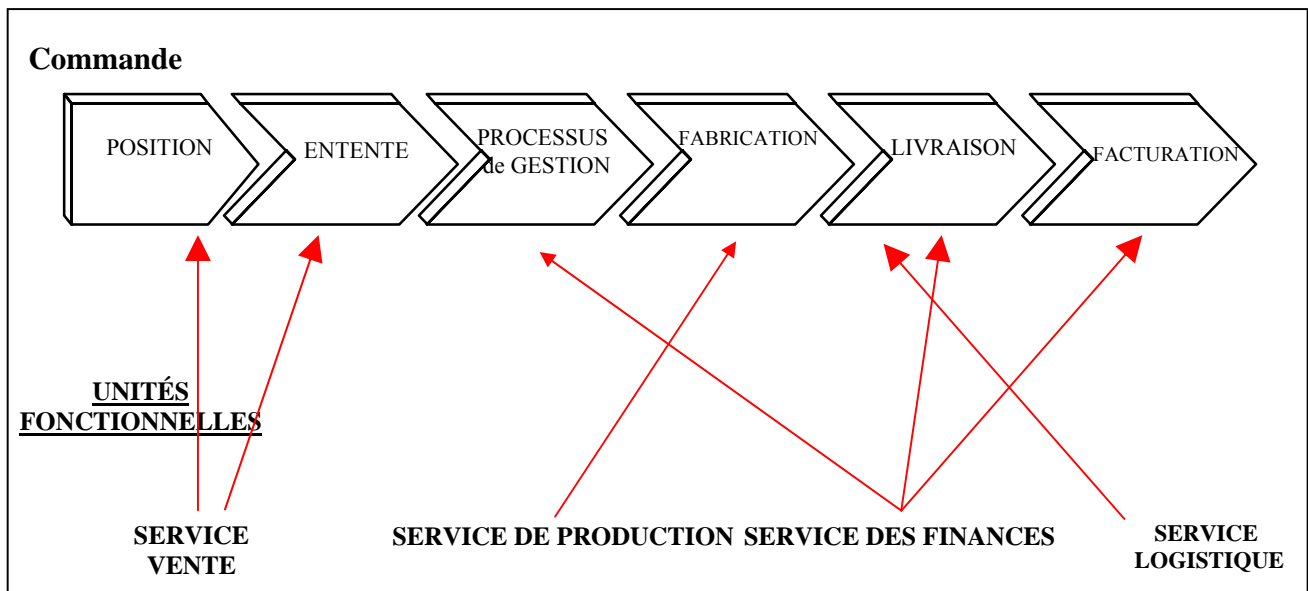
Donc on peut dire qu'un processus métier est un enchaînement d'activités qui prend un input (de n'importe quelle forme) lui rajoute de la valeur à l'aide de ressources et fournit un output (produit /service) répondant aux objectifs de l'entreprise. Il est déclenché par des événements internes ou externes de l'entreprise. Il peut être décomposé en sous- processus et communiquer avec d'autres processus.

Un processus possède donc les caractéristiques suivantes :

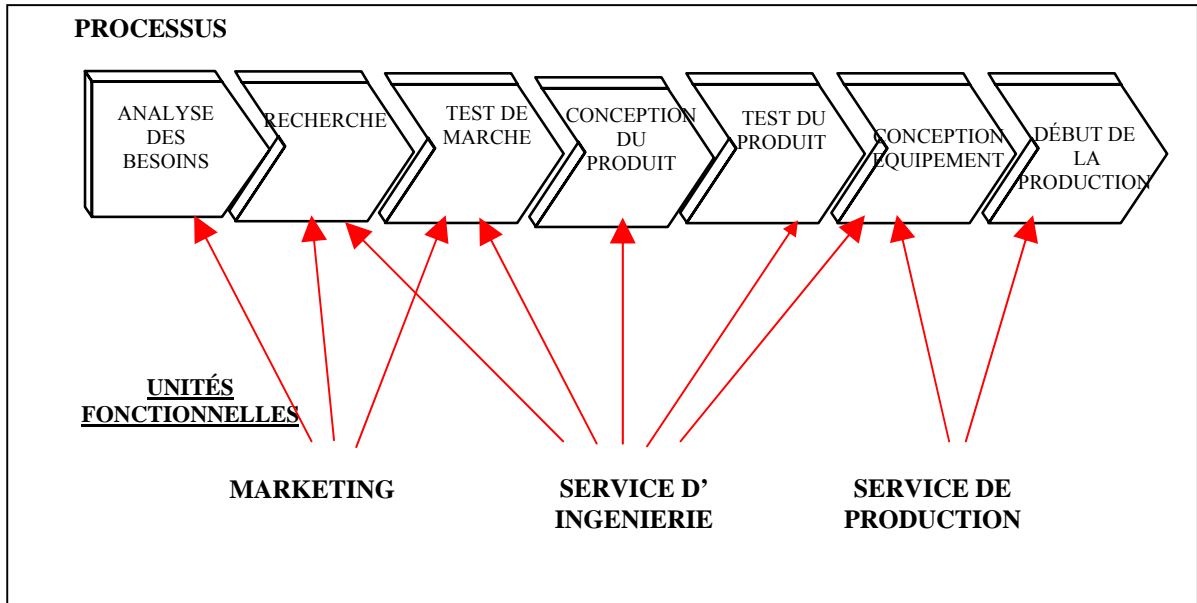
- Représente une vue dynamique de l'organisation (Possède un **but**).
- Possède une **entrée** et une **sortie**.
- Est composé de sous-processus, puis **d'activités** qui sont les éléments d'action atomiques. Une activité exprime la transformation d'une ressource d'entrée en une ressource de sortie.
- Un **graphe d'activités**, qui représente l'enchaînement des activités nécessaires à la réalisation de l'objectif.
- Ajoute de la valeur aux biens ou aux services;
- Des **rôles**, qui expriment l'organisation dans le processus.
- Une **fonction de transition**, qui contrôle le déroulement du processus.
- Des **ressources**, qui peuvent être des moyens, des informations ou des outils utilisés par une activité.

- Peut impliquer plusieurs unités fonctionnelles.
- S'exécute généralement horizontalement à travers une organisation verticale.

Un processus métier est un processus mis en œuvre au sein d'une organisation et dont les sorties répondent aux exigences d'un client externe ou interne de l'entreprise. La figure 1.2 montre des exemples de processus Métiers :



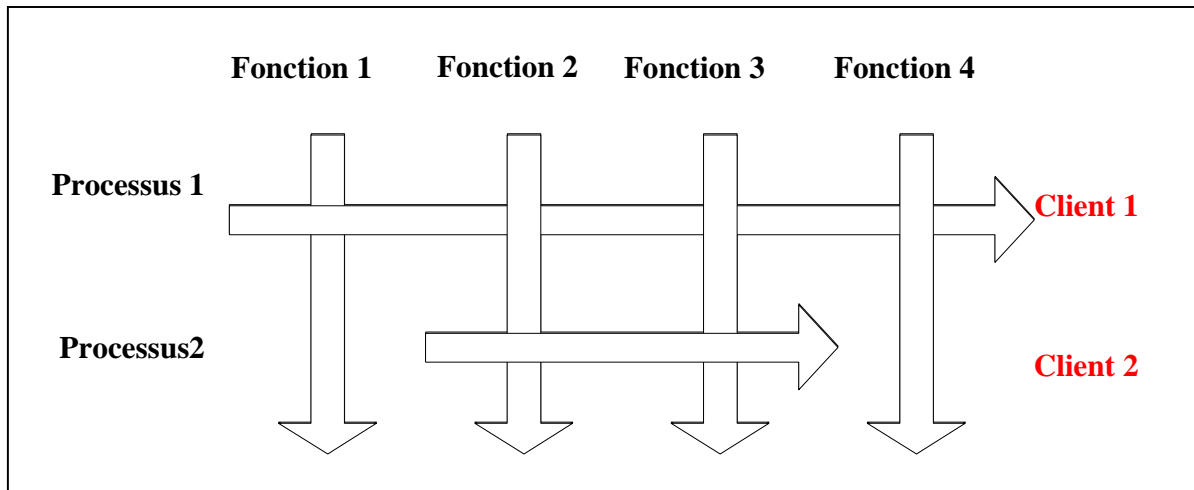
(a) : Processus de gestion de commandes



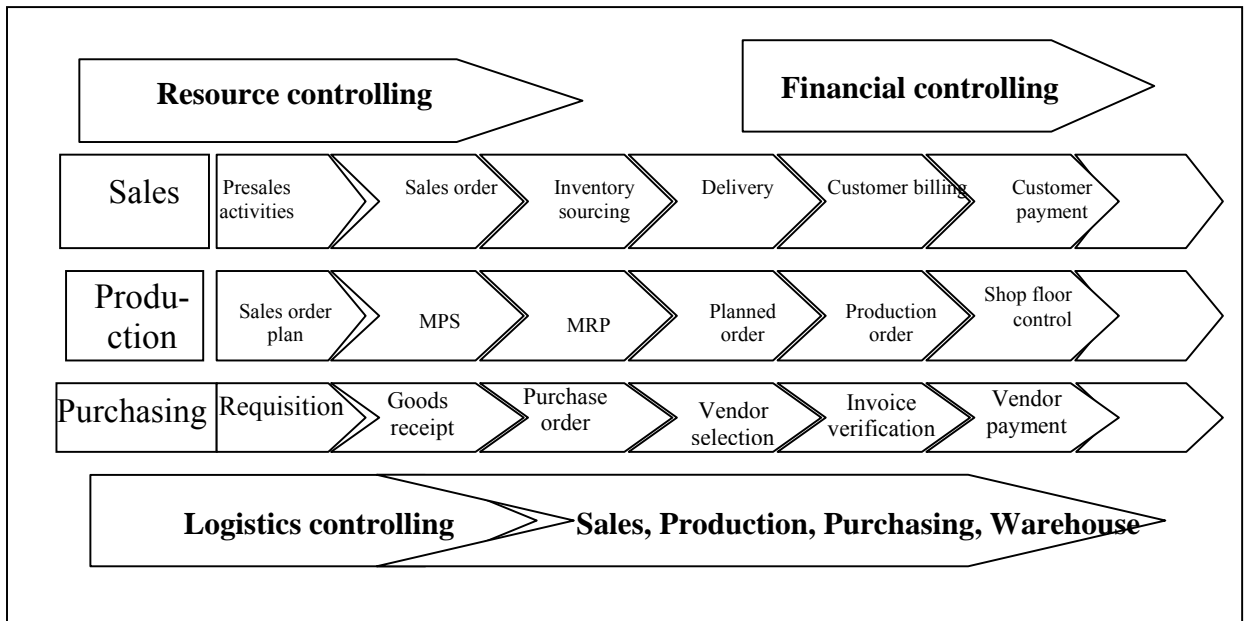
(b) : Processus de développement d'un nouveau produit

Figure 1.2 : (a) et (b) Exemples de processus

Les processus d'une organisation s'étendent de façon fonctionnelle sur plusieurs services ou départements. Ils sont interdépartementaux et représentent la *carte fonctionnelle* de l'entreprise :



(a)



(b)

Figure 1.3 : (a) l'entreprise vue par ses processus (b) Entreprise vue par ses processus [Talbot03]

### 1.6.1 Modélisation des processus

Dans le contexte de la modélisation des entreprises, la modélisation des processus métiers joue un rôle fondamental. L'aspect fonctionnel est au cœur des autres vues des entreprises. Elle consiste à représenter la structure et le fonctionnement de l'entreprise.

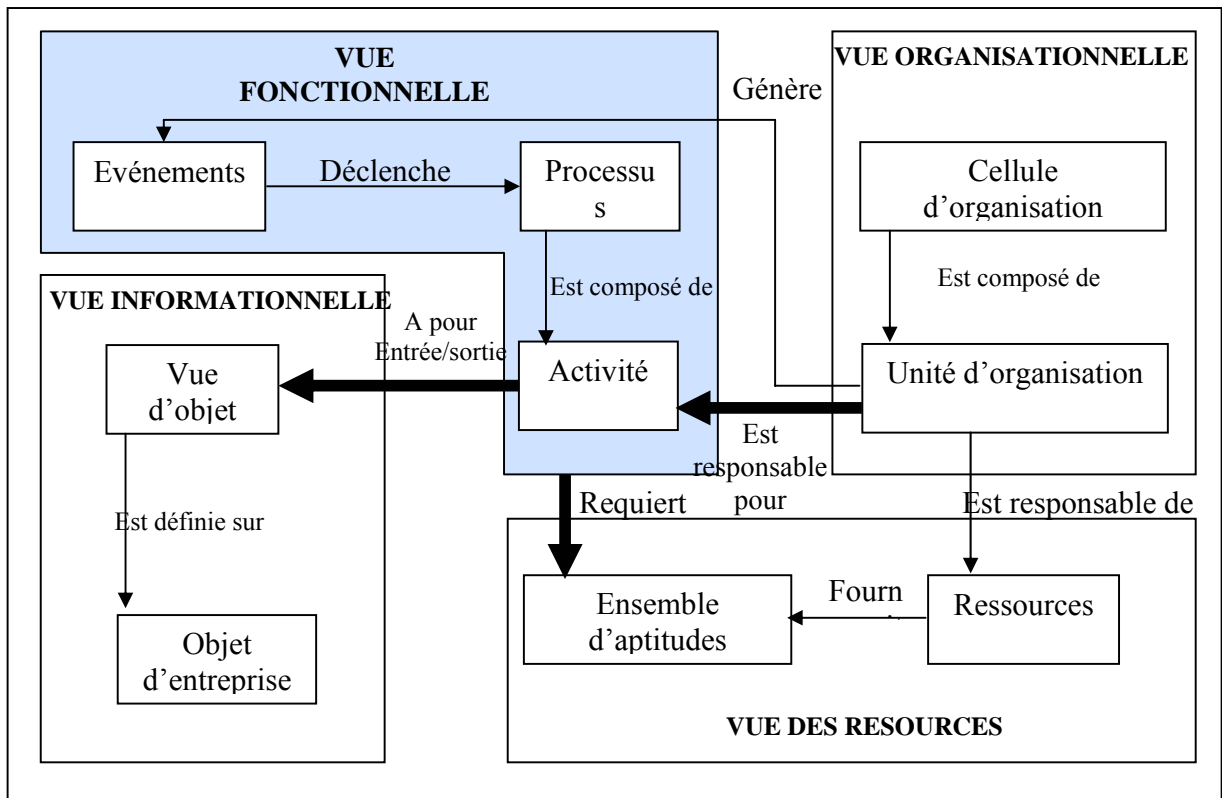


Figure 1.4 : Modélisation des entreprises

En effet, la vue fonctionnelle est la seule vue qui est connectée à toutes les autres vues (Figure 1.4). L'élément *activité* de cette vue assure cette connexion avec les autres vues : une *activité* a pour entrée / sortie des *objets* (vue informationnelle), il existe une *unité d'organisation* (vue organisationnelle) de l'entreprise, responsable de l'*activité* et l'*activité* requiert pour son exécution un *ensemble d'aptitudes* fournies par des *ressources* (vue des ressources) de l'entreprise. Si nous devons caractériser un modèle d'entreprise et nous poser la question de la vue sur laquelle s'appuyer pour commencer cette caractérisation, la réponse est certainement la vue fonctionnelle [Touzi07].

### 1.6 .2 Objectifs de la modélisation

La modélisation des processus a beaucoup d'avantages pour les entreprises qui cherchent à améliorer leurs performances. On peut citer :

- Faciliter la communication en utilisant un langage commun

- Meilleure compréhension de l'existant
- Documentation du processus métier
- Améliorer la situation actuelle
- Expérimenter et simuler de nouvelles situations et de nouveaux concepts et leurs impacts sur l'organisation
- Automatiser le processus, ...

Bien que la modélisation des processus métiers ait beaucoup d'avantages, elle l'est rarement faite. Et si c'est le cas, elle ne l'est pas bien faite. Ceci est dû à la nature complexe et floue des processus métiers qui les différencie des autres projets d'ingénieurs qui se font de manière bien structurée. L'architecture, qui est le haut niveau de la conception fonctionnelle des processus métiers est plus un art qu'une science [Dufresne03].

Cette complexité extrême des processus métiers est due à plusieurs raisons [Dufresne03] :

- Ils requièrent plusieurs domaines de connaissance.
- Ils opèrent dans des échelles de temps " largement différentes".
- Ils sont souvent indépendants.
- Les gens ont besoin d'années d'entraînement pour les comprendre ou comprendre "leur raison d'être"
- Ils ont beaucoup " de modification non- contrôlées"

### 1.6.3. Critères de modélisation

Stefan Harbal a développé un ensemble de critères pour évaluer les modèles des processus métiers [harbal] :

- Doivent être capables de modéliser toutes les complexités des PM, à savoir :
  - ❖ Séquencement
  - ❖ Choix
  - ❖ Boucle
  - ❖ Constructeurs de concurrences (fork et synchronize)
  - ❖ Timeouts
  - ❖ Deadlines
  - ❖ Fautes
  - ❖ Agrégation



- Doivent avoir un moyen de distinguer les rôles et de les affecter aux différentes tâches.
- Doivent avoir une représentation graphique non ambiguë du langage.
- Doivent avoir un model de transaction qui permet la description du comment "le processus peut être non accompli".
- Doivent spécifier comment les instances du processus vont être déclenchées et identifiées durant leur exécution.
- Doivent avoir un moyen de spécifier les caractéristiques du PM qui peuvent intéresser les utilisateurs externes, telles que la qualité de service et le prix.
- Le langage ne doit pas s'embrouiller dans les détails des protocoles de communication.

Les trois premiers critères sont importants pour obtenir les cinq premiers avantages cités plus haut. Les trois suivants sont essentiels pour automatiser l'exécution des processus entre les organisations séparées et qui collaborent entre elles. Le dernier critère garde le modèle dans le niveau d'abstraction adéquat.

## **1.7 Techniques de modélisation des processus métiers**

Durant les dernières années, la modélisation des entreprises a été un domaine de recherche très actif. En réalité les gens modélisaient les PM depuis plusieurs années, même s'ils n'appelaient pas leurs modèles des " modèles de processus métiers".

### **1.7.1 Organigrammes (Flowcharts) :**

Adoptés par la communauté des programmeurs depuis longtemps, ils représentent probablement les premières tentatives de modélisation de processus. Leur symbolisation et leur sémantique se limitent aux structures de contrôle atomique disponibles aux programmeurs. C'est une façon typique de modélisation des structures d'organisation. Ce modèle illustre un aspect de la vue organisationnelle de l'entreprise. Dans l'organigramme sont représentées, en fonction des critères de structuration sélectionnés, les unités organisationnelles (en tant que responsables des tâches) créées et leurs relations. Les unités organisationnelles sont les responsables des tâches à accomplir pour atteindre les objectifs de l'entreprise.

L'organigramme est composé d'unités organisationnelles (formes ovales) et de postes de travail (formes rectangulaires). La modélisation de l'organigramme est le point de départ

de la modélisation de l'entreprise. Elle permet de déclarer tous les acteurs des processus, et de réutiliser ces objets tout au long de la modélisation. Ils ne sont pas assez expressifs pour modéliser des groupes de processus coopératifs.

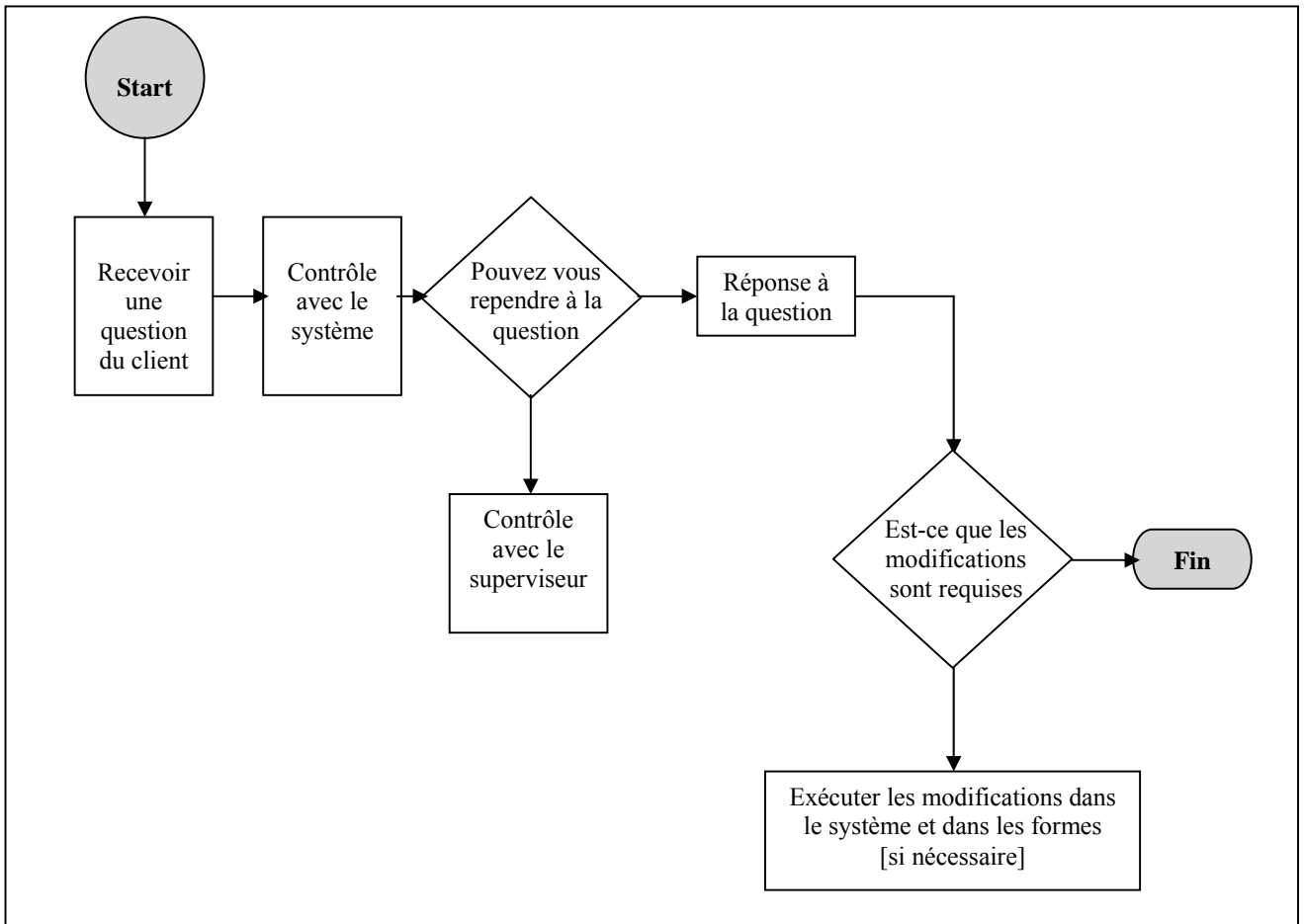


Figure 1.5 : Exemple d'organigrammes

### 1.7.2 Les diagrammes des flux de données DFD (Data Flow Diagrams)

Les DFDs s'intéressent au flux logique des données entre les différents processus du système plutôt qu'au flux de contrôle [Alter02]. Les DFDs étaient les outils clés de modélisation dans les méthodes *classiques* de développement des systèmes d'information Merise et SADT.

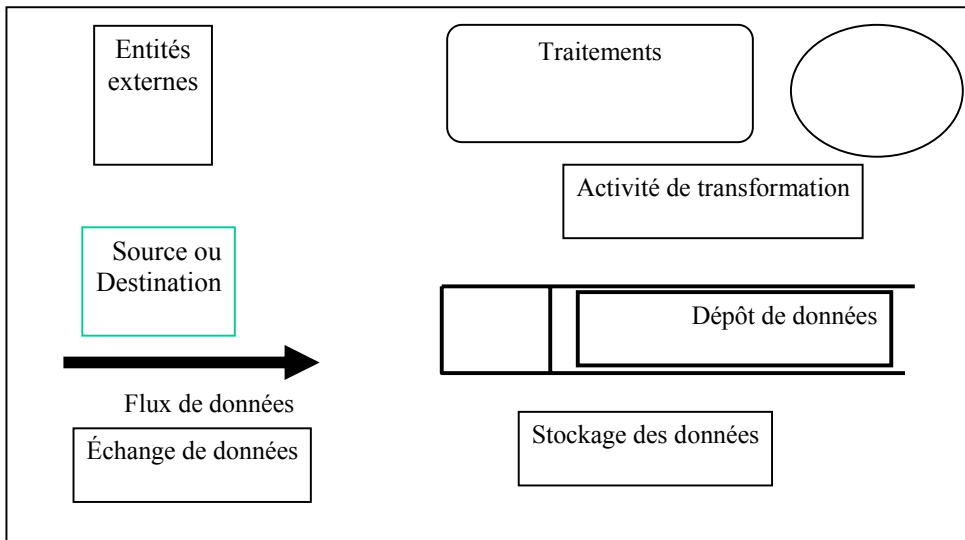


Figure 1.6 : Symboles de base des DFDS.

Les quatre symboles de base des DFDs sont : le carré représentant les sources et les destinations externes des données. Le rectangle arrondi, représentant les processus. Le dépôt de données est représenté par un rectangle ouvert. Une flèche étiquetée représente le flux de données.

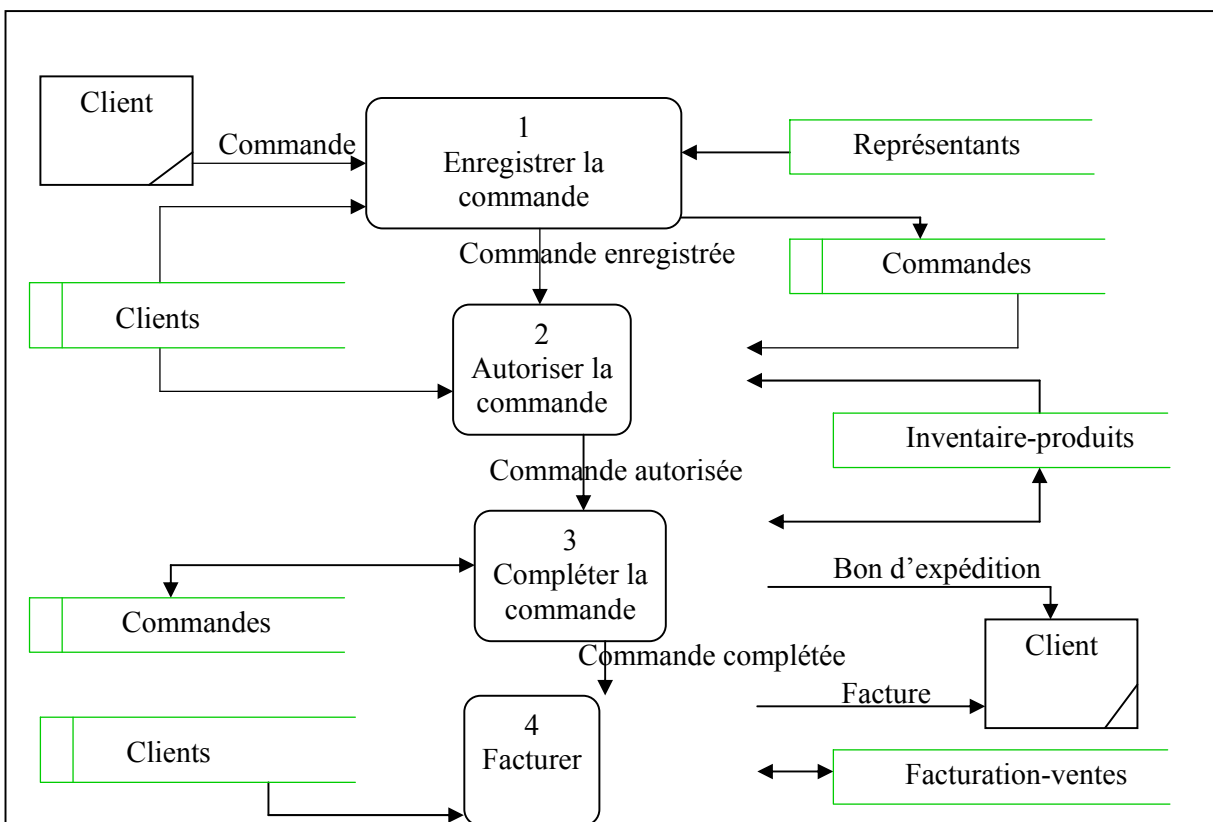


Figure 1.7 : Traitement des commandes de la clientèle représenté par un DFD.

Les DFDs sont probablement les plus simples et jusqu'à maintenant sont considérés comme une des plus puissantes techniques de modélisation des processus. De part leur petit ensemble de vocabulaire et de règles, les DFDs sont facilement lisibles par les utilisateurs. Cependant, du fait que le timing (contrôle de flux) n'est pas pris en considération par les DFDs, il n'est pas possible de capter complètement et de façon non ambiguë le comportement des processus.

### 1.7.3. IDEF

La série IDEF (Icam DEFinition) a été développée dans les années 70 par l'US Air Force dans un projet nommé ICAM (Integrated Computer Aided Manufacturing). Elle regroupe cinq méthodes complémentaires: IDEF0/SADT, IDEF1-IDEF1x, IDEF2, IDEF3. Nous donnerons les grandes lignes de IDEF0, IDEF2, et IDEF3 [Darras 05].

#### 1.7.3.1 IDEF0

La méthode SADT, acronyme de **S**tructured **A**nalysis and **D**esign **T**echnique fût développée par D.T.Ross à la fin des années 60 aux USA. Au départ, elle a été utilisée pour la modélisation de systèmes logiciels au cours des étapes d'analyse et de conception. Le département de la défense américain a standardisé et rendu publique la méthode SADT sous le nom d'IDEF0 [Marca88]. Elle est très utilisée pour la modélisation du *point de vue fonctionnel*. Son objectif est de réaliser des schémas directeurs afin de mieux connaître la situation actuelle d'une entreprise. Elle permet de dégager à partir de l'analyse des besoins, les spécifications fonctionnelles du système à concevoir. Elle repose sur un graphe simple dont:

- *les noeuds* sont les activités du système. Ces activités permettent la transformation d'un flux intrant en un flux extrant (réalisation d'une tâche). Cette transformation se fait à partir de directives de contrôle (conditions exigées pour produire la sortie correcte) en s'appuyant sur des mécanismes (moyens mis en oeuvre pour exécuter l'activité).

- et *les flèches* sont les flux intrants et sortants des activités précédemment décrites.

Le résultat de cette représentation (figure 1.8) s'organise par une suite de diagrammes, textes, etc. ...

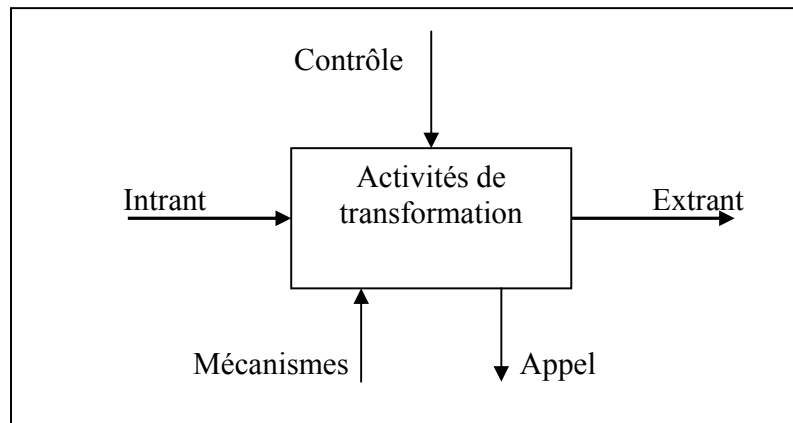


Figure1.8 : Le modèle SADT/IDEF0.

### 1.7.3 .2 IDEF1

La méthode IDEF1 a été conçue pour développer des modèles reflétant l'intégration de l'ensemble des informations de l'entreprise; *point de vue informationnel*. L'approche d'IDEF1 est la suivante :

1. établir un modèle informationnel,
2. concevoir la base de données du modèle,
3. implémenter et installer la base de données ainsi que les fonctions et les procédures associées.

La méthode IDEF1 offre un ensemble de règles et procédures pour la création des modèles informationnels. Elle incorpore les graphiques, le texte et les formes nécessaires pour la meilleure description du modèle. Il existe deux composants fondamentaux dans cette représentation :

- *les diagrammes* : les caractéristiques du modèle informationnel, représentées selon un ensemble de règles et de procédures,
- *le dictionnaire* : chaque élément du modèle est décrit textuellement pour obtenir une définition explicite.

### 1.7.3 .3 IDEF2

IDEF2 est un langage de modélisation du comportement d'un système basé sur le principe des files d'attente. C'est une méthode complémentaire à SADT/IDEF0 qui vise à répondre aux

lacunes du point de vue analyse des aspects dynamiques d'un système. Cette méthode est basée sur quatre modèles : système physique, flux des entités, contrôle du système et gestion des ressources. IDEF2 aborde donc de façon privilégiée les vues informationnelles et ressources sur des niveaux d'abstraction proches de l'exécution.

#### 1.7.3 .4 IDEF3

La méthode IDEF3 est proposée en 1992 pour dépasser les limites d'IDEF0 en matière de modélisation du comportement de l'entreprise, donc la représentation des processus opérationnels.

IDEF3 modélise, comme IDEF0, les processus sous forme d'un enchaînement d'étapes, appelées unités de comportement, connectées par des boîtes de jonction et des liens. L'avantage, par rapport à l'utilisation d'IDEF0, est la possibilité de décrire un flux de contrôle du processus de manière plus complexe. Par exemple, les boîtes de jonctions permettent de représenter, à l'aide de connecteurs logiques (AND, OR ou XOR), les différents flux, les phénomènes de séquençement, de parallélisme ou de synchronisation d'activités.

IDEF2 et IDEF3 représentent donc *le point de vue dynamique*.

#### 1.7.4. Réseau de Pétri (RdP) :

Fortement utilisé dans le domaine industriel, il sert à traiter les problèmes de synchronisation d'activités. Les notions graphiques sont: les places (noeuds), les arcs (flèches) et les transitions (contrôles). Les places correspondent aux activités des processus modélisées. Les arcs sont associés aux évolutions du processus et des flux d'informations.

Les transitions représentent les évènements ou les conditions à vérifier pour avancer dans le processus.

#### 1.7.5. La méthode GRAI

La méthode GRAI (Graphes à Résultats et Activités Interreliés) impose une démarche originale, puisque le point d'entrée dans la conception du modèle est la partie décisionnelle d'un système [Doumeingts84] et [Roboam 93]. La méthode GRAI définit un centre de décision comme le croisement d'une fonction de l'entreprise et d'un horizon/période de décision. La grille GRAI permet de différencier les liaisons de dépendance fonctionnelle (double flèche,

transmission d'une consigne ou d'un objectif) des liaisons informationnelles (simple flèche, transmission d'un flux informationnel) entre centres de décisions.

Pour compléter la grille, la mise en oeuvre de la décision est détaillée en réseau d'activités. Ce réseau (figure 1.10) permet de différencier les activités "d'exécution" de celles de "décision".

Au début des années 90, la méthode GRAI a donné naissance à la méthodologie GIM (GRAI Integrated Methodology). GIM propose un cadre conceptuel basé sur la norme ENV 40003 (dimensions d'abstraction et de généralité) et utilise les méthodologies suivantes pour représenter différents points de vue :

- les formalismes MERISE (entité-relation) pour la vue informationnelle,
- la méthodologie IDEF0/SADT pour la vue physique et la vue fonctionnelle,
- la grille et le réseau GRAI pour la vue décisionnelle.

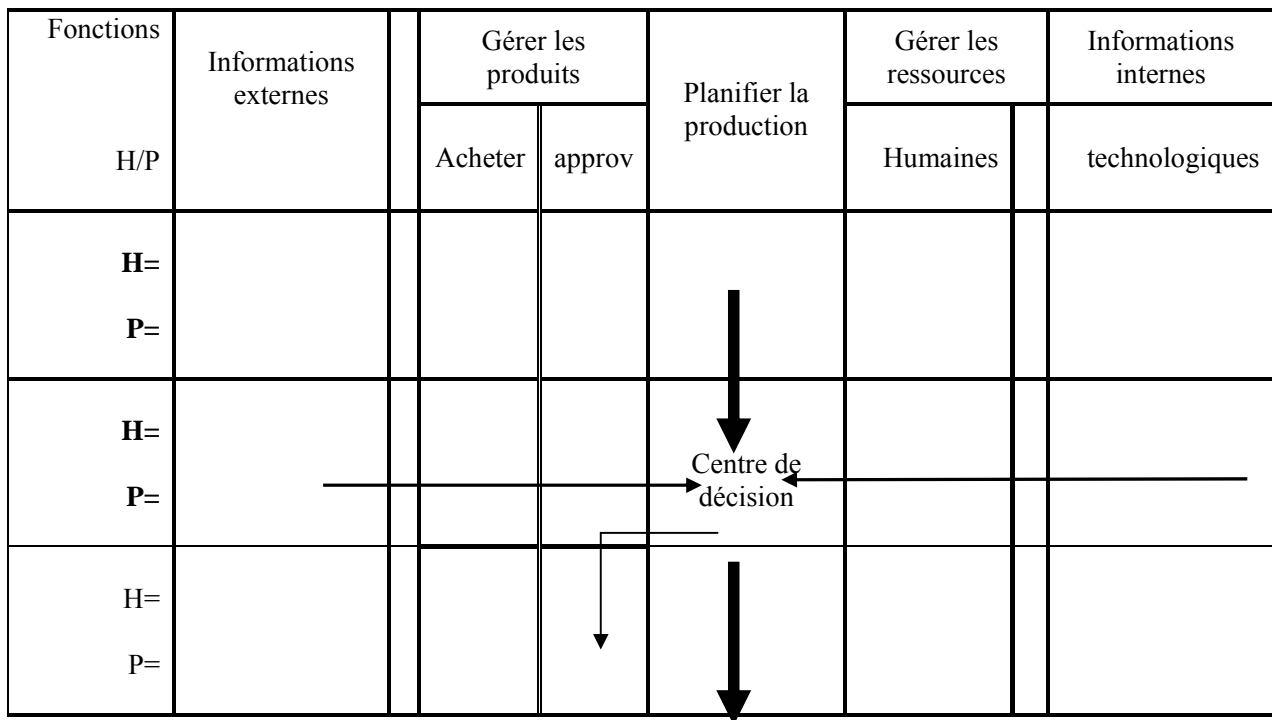


Figure 1.9 : La grille de GRAI

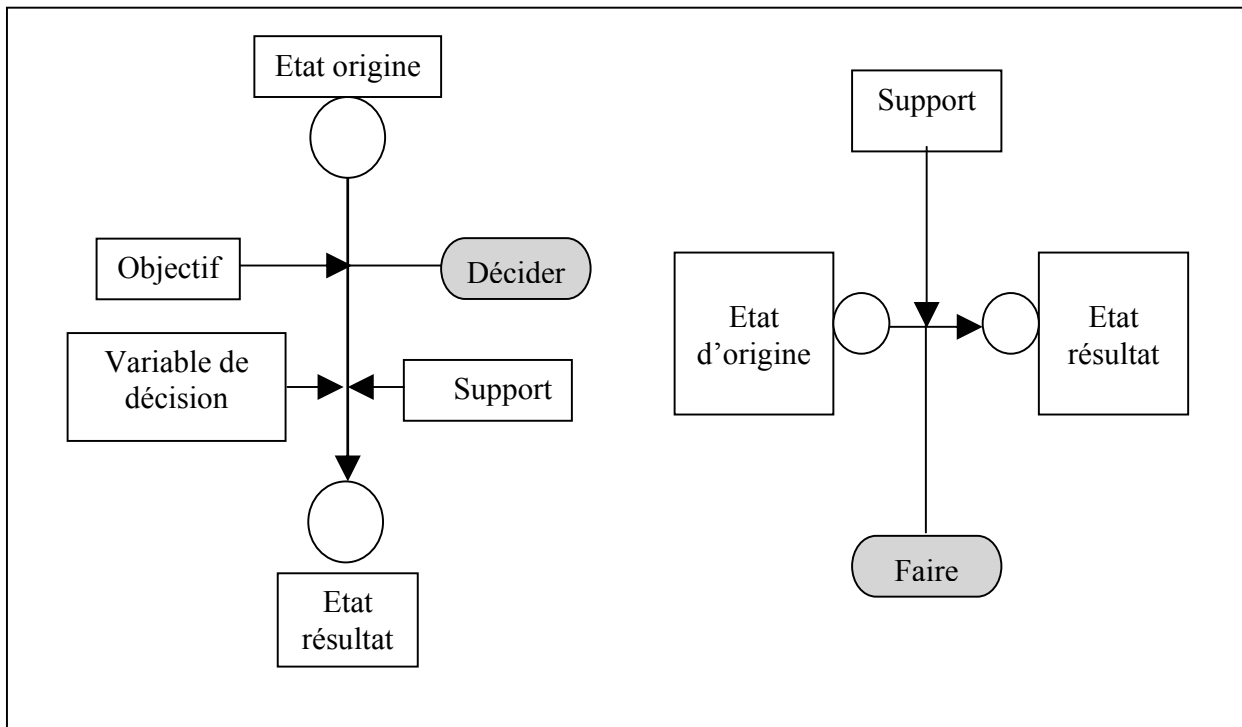


Figure 1.10 : Réseau GRAI, activité de décision et activité d'exécution

### 1.7.6. CIMOSA

La méthodologie CIMOSA (Open System Architecture for CIM) a été développée par le consortium AMICE dans le cadre du programme ESPRIT, entre les années 1984 et 1994. Plus d'une vingtaine d'entreprises implantées en Europe ont contribué à CIMOSA. [Lutherer96]. La méthodologie CIMOSA repose sur une séparation entre l'environnement de l'ingénierie et l'environnement opérationnel de l'entreprise. Son édifice comprend:

- 1- Un cadre de modélisation (incluant une architecture de référence qui fournit une intégration conceptuelle par unification sémantique).
- 2- Une infrastructure intégrante (permettant l'intégration physique et l'intégration des applications).
- 3- Un cadre méthodologique (couvrant le cycle de vie du système de production et assurant la cohérence de l'ensemble).

Le but du *cadre de modélisation* de CIMOSA ou CUBE CIMOSA est de fournir un cadre conceptuel, une méthode et des outils de modélisation pour assister l'utilisateur dans le développement du modèle particulier, propre à son entreprise. Il est composé de deux parties:



Une architecture de référence, générique et réutilisable dans différents projets et une architecture particulière propre à l'entreprise.

Le cadre de modélisation développé dans CIMOSA (figure 1.11) s'articule autour de trois axes de modélisation orthogonaux:

-L'axe de généricité qui suggère de construire le modèle particulier de l'entreprise à partir de modèles partiels, eux mêmes exprimés en termes de construction générique de base.

-L'axe de génération qui propose de modéliser d'abord les besoins de l'entreprise. Il est appelé aussi axe des vues (fonctionnelle, informationnel, ressources et organisation).

-L'axe de dérivation qui invite à modéliser d'abord les besoins de l'entreprise, puis les spécifications de conception et enfin la description de l'implantation.

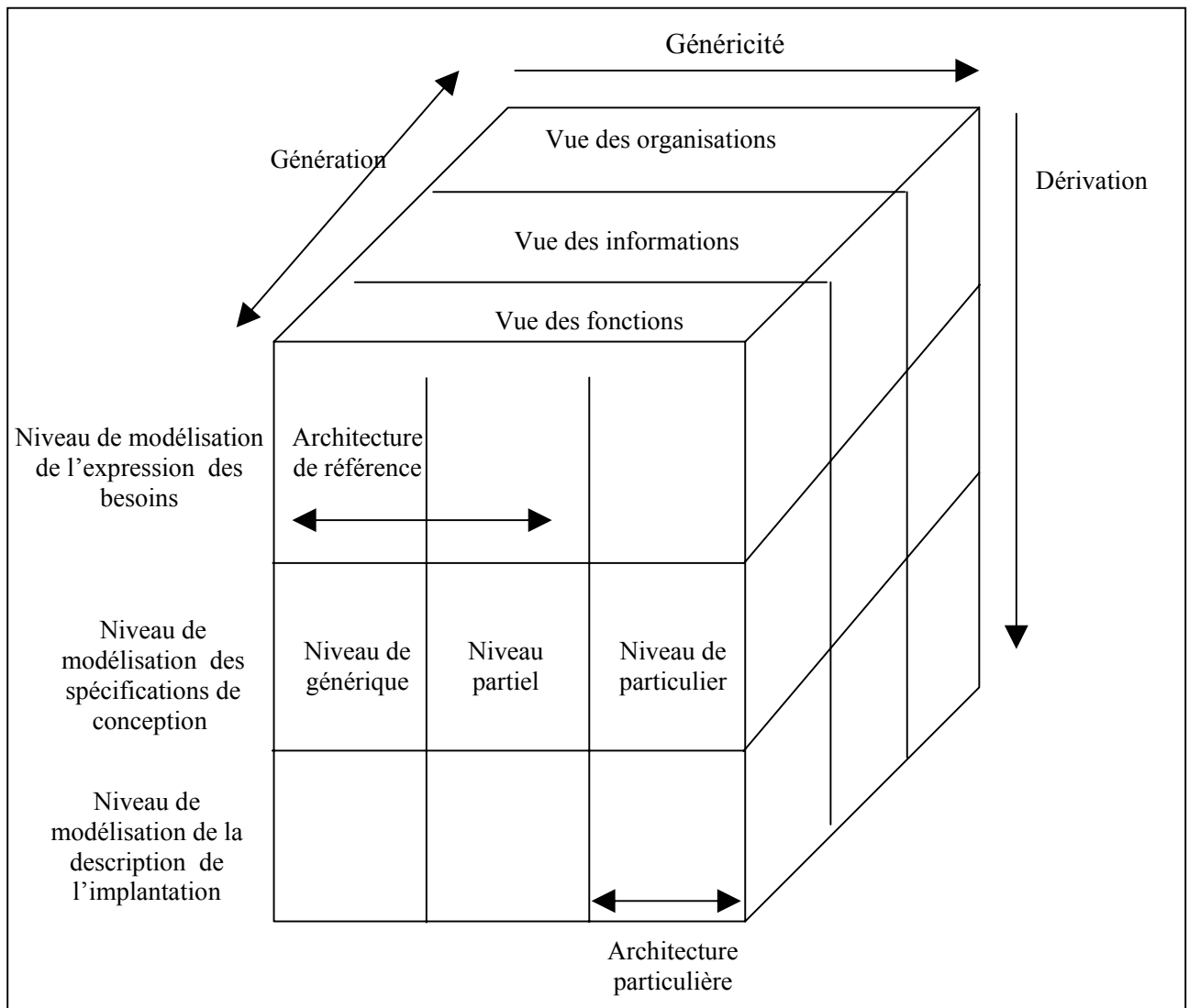


Figure 1.11 : Le cadre de modélisation de CIMOSA.

CIMOSA fournit un ensemble très riche de concepts de base dans son langage de modélisation, couvrant les quatre vues mentionnées ci-dessus.

Le modèle de CIMOSA est basé sur les concepts d'événements, de processus, d'activités, et d'opérations.

CIMOSA propose une modélisation cohérente de l'entreprise, depuis l'expression précise des besoins jusqu'à une description conforme de l'implantation. Toutefois, la représentation textuelle préconisée par CIMOSA pendant l'analyse comportementale ne permet pas de visualiser facilement le comportement d'un processus [Lutherer96].

### 1.7.7 PERA

PERA, (acronyme de Purdue Enterprise Reference Architecture) a été développée en 1990 par the Purdue Laboratory for Applied Industrial Control, université de Purdue, USA [Willams92].

Son objectif est d'établir les bases pour le traitement des fonctions mises en oeuvre par l'être humain dans le domaine de l'intégration des entreprises.

La description des tâches et des fonctions de l'entreprise est décomposée en deux grands courants [Tham96]:

- Le courant des informations.
- Le courant de la production.

Dans le modèle de PERA, les classes des fonctions concernant la décision, le contrôle et l'information sont regroupées dans un seul courant appelé fonction d'information.

- Pendant l'implémentation, les deux courants cités ci-dessus sont réaménagés dans trois jeux d'implémentation de tâches et de fonctions:

a- Les activités de l'être humain qui sont:

- Activités d'information.
- Activités de production.

b- Les activités d'information non effectuées par l'être humain.

c- Les activités de production non effectuées par l'être humain.

Différemment de CIMOSA qui définit quatre vues: fonctionnelle, informationnelle, ressources et organisationnelle, PERA se focalise sur seulement deux vues: Une vue fonctionnelle et une vue d'implémentation.

Le paradigme de modélisation dans PERA s'organise autour de la représentation des tâches du système d'information, de la production et celles effectuées par l'être humain de l'entreprise modélisée. Le schéma de représentation de la connaissance des tâches citées ci-dessus est similaire à celui utilisé dans IDEF. Il comporte des entrées prenant en compte le temps, des paramètres de validation, processus de transformation, sorties et mémorisation des entités.

### 1.7.8. GERAM

L'architecture de référence GERAM (acronyme de **G**eneric **E**nterprise **R**eference **A**rchitecture and methodology) a été introduite par le groupe de travail IFAC/IFIP Task Force on Architectures Enterprise Integration. Après avoir analysé les principales architectures de référence disponibles, à savoir CIMOSA, GRAI-GIM et PERA, le groupe a constaté qu'il fallait préserver le meilleur des méthodes de modélisation et architectures existantes pour créer une nouvelle architecture ayant les qualités de ses aînés sans leurs défauts.

GERAM a plusieurs objectifs, nous citerons [Williams94]:

- Fournir un environnement de modélisation consistant qui va mener éventuellement à un code exécutable par l'ordinateur.
- Promouvoir une ingénierie pratique pour des structures réutilisables des modèles standard.
- Se munir d'une méthodologie détaillée pour l'utilisation, de laquelle le développement personnel de tout type d'entreprise puisse facilement découler.
- Donner le meilleur traitement possible des capacités d'une entreprise d'un point de vue des systèmes.
- Etre générique à tout type d'entreprise sans se soucier de la complexité de l'industrie et de ses applications.
- Fournir une unification des perspectives pour la production, traitements, développement de l'entreprise et une gestion stratégique.

GERAM est basée sur un modèle graphique matriciel du cycle de vie d'une entreprise, utilisé comme base pour la comparaison et l'évaluation des compétences de chacune des architectures étudiées. Ce modèle a été structuré pour inclure une présentation des capacités et points forts des architectures.

### 1.7.9. Diagrammes d'enchaînement de processus d'ARIS

ARIS (Architecture for Integrated Information Systems) [Scheer99] est une méthode de modélisation d'entreprise fortement utilisée dans le monde industriel (SAP3, BaanERP4, etc.). ARIS définit Trois vues : la vue fonctionnelle, la vue informationnelle et la vue organisationnelle. Chaque vue est traitée à part, et les relations entre les trois vues sont représentées par des contrôles de flux. La vue de processus dans cette méthode est basée sur les diagrammes d'enchaînement de processus ou (*Process Chain Diagram* (PCD)).

Les PCD peuvent exister sous deux formes : *Tableau* ou *Diagramme*. Le *Tableau* réunit les entités provenant des différents modèles de vue, ce qui permet d'offrir une représentation "étendue" des processus de l'entreprise. Il permet de séparer les concepts (Événement, Fonction, Données, Système applicatif et unité d'organisation) et de les présenter dans des couloirs. Cette représentation offre une vue générale de la composition structurelle des processus et de leurs relations avec les autres vues de l'entreprise. Le *Diagramme* reprend exactement les mêmes concepts, mais avec une représentation "à plat".

### 1.7.10. La Chaîne de Processus Événementielle (CPE)

SAP AG, le fournisseur leader des solutions logicielles des entreprises a développé le modèle de référence R/3. Il a été initialement destiné pour l'identification des possibilités pour l'optimisation des routines et des procédures dans une organisation. Son plus grand apport est le contrôle logique; " Quand quelque chose doit-il arrivé et dans quel ordre". Les processus sont décrits par les " Event Driven Process Chains". Une chaîne de processus événementielle comprend :

- **L'enchaînement des fonctions** (représentées par des parallélogrammes) dans le sens du processus de l'entreprise.
- **Les événements déclencheurs et résultats** (représentés par des rectangles) pour chaque fonction. La désignation d'un événement doit comprendre à la fois l'objet support ('patient') et l'information de changement d'état de cet objet ("arrivé"). Comme les événements définissent l'état ou la condition qui déclenche une fonction ainsi que l'état qui en marque l'achèvement, les noeuds de départ et d'arrivée d'une telle CPE sont toujours des événements.

Un événement peut déclencher plusieurs fonctions simultanément et, inversement, une fonction peut engendrer plusieurs événements. Pour pouvoir représenter ces ramifications

et des boucles de traitement dans une CPE, on utilise des connecteurs sous forme de cercles. Ceux-ci ne sont cependant pas des simples points de convergence ou de divergence; ils définissent également des connexions logiques entre les objets qu'ils relient. Il existe trois types de règles : ET, OU, XOR (OU exclusif). Une CPE est donc composée d'objets tels que les fonctions, les événements, les entités organisationnelles réalisant les fonctions, les données utilisées et générées, etc.

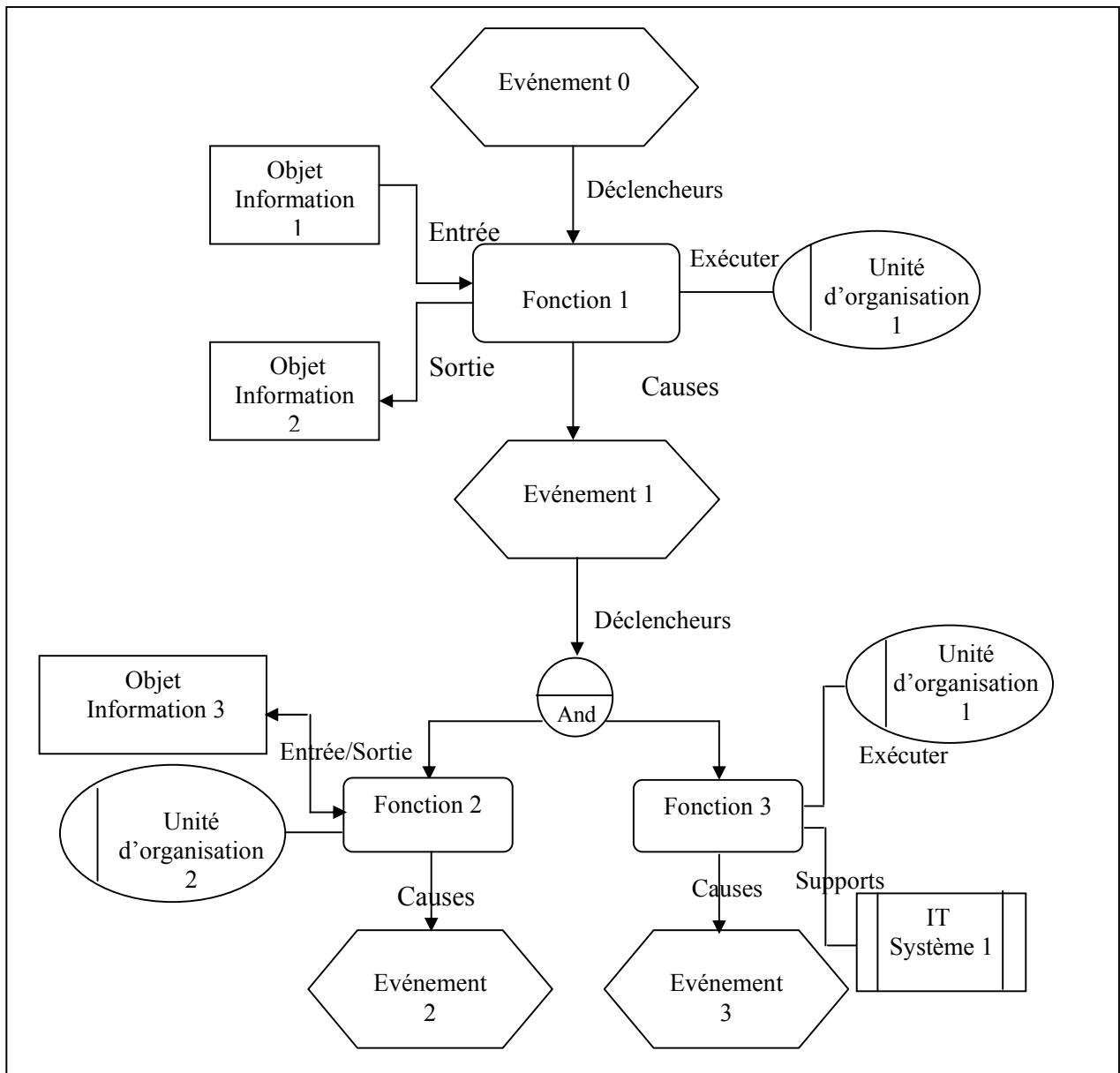


Figure1.12 Chaîne de processus.

### 1.7.11. BPMN

Le Business Process Management Initiative (BPMI), organisme qui anime un courant de standardisation dans le domaine du management par les processus métiers, a mis au point une norme de notation de modélisation des processus métiers ; la "Business Process Modeling Notation (BPMN)". La spécification BPMN 1.0 a été libérée au public en Mai 2004.

Cette spécification représente plus de deux années d'efforts du Groupe de travail de notation BPMI.

L'objectif principal de l'effort de BPMN était de fournir une notation qui soit facilement compréhensible par tous les utilisateurs métiers, depuis les analystes qui créent les premières ébauches des processus, jusqu'aux développeurs techniques responsables de l'implémentation de la technologie qui va s'acquitter de ces processus, et, enfin, les gens d'affaires qui vont gérer et contrôler ces processus.

La BPMN sera également soutenue par un modèle interne qui permettra la génération d'exécutables sous le format BPEL (business Process Executable Language). Ainsi, BPMN veut créer un pont standard pour l'écart entre la conception et la mise en œuvre des processus métiers [Touzi07].

#### 1.7.11.1 Notions de base

Les éléments de base de représentation graphique avec BPMN sont de trois types :

- **les conteneurs** (point de vue *organisationnel*) qui sont les couloirs ("*pool*") et les bandes ("*lane*"). Ce sont des partitions du processus qui relèvent d'un acteur ou d'une entité organisationnelle particulière.
- **Les noeuds du graphe** (point de vue *fonctionnel*) représentent les activités ("*activities*", symbole rectangulaire), les événements ("*events*", symbole circulaire) et les branchements conditionnels ("*gateways*", symbole losange).
- **Les arcs** (point de vue *informationnel*) représentent les flux d'information. Il en existe trois types : les flux de contrôle séquentiel, qui caractérisent une communication interne (dans un même couloir ou une même bande, "*sequenceflow*", trait plein), les flux de message, qui caractérisent une communication interconteneurs ("*message flow*", trait pointillé) et les associations ("*Association*", trait incliné). Cette dernière est utilisée, par exemple, pour

associer un élément de documentation d'une entité, pour identifier des données en tant qu'entrées ou sorties d'une activité ou pour associer une compensation à une activité, etc.

Les évènements dans BPMN sont typés et à chaque type correspond un symbole particulier normalisé: début et fin, arrivée d'un message, échéance d'une temporisation, exception, nécessité de mettre en oeuvre une compensation (défaire ce qui a été fait précédemment pour revenir à un état stable).



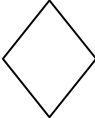
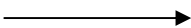
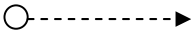

| Objet                            | Symbole  |
|----------------------------------|--|
| Événement (Event)                |    |
| Activité (Activity)              |   |
| Branchement séquentiel (Gateway) |  |
| Flux séquentiel (Sequence Flow)  |  |
| Flux de messages (Message Flows) |  |
| Associations                     |  |

Figure 1.13 : Symbolisation BPMN.

En résumé, les points forts du formalisme BPMN sont [Touzi07] :

- Une notation intuitive et plus ergonomique à l'usage des acteurs de l'organisation et de la gestion d'entreprise.

- Un vocabulaire riche et adapté aux besoins de conception de processus métiers complexes – ensemble de concepts et de relations – rigoureusement défini pour fournir un socle robuste à l’outillage des approches processus.
- Un lien fort avec le format d’échange BPEL.

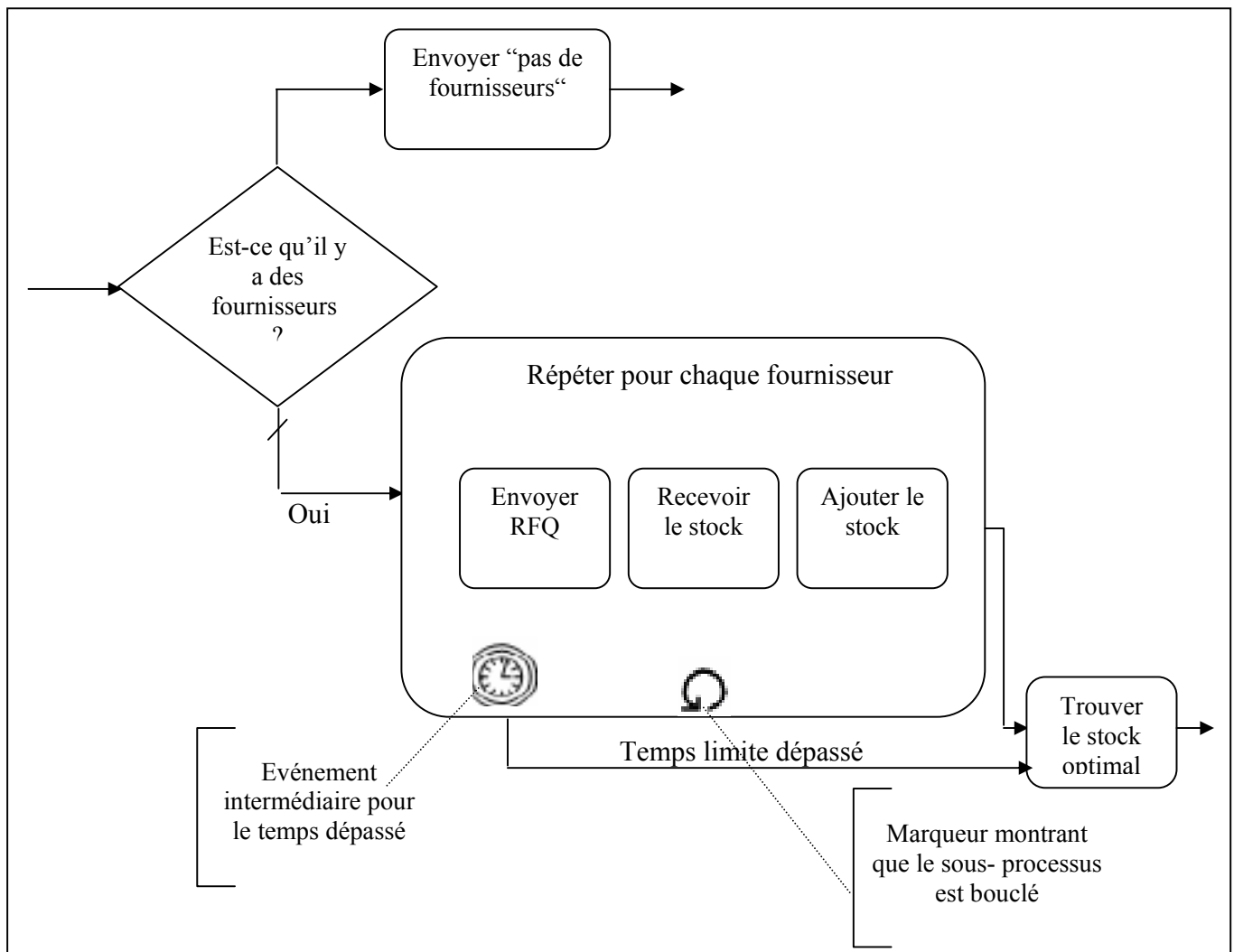


Figure 1.14 : Segment d'un processus.

### 1.7.12. Unified Modelling Language: UML

"UML est un langage pour spécifier, visualiser, construire, et documenter les artefacts des systèmes logiciels, ainsi que pour la modélisation d'entreprise et des systèmes non logiciels" [www.omg.org].



Comme son nom l'indique, ce langage est né de la fusion de plusieurs langages de modélisation, issus notamment de la méthode de Grady Booch particulièrement adaptée à la conception et à l'implémentation, de la méthode OOSE (Object Oriented Software Engineering) de Ivar Jacobson: qui permettait essentiellement l'expression des besoins, et de la méthode OMT (Object Modelling Technique) de James Rumbaugh: pour l'analyse et applications orientées données.

En 1994 Rumbaugh rejoint Booch chez Rational., puis en 1995 Jacobson rejoint Rational et le 14 Novembre 1997: UML est adopté par l'OMG (ObjectManagement Group). La figure 1.15 montre les différentes étapes par lesquelles est passé UML:

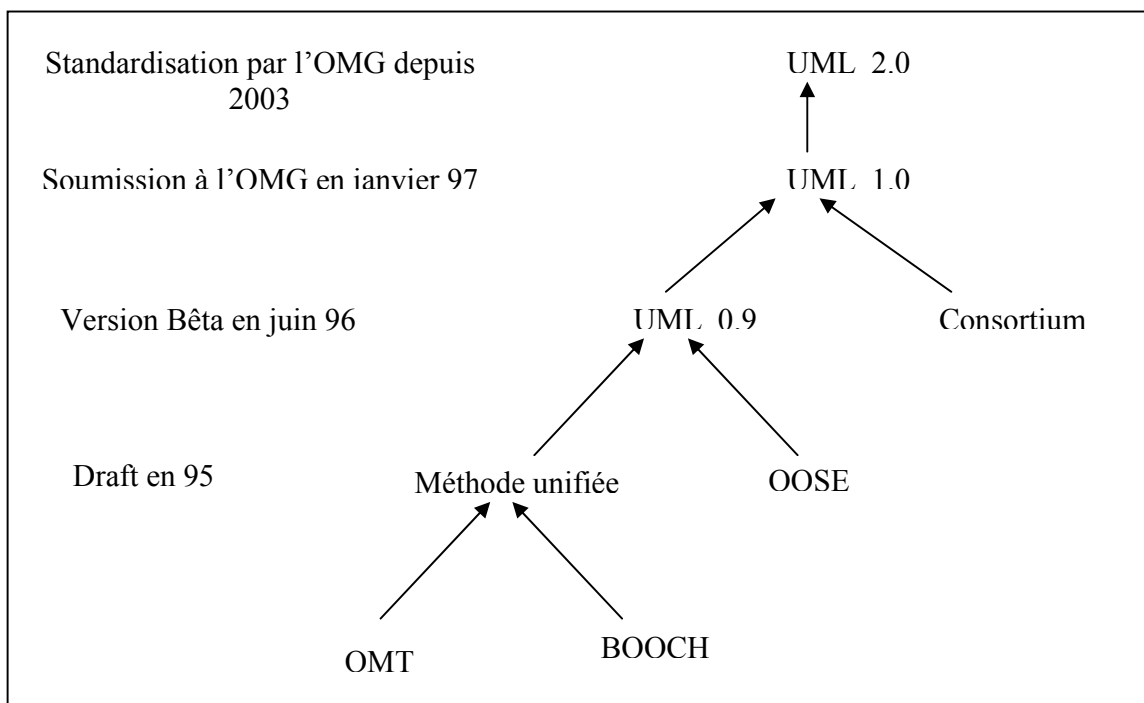


Figure 1.15 : Evolution de UML.

## Les Diagrammes UML

UML présente un ensemble assez riche de diagrammes permettant de décrire les besoins des utilisateurs, ainsi que les propriétés statiques et dynamiques du système. On peut distinguer deux catégories de diagrammes UML:

- **Diagrammes décrivant l'aspect structurel du système :**
  - ❖ *les diagrammes de classes*: expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre ces classes. De plus, ils présentent un ensemble d'interfaces et de paquetages, ainsi que leurs relations.
  - ❖ *les diagrammes d'objets*: sont des graphes d'instances qui incluent les objets et les valeurs de données. Un diagramme d'objets statiques est une instance d'un diagramme de classes, présente un snapshot de l'état détaillé d'un système à un instant donné.
  - ❖ *les diagrammes de cas d'utilisation*: représentent la structure des grandes fonctionnalités nécessaires aux utilisateurs du système. Ils assurent la relation entre l'utilisateur et les objets que le système met en œuvre en décrivant, sous forme d'actions et de réactions, le comportement d'un système du point de vue utilisateur.
  - ❖ *les diagrammes de composants* : présentent les dépendances entre les composants logiciels. Ils incluent les classificateurs (i.e. classes) qui spécifient les composants, et les artefacts qui les implémentent, tels que les fichiers de code source, de code binaire, exécutables ou scripts.
  - ❖ *les diagrammes de déploiements* : présentent la configuration des éléments de traitement en temps d'exécution, ainsi que les composants logiciels, les processus et les objets qui les exécutent.
- **Diagrammes décrivant l'aspect comportemental du système :**
  - ❖ *les diagrammes de séquence*: montrent les interactions entre les objets dont la représentation se concentre sur la séquence des interactions selon un point de vue temporel.
  - ❖ *les diagrammes de collaborations*: présentent l'ensemble des rôles joués par des objets dans un contexte particulier, ainsi que les liens entre ces objets.
  - ❖ *les diagrammes d'états-transitions*: représentent les automates d'états finis du point de vue des états et des transitions.
  - ❖ *les diagrammes d'activités* : sont une variation des diagrammes d'états-transitions dans laquelle les états représentent à la fois la réalisation des actions ou sous activités et les transitions déclenchées par la finalisation des actions ou sous activités.

Les diagrammes de séquence, d'activité et d'état transition sont les plus utilisés pour la modélisation des processus métiers.

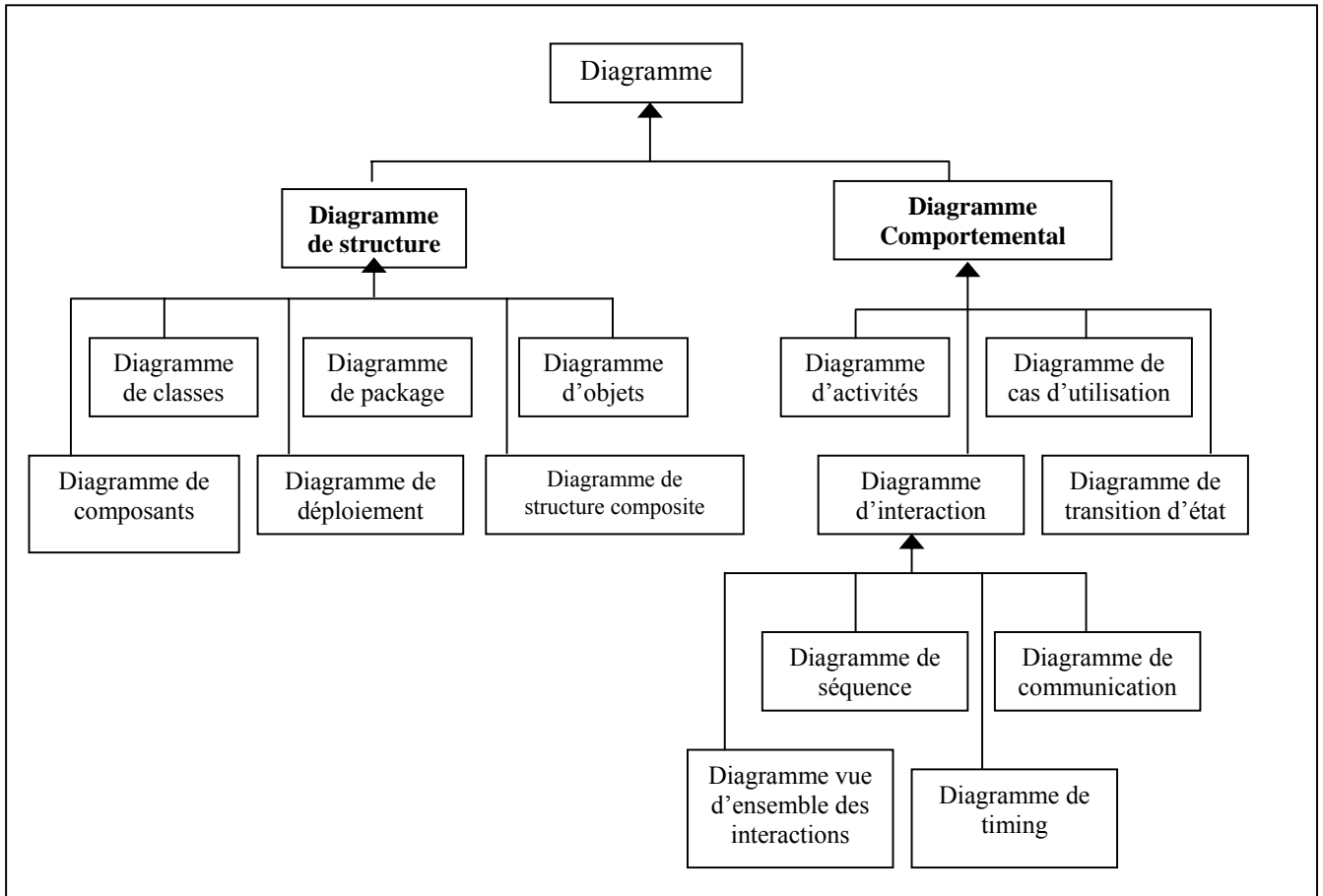


Figure 1.16 : Diagrammes UML

### 1.8. Evolution des organisations

Aujourd'hui, les entreprises rencontrent de nombreux changements dans l'environnement de gestion dus à plusieurs facteurs complexes. L'accentuation de la concurrence est l'un des phénomènes principaux de l'environnement de l'entreprise actuelle. En effet, les entreprises ne sont plus soumises à une compétition nationale ou régionale, mais à une concurrence mondiale dans un marché international. L'économie de globalisation due à la disparition des barrières entre les pays, conduit à de nouvelles formes d'organisation capables de répondre aux exigences des clients dans le monde entier.

Un autre aspect du changement dans l'environnement de gestion des entreprises, est la concentration sur la satisfaction des clients, non plus par une production de masse, mais par une personnalisation et une diversification de produits pour répondre à la variabilité importante des demandes clients. Ces changements révolutionnaires de l'entreprise exigent une forte coopération des différents acteurs. Ceci est favorisé par le développement rapide des technologies de l'information et de communication. Ce développement est visible notamment dans trois aspects :

- L'augmentation de la puissance des ordinateurs. Ceci implique l'augmentation des informations traitées par les entreprises.
- Le second est l'utilisation des données externes à l'entreprise grâce au Web. Ceci favorise le commerce électronique et facilite les échanges.
- Enfin le développement des technologies de communication permet la coopération entre membres géographiquement dispersés et une transmission de l'information très efficace.

Tous ces facteurs ont mené à l'apparition de nouvelles formes de l'entreprise. L'entreprise étendue, l'entreprise virtuelle et la chaîne logistique étendue en sont des exemples.

Mais avant d'entamer les nouvelles structures des entreprises, on rappelle quelques structures de l'organisation classique [Megarsti97].

### 1.8.1. Les organisations classiques

Nous pouvons distinguer cinq directions classiques:

**1-La structure simple** : repose sur un individu. C'est la structure la plus risquée car elle repose sur la santé et la volonté de son unique individu.

**2-La bureaucratie mécaniste**: Cette structure peut très bien fonctionner dans des environnements stables et simples, cependant, elle souffre de rigidité.

**3-La bureaucratie professionnelle**: adaptée à la production standardisée mais pas à l'innovation. C'est également une structure rigide.

**4-La structure divisionnalisée**: adoptée lorsqu'une bureaucratie mécaniste fonctionnant dans un environnement stable décide de diversifier horizontalement ses lignes de produits ou de services.

**5-L'adhocratie:** C'est une structure qui évolue dans un environnement à la fois complexe et dynamique. Les fréquents changements de produits qui s'y opèrent permettent des conditions dynamiques.

On pourra retenir que si les structures simples et mécanistes ont été les structures d'hier et que la bureaucratie professionnelle et la structure divisionnalisée sont celles adoptées généralement aujourd'hui, l'adhocratie semble être la structure de demain [Khandriche97].

### **1.8.2. Les nouvelles structures d'organisation**

Plusieurs formes sont apparues. On peut citer :

#### *1.8.2.1. Entreprise réseau*

C'est un ensemble d'entreprises liées les unes aux autres par un cycle de production. Le lien n'est ni juridique, ni structurel. Il revêt souvent la forme de simples accords. Ces entreprises ont en commun un puissant système de coopération fonctionnelle [Butera91].

L'entreprise réseau est une entreprise qui a choisi d'étendre son action pour partager des défis d'environnement économique dont elle maîtrise d'autant mieux la complexité qu'elle est outillée des compétences de ses partenaires. Des partenaires convaincus comme elle-même des multiples avantages d'une alliance [Poulin84].

Cette forme d'organisation est aussi désignée dans la communauté scientifique par "filiale" [Bellon84], "constellation d'entreprises" [Lorenzoni88] ou "entreprise modulaire" [Brilman95]. Cette même communauté distingue trois types d'entreprise réseau :

- *L'entreprise réseau hiérarchique* qui désigne une organisation composée de relations quasi hiérarchiques entre les donneurs d'ordres et les entreprises.
- *L'entreprise réseau au centre de gravité*, où les différents intervenants sont le donneur d'ordre (appelé firme pivot), les sous-traitants d'intelligence (appelés sous-traitants de premier niveau) et les sous-traitants de capacité ou de spécialité (appelés sous-traitants de deuxième niveau).
- *L'entreprise réseau coopérative* qui ne possède pas de centre de gravité et qui met en oeuvre des relations d'interdépendances entre les partenaires.

### 1.8.2.2. *Entreprise virtuelle*

L'entreprise virtuelle est constituée de plusieurs entités géographiquement dispersées et gérées comme une entité unique. Toutefois, des regroupements d'entités peuvent avoir une gestion propre. C'est une organisation temporaire formée à partir d'alliances stratégiques qui peuvent se dissoudre quand l'affaire ou le projet sont terminés. Les partenaires de ce type d'organisation apportent leurs compétences et leurs ressources pour conquérir les marchés mondiaux, chacun contribuant à ce en quoi il excelle [Brilman95].

C'est un consortium temporaire de partenaires, réalisant une activité à valeur ajoutée, établi dans l'objectif de livrer des produits ou des services à un client. Le cycle de vie d'une entreprise virtuelle est restreint : elle est créée à partir d'un réseau de partenaires pour une tâche définie et elle est dissoute à l'accomplissement de celle-ci [Voster04].

L'entreprise virtuelle peut être définie comme étant une agrégation temporelle de compétences et de ressources qui collaborent ensemble pour un besoin spécifique tel une opportunité d'affaire [Goranson 97].

L'entreprise virtuelle se distingue des autres formes par un cycle de vie restreint. Elle dure la durée de vie de développement d'une opportunité.

### 1.8.2.3. *Entreprise étendue*

Le concept d'entreprise étendue est principalement fondé sur l'idée de considérer l'entreprise manufacturière traditionnelle et d'y incorporer les entreprises avec lesquelles elle entretient des relations dans le cadre de sa production [Browne95]

Le système manufacturier d'une entreprise est vu dans un contexte plus global qui englobe les relations de l'entreprise avec ses clients et fournisseurs [Cloutier99].

Les concepts d'entreprises étendues et d'entreprises virtuelles sont assez proches. Jadev et brown disent que la différence entre les deux concepts est juste une question de sémantique [Jagdev98]. Ils affirment, par contre, qu'il existe une différence dans l'objectif de coopération qui est plus prononcé dans l'entreprise virtuelle. Dès lors, pour Rolstadas, [Rolstadas 98] l'entreprise étendue n'est qu'un cas particulier de l'entreprise virtuelle.

#### 1.8.2.4. Réseau d'entreprises

Une entreprise distribuée est dite réseau d'entreprises lorsqu'elle réunit principalement des entreprises considérées dans leur intégralité, et non seulement des départements, des ateliers ou des dirigeants [Poulin84].

Un réseau d'entreprises est une alliance coopérative d'un ensemble d'entreprises établie conjointement pour l'exploitation d'opportunités à travers la création d'entreprises virtuelles [Vesterager99].

### 1.9. Processus métiers et entreprises virtuelles

Le but de l'EV est d'optimiser les affaires des entreprises. Ceci revient en premier lieu à optimiser leurs processus métiers.

Le processus métiers dans une entreprise virtuelle est un processus dynamique, qui est composé de tous les partenaires de l'EV. Il possède des caractéristiques que l'entreprise générique n'a pas, comme le suivi des performances des partenaires, la coordination des relations des partenaires et la distribution des ressources entre les partenaires [Quanlong99].

De ce fait, une seule méthode ne peut souvent pas être appliquée à tout projet de développement sans des modifications ou des ajustements. Cela dépendra de la taille et de l'objectif du projet, des parties concernées (organisations), de l'ancien processus de développement ou de méthodologies, de la complexité des processus métiers et des applications, du type d'applications et de leur besoin de d'intégration, et d'autres facteurs [Roser05].

Un langage de modélisation intégré doit donc être appliqué pour décrire de manière complète le modèle de processus métier dans les EV. Les processus de modélisation connus tels que IDEF0, IDEF3, les réseaux de Petri, etc. ont tous leurs propres caractéristiques, mais ils ont tous une déficience lorsqu'ils sont appliqués à décrire des processus métiers des EV [Yang].

En règle générale, un processus métier peut être décomposé en de nombreuses activités, qui seront affectées à des entités réalisant chacune une tâche particulière. Dans une entreprise virtuelle, les activités métiers sont réparties dans de multiples membres le l'EV et sont réalisées en parallèle. Des relations logiques et temporelles existent entre ces activités. Donc une entreprise

virtuelle peut être modélisée par des objets métiers distribués et communiquant les uns avec les autres. Les activités qu'ils réalisent en parallèle représentent le processus métier de l'EV.

Une méthode intégrée de modélisation des processus métiers basée sur UML et les ECATNets, réseaux de Petri de haut niveau (voir chapitre 2), sera présentée dans le chapitre 5.

### **1.10. Conclusion**

Le but de ce chapitre était de donner une idée sur les entreprises, leur modélisation et les évolutions qu'elles ont subies. Nous avons mis l'accent sur la notion de processus et le rôle crucial que jouent les processus métiers dans la prospérité des entreprises d'aujourd'hui. Ceci nécessite une attention particulière lors de leur modélisation afin d'éviter d'introduire des erreurs.

Cependant, comme nous pouvons le constater, la plupart des modèles des processus métiers présentés, souffrent de l'absence d'approches effectives d'analyse, et vu l'importance des processus métiers dans les entreprises, c'est un vrai danger de laisser passer des anomalies telles que l'interblocage, la vivacité et la répétition multiple. Le danger est d'autant plus grand lorsqu'il s'agit de systèmes distribués et complexes tels que l'entreprise virtuelle.

Justement, la vérification sert à déterminer à l'avance si un modèle de processus présente certains comportements souhaitables.

En effectuant cette vérification au moment de la conception, il est possible d'identifier des problèmes potentiels et, le cas échéant, le modèle peut être modifié avant son exécution. Une analyse des modèles de processus lors de la conception peut grandement améliorer la fiabilité des systèmes.

Plusieurs outils et méthodes de vérification et de validation existent. Le chapitre suivant en donnera un aperçu panoramique.



## Chapitre 2

### Méthodes Formelles pour la vérification des systèmes

#### 2.1 Introduction

Un modèle sert à comprendre, à raisonner et à communiquer avec d'autres acteurs. Pour être exploitables et analysables, les modèles obtenus doivent être cohérents, consistants et complets.

Le but de ce chapitre est de présenter un panorama sur les méthodes de vérification et de validation des systèmes.

Les réseaux de Petri représentent une technique formelle largement utilisée pour l'analyse des propriétés. Un intérêt particulier sera porté sur eux.

#### 2.2 Critères d'un bon modèle

L'utilisateur doit avoir "*confiance*" dans les modèles qu'il manipule. Il doit **s'assurer** de la "qualité" du modèle et être capable de juger des forces et des faiblesses du système modélisé [Chapurlat05].

- **Qualité du modèle** : Selon l'ISO 8402.94, la qualité est "*l'ensemble des caractéristiques d'une entité qui lui confèrent l'aptitude à satisfaire des besoins exprimés et implicites*". Pour un modèle ceci revient à s'assurer de :

- Sa **complétude**: le modèle contient toute l'information nécessaire qui peut être utilisée pour démontrer ou infirmer sans ambiguïté ni doute la véracité d'une affirmation.

- Sa **cohérence** : il n'existe pas de contradiction dans l'information contenue dans un ou plusieurs modèles. La cohérence d'un modèle est donc définie par l'impossibilité de trouver deux informations contradictoires dans la spécification des exigences et, entre les exigences spécifiées et les modèles de conception.

- Sa **pertinence**: Un modèle est pertinent s'il représente le système avec une qualité suffisante. Ce niveau de qualité est relatif selon les langages de modélisation ou l'approche de modélisation choisis. La pertinence nécessite de s'assurer que le modèle répond aux objectifs du modéleur. Ce dernier peut être un utilisateur cible ou un concepteur du système et donc posséder des objectifs différents. L'utilisateur final décrira le système selon une approche boîte noire et spécifiera les exigences

fonctionnelles et non fonctionnelles du système vis-à-vis de son environnement. Le concepteur adoptera une approche boîte blanche pour aboutir à une spécification de conception du système en terme de solution technologique qui soit manipulable sans introduire d'ambiguïté ou de biais (complétude vs. fidélité).

- **De pouvoir juger des forces et des faiblesses** du système modélisé. Cela concerne *la performance* [Berrah97], *la stabilité dans le temps* (est-il possible, suite à une situation imprévue, de revenir vers un fonctionnement maîtrisé ou vers une situation de routine [Montmain06]) et d'*intégrité* (comment le système anticipe-t-il et évolue-t-il pour s'adapter à de nouvelles situations ?).

Des techniques de validation telles que la simulation et l'expertise sont alors utilisées pour analyser le système en question au travers de ses modèles.

### 2.3. Techniques d'analyse

Quand les systèmes deviennent complexes, il devient encore plus difficile d'assurer la qualité du modèle. Pour cela, on recourt à leur analyse par le biais de plusieurs techniques.

#### 2.3.1. Vérification

La vérification répond à la question « *Construisons-nous correctement le modèle ? ("is the system being built right?")* »

La vérification est l'ensemble des actions de revue, inspection, test, preuve automatique, ou autres techniques appropriées permettant d'établir et de documenter la conformité des artefacts du développement vis-à-vis des critères préétablis. (ISO 8402) définit la vérification comme étant "*la confirmation par examen et apport de preuves tangibles (informations dont la véracité peut être démontrée, fondée sur des faits obtenus par observation, mesures, essais ou autres moyens) que les exigences spécifiées ont été satisfaites*".

#### 2.3.2. Validation

La validation consiste à évaluer l'adéquation du système développé vis-à-vis des besoins exprimés par ses futurs utilisateurs. La validation cherche à répondre à la question "*Construisons-nous le bon modèle ? ("is the right system being built?")*". Par définition la validation est la "*confirmation par examen et apport de preuves tangibles que les exigences particulières pour*

*un usage spécifique prévu sont satisfaites. Plusieurs validations peuvent être effectuées s'il y a différents usages prévus" [ISO 8402].*

### 2.3.3 Qualification

Consiste à s'assurer que le modèle peut servir de base à la communication sans ambiguïtés ni interprétation parasite possible entre les activités du projet et/ou entre les acteurs d'un groupe de travail.

### 2.3.4. Certification

Consiste à s'assurer que le modèle respecte une norme et peut servir de base à l'établissement d'un référentiel réutilisable et générique à un domaine. Cela sous entend la nécessaire implication et la responsabilité d'un organisme tiers qui reconnaît la pertinence, la rigueur, l'intérêt du modèle et garantit ces qualités lors de sa diffusion.

## 2.4. Vérification et validation

Dans notre travail on s'intéresse à la vérification et la validation. Parmi leurs objectifs, on peut citer :

- Améliorer l'aide à la décision :
  - ❖ Une décision se justifie usuellement à partir de l'évaluation ou de l'interprétation de modèles du problème, du système cible, etc.
  - ❖ Aide pour diagnostiquer certains faits (règles de construction formalisées, ...)
- Aider à la certification
  - ❖ La certification passe par des représentations, donc des modèles, qui, s'ils sont vérifiés et validés de manière formelle vont faciliter cette dernière.

Il existe de nombreuses techniques de validation et de vérification qu'on peut classifier selon trois critères [chapurlat07] :

- **La technique** est-elle plutôt considérée comme une technique de vérification ou de validation?
- **La focalisation** de cette technique: se focalise-t-elle sur un aspect statique et descriptif, ou se focalise-t-elle sur la dynamique du système étudié ?

- **Le niveau de formalisation** de la technique: est elle construite sur une base formelle, semi- formelle ou, au contraire, non formelle ? Nécessite-t-elle la réécriture du modèle à vérifier ou à valider vers un modèle tiers vérifiable ?

On peut donc distinguer :

- ❖ **Des techniques non formelles:** elles consistent à détecter des erreurs ou ce qui semble l'être en examinant et expertisant le modèle de manière manuelle au cours d'expertises, de revues de modèles, de test, etc. Ce sont donc plutôt des techniques de validation.
- ❖ **Des techniques semi formelles :** il s'agit de techniques dont le niveau de résultat peut être considéré comme indépendant d'une expertise humaine mais sans pour autant revêtir un caractère exhaustif et indiscutable. D'un point de vue statique, il peut s'agir de vérifier la conformité du modèle à un standard existant. D'un point de vue dynamique, les techniques usuelles sont la simulation, l'émulation ou le test. Elles ont pour inconvénient la non exhaustivité des cas envisagés et donc l'impossibilité de garantir l'absence de fautes. De fait, la pertinence et la qualité des résultats issus d'une simulation reposent essentiellement sur l'interprétation et donc sur la qualité du modèle de simulation [Balci02], et sur les compétences et/ou l'expérience des utilisateurs. Il s'agit donc plutôt de techniques utilisables dans le cadre d'une validation ou de vérification en ce qui concerne par exemple le débogage (symbolique ou non) d'un modèle.
- ❖ **Des techniques formelles:** ce sont des techniques basées sur l'emploi de méthodes formelles [Nasa98, Lamsweerde02, Jackson01, Chatel04]. Elles reposent sur l'utilisation de langages et de concepts relevant du domaine des mathématiques.

Dans notre travail, nous nous intéressons particulièrement aux méthodes formelles.

## 2.5. Les méthodes formelles

Les Méthodes formelles forment la base mathématique du logiciel. Une méthode est formelle si elle a une base mathématique solide, normalement donnée à l'aide d'un langage de spécification formel. Par définition, un langage formel est en effet *un langage doté d'une sémantique mathématique adéquate basée sur des règles d'interprétation qui garantissent*

*l'absence d'ambiguïté dans les descriptions produites et des règles de déduction qui permettent de raisonner sur les spécifications afin de découvrir de potentielles incomplétudes, inconsistances ou pour prouver des propriétés* [Petit99, Benzaken91].

Les méthodes formelles permettent de détecter des ambiguïtés, des problèmes de non complétude et d'inconsistance. Elles peuvent aider à découvrir des failles, des contradictions et des incohérences dans la spécification aussi bien qu'à déterminer la correction de l'implantation d'un système.

### **2.5.1. Les mythes autour des méthodes formelles**

L'utilisation des méthodes formelles a une longue histoire. Bien qu'il y ait eu une utilisation significative des méthodes formelles dans les industries critiques [Hinchey95], elles n'ont pas été très bien accueillies en général, par la communauté de développement de logiciels [Bloomfield99]. Il y a eu beaucoup d'idées fausses à propos des méthodes formelles ; dans [Hall90], l'auteur cite sept mythes à leur sujet:

*1- L'utilisation des méthodes formelles produit un logiciel parfait* : non-sens, une spécification formelle est un modèle du monde réel et peut donc inclure des erreurs, des omissions et des malentendus.

*2- Utiliser les méthodes formelles signifie faire de la preuve de programme* : La spécification formelle d'un système est valable sans vérification formelle des programmes, car elle force rapidement dans le cycle de développement, à une analyse détaillée.

*3- Les méthodes formelles ne sont justifiables que pour les systèmes critiques*: L'expérience industrielle montre que les coûts de développement sont réduits pour tous les types de systèmes.

*4- Les méthodes formelles sont pour les mathématiciens*: non-sens, les mathématiques employées sont élémentaires.

*5- Les méthodes formelles augmentent les coûts de développement* : Non prouvé, il y a un déplacement des coûts vers les premières étapes.

*6- Les clients ne peuvent comprendre les spécifications formelles*: Il faut les paraphraser en langage naturel, ou utiliser le prototypage.

*7- Les méthodes formelles ne sont utilisées que pour les systèmes triviaux* : Il y a maintenant de nombreux projets industriels concernant des systèmes non-triviaux qui ont été mis en oeuvre.

### 2.5.2 Classification des méthodes formelles

Il existe plusieurs classifications des méthodes formelles. On peut distinguer les méthodes :

- Basées modèle
- Basées logique
- Algébriques
- Algèbre de processus
- Orientées réseaux (graphiques)

**a- Approches basées modèle** : Un système est modélisé par la définition explicite des états et des opérations qui le transforment d'un état à un autre. Dans cette approche. Il n'y a pas de représentation explicite de la concurrence. Des besoins Non fonctionnels (telles que l'exigence temporelle) peuvent être exprimée dans certains cas.

Exemples de méthodes : Z, VDM, et B.

**b- Approches basées logique** : Dans cette approche la logique est utilisée pour décrire les propriétés du système, y compris les comportements temporels et probalistiques. La validité de ces propriétés est réalisée en utilisant le système d'axiomes associé à la logique. Dans certains cas, un sous-ensemble de la logique peut être exécuté (par exemple, le système de tempura). La spécification exécutable peut ensuite être utilisée pour des fins de simulation et de prototypage rapide.

On peut citer comme exemples: La logique de Hoare, Dijkstra, logique temporelle etc.

**c- Approches algébriques** : Dans cette approche, une définition implicite des opérations est donnée en reliant le comportement des différentes opérations sans définir la signification des états actuels. Similaire à l'approche fondée sur les modèles, il n'y a pas de représentation explicite de la concurrence.

Exemples: OBJ, Larch.

**d- Algèbre de processus** : Dans cette approche, une représentation explicite des processus concurrents est autorisée. Le comportement du système est représenté par les contraintes sur toutes les communications autorisées et observables entre processus.

Exemples: Communicating Sequential Processes (CSP), Lotos, Timed CSP.

**e- Approches basées graphes** : les notations graphiques sont très populaires pour la spécification des systèmes car elles sont faciles à comprendre et, par conséquent, plus accessibles aux non-

spécialistes. Cette approche combine des langages graphiques avec la sémantique formelle, ce qui apporte des avantages particuliers dans le développement de systèmes et de la réingénierie.

Exemples: Petri Nets, State Charts.

## 2.6. Techniques de vérification formelle

Classiquement, on distingue deux grandes familles de techniques pour vérifier formellement la correction d'un système. En premier lieu, les techniques de preuve (*theorem proving*) qui sont des démonstrations mathématiques au sens classique du terme où la vérification des propriétés est effectuée par déduction à partir d'un ensemble d'axiomes et de règles d'inférences. La seconde famille de techniques est appelée vérification de modèles, ou *model-checking*, et consiste à construire un modèle à partir d'une description formelle d'un système,

### 2.6.1. Vérification de modèles (Model-Checking)

La vérification de modèles est une technique qui consiste à construire un modèle fini d'un système et vérifier qu'une propriété cherchée est vraie dans ce modèle. Il y a deux façons générales de vérification dans le model-checking: vérifier qu'une propriété exprimée dans une logique temporelle est vraie dans le système, ou comparer (en utilisant une relation d'équivalence ou de préordre) le système avec une spécification pour vérifier si le système correspond à la spécification ou non. Au contraire du theorem proving, le model-checking est complètement automatique et rapide. Il produit aussi des contre-exemples qui représentent des erreurs subtiles dans la conception et ainsi il peut être utilisé pour aider le débogage.

### 2.6.2. Preuve de théorèmes (Theorem proving)

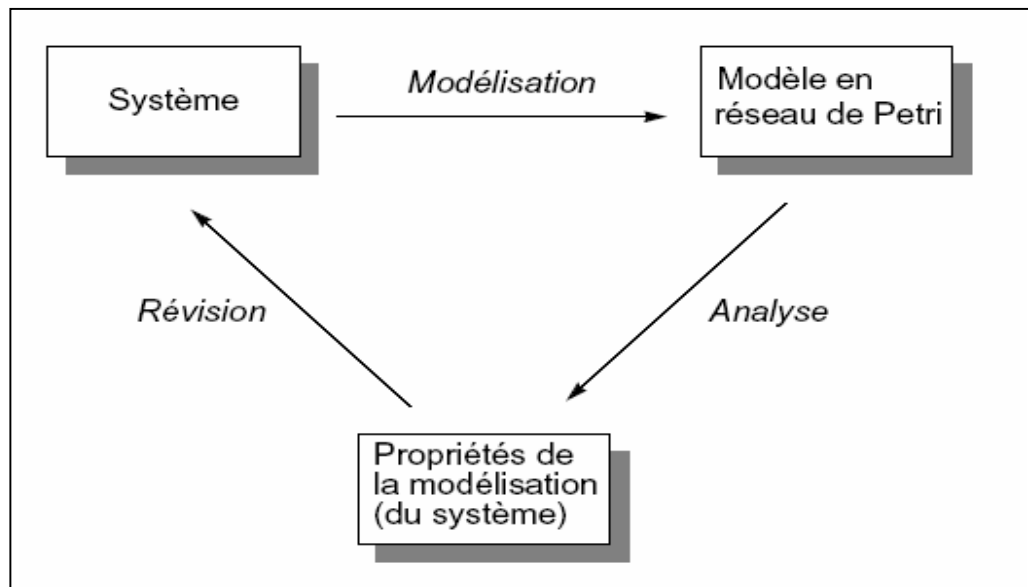
La preuve de théorèmes est une technique où le système et les propriétés recherchées sont exprimés comme des formules dans une logique mathématique. Cette logique est décrite par un système formel qui définit un ensemble d'axiomes et de règles de déduction. La preuve de théorèmes est le processus de recherche de la preuve d'une propriété à partir des axiomes du système. Les étapes pendant la preuve font appel aux axiomes et aux règles, ainsi qu'aux définitions et lemmes qui ont été possiblement dérivés.

Au contraire du model-checking, le theorem proving peut s'utiliser avec des espaces d'états infinis à l'aide de techniques comme l'induction structurelle. Son principal inconvénient est que

le processus de vérification est normalement lent, sujet à l'erreur, demande beaucoup de travail et des utilisateurs très spécialisés avec beaucoup d'expertise.

## 2.7. Réseaux de Petri

Le formalisme des réseaux de Petri (RdP) a été proposé par *Carl Adam Petri* en 1962 à l'université Darmstadt dans sa thèse intitulée : "Communication avec des automates" comme un outil mathématique permettant la modélisation des systèmes dynamiques à événements discrets. Ce formalisme bénéficie d'une riche panoplie de résultats théoriques, de techniques d'analyse et d'outils. L'analyse d'un réseau de Petri peut révéler des caractéristiques importantes du système concernant sa structure et son comportement dynamique. Les résultats de cette analyse sont utilisés pour évaluer le système et en permettre la modification ou l'amélioration. La figure 2.1 montre la méthode générale basée sur le formalisme des réseaux de Petri pour la modélisation et l'analyse des systèmes.



**Figure 2.1:** Méthode générale de modélisation et d'analyse basée sur les réseaux de Petri

Les réseaux de Petri sont utilisés dans une grande variété de domaines tels que les protocoles de communication, les systèmes distribués, l'architecture des ordinateurs, etc ...



Avant de rappeler les notions de base des réseaux de Petri, rappelons tout d'abord ce que sont ces systèmes à événements discrets ("*Discrete Event Dynamic Systems*", DEDS).

### 2.7.1. Systèmes à événements discrets [Valette02]

#### 2.7.1.1. Les types de variables

En général lorsqu'on construit un modèle mathématique d'un système, on classe les variables en deux catégories :

- Les *variables continues*: sont des variables qui prennent leurs valeurs sur le domaine des réels.
- Les *variables discrètes* : sont des variables qui prennent leurs valeurs sur un domaine dénombrable comme l'ensemble des entiers naturels, ou bien sur des ensembles dont le nombre d'éléments est fini.

A partir de cette classification, les systèmes dynamiques peuvent être classés dans diverses catégories selon les modèles mathématiques qui sont utilisés pour les représenter.

#### 2.7.1.2 Types de systèmes

##### a) Les systèmes continus

Le modèle d'un système continu ne comprend que des variables continues. De plus, le temps est également une variable continue. Quand le temps est représenté par une variable continue, on dit qu'il s'agit d'un *temps dense*. C'est le grand domaine des systèmes représentés par des équations algébro-différentielles.

##### b) Les systèmes échantillonnés

Ce sont ceux pour lesquels on a choisi d'avoir une représentation discrète du temps associée à des variables d'état continues. Le temps est représenté comme une suite infinie d'instantanés repérés par des entiers naturels  $E_1, E_2, \dots$

##### c) Les systèmes discrets

Ce sont ceux pour lesquels, on a choisi une représentation continue du temps (temps dense) associée à une représentation discrète de l'état. Cela peut être le cas lorsque l'état est naturellement discret. Par exemple une machine est allumée ou éteinte, etc.

##### d) Les systèmes à événements discrets

Ce sont ceux pour lesquels on ne conserve de la représentation du système qu'une suite d'événements discrets. Le temps n'est codé que par la suite des événements. Il est donc discrétisé.

Les événements sont discrets car ils correspondent à des changements d'états entre des états discrets du système. Temps et états sont donc discrétisés. Les événements ne peuvent pas être répartis également dans le temps comme dans le cas des systèmes échantillonnés. En effet, ils correspondent à des passages d'un état qualitatif du système à un autre. La durée, vue d'un point de vue continu, s'écoulant entre deux événements successifs, dépend de la dynamique du système.

#### e) Les systèmes hybrides

Ce sont ceux qui comprennent à la fois des variables d'état continues et des variables d'état discrètes. On peut aussi à la fois manipuler du temps dense, codé par des réels et du temps discret sous la forme d'un ensemble d'événements. C'est donc une classe de systèmes qui comprend les quatre classes précédentes.

Dans ce qui suit on présentera une brève introduction aux réseaux de Petri. Un survol sur les extensions de ces derniers, et précisément les ECATNets, objets de notre travail, sera également fait.

### 2.7.2 Concepts de base des réseaux de Petri ordinaires

Nous allons commencer par quelques définitions informelles.

#### 2.7.2.1. Définitions informelles

##### **Condition**

Une condition est un prédicat ou une description logique d'un état du système. Une condition est vraie ou fausse. Un état du système peut être décrit comme un ensemble de conditions.

##### **Événement**

Les événements sont des actions se déroulant dans le système. Le déclenchement d'un événement dépend de l'état du système.

##### **Déclenchement, pré-condition, post-condition**

Les conditions nécessaires au déclenchement d'un événement sont les *préconditions* de l'événement. Lorsqu'un événement se produit, certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres conditions, appelées *postconditions* de l'événement, deviennent vraies.

Intuitivement, un réseau de Petri est un graphe orienté biparti (ayant deux types de nœuds) : des *places* représentées par des *cercles* et des *transitions* représentées par des *rectangles*. Les arcs du graphe ne peuvent relier que des places vers des transitions, ou des transitions vers des places (pas d'arcs entre places ni entre transitions). Un exemple de réseau de Petri est illustré par la figure 2.2.

Un réseau de Petri décrit un *système dynamique à événements discrets*. Les places permettent la description des états possibles du système (qui sont discrets), les transitions permettent la description des événements ou les actions qui causent le changement de l'état.

Un réseau de Petri est un graphe muni d'une *sémantique opérationnelle*, c'est-à-dire qu'un comportement est associé au graphe, ce qui permet de décrire la dynamique du système représenté. Pour cela un troisième élément est ajouté aux places et aux transitions : *les jetons*. Une répartition des jetons dans les places à un instant donné est appelée *marquage* du réseau de Petri. Un marquage donne *l'état* du système. Le nombre de jetons contenus dans une place est un *entier positif ou nul*. Il ne peut pas être négatif.

Pour un marquage donné, une transition peut être *sensibilisée* ou non. Une transition est sensibilisée si chacune de ses places d'entrée contient au moins un jeton. L'ensemble des transitions sensibilisées pour un marquage donné définit l'ensemble des changements d'états possibles du système depuis l'état correspondant à ce marquage. C'est un moyen de définir l'ensemble des événements auxquels ce système est *réceptif* dans cet état.

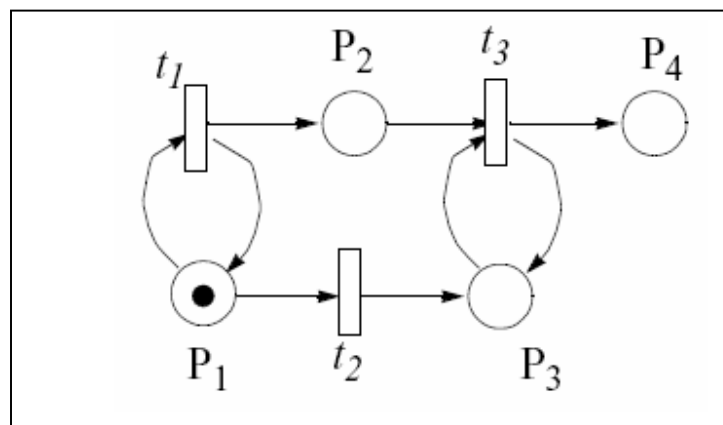


Figure 2.2 : Exemple de réseau de Petri marqué.

### 2.7.2.2. Définitions formelles

Formellement, un réseau de Petri ( $R$ ) est un triplet  $R = (P, T, W)$  où  $P$  est l'ensemble des places (les places représentent les conditions) et  $T$  l'ensemble des transitions (les transitions représentent les événements ou les actions) tel que  $P \cap T = \emptyset$  et  $W$  est la fonction définissant le poids porté par les arcs où  $W : ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N} = \{0, 1, 2, \dots\}$ .

Le réseau  $R$  est fini si l'ensemble des places et des transitions est fini, c-à-d  $|P \cup T| \in \mathbb{N}$ .

Un réseau  $R = (P, T, W)$  est ordinaire si pour tout  $(x, y) \in ((P \times T) \cup (T \times P))$ :  $W(x, y) \leq 1$ . Dans un réseau ordinaire la fonction  $W$  est remplacée par  $F$  où :

$F \subseteq ((P \times T) \cup (T \times P))$  tel que  $(x, y) \in F \Leftrightarrow W(x, y) \neq 0$ .

Pour chaque  $x \in P \cup T$  :

${}^*x$  représente l'ensemble des entrées de  $x$  :  ${}^*x = \{y \in P \cup T \mid W(y, x) \neq 0\}$

$x^*$  représente l'ensemble des sorties de  $x$  :  $x^* = \{y \in P \cup T \mid W(x, y) \neq 0\}$

#### Remarque

si  ${}^*x = \emptyset$ ,  $x$  est dite source, si  $x^* = \emptyset$ ,  $x$  est dite puits.

Le *comportement* d'un réseau de Petri est déterminé par sa structure et par son état. Pour exprimer l'état d'un réseau de Petri, les places peuvent contenir des jetons qui ne sont que de simples marques.

#### Notion d'état dans un réseau de Petri

Dans la théorie des réseaux de Petri, l'état d'un réseau est souvent appelé *marquage* du réseau. Il est défini par la distribution des jetons sur les places. Le marquage d'un réseau de Petri  $R = (P, T, W)$  est défini par la fonction de marquage  $M : P \rightarrow \mathbb{N}$ .

Un réseau de Petri marqué est donné par  $\Sigma = (P, T, W, M_0)$  où  $M_0$  est le marquage initial. Le comportement d'un réseau de Petri marqué est défini par une *règle de franchissement*.

#### Notion de changement d'état de règle de franchissement

Un *changement d'état* est dénoté par un mouvement de jetons (points noirs) de places d'entrée vers des places de sortie d'une transition. Ce mouvement est causé par le franchissement d'une transition. Le franchissement représente une occurrence d'un événement ou d'une action.

Le franchissement est conditionné par la présence de jetons dans les places d'entrée de la transition.

Quand toutes les *places d'entrée* à une transition sont suffisamment *marquées*, la transition peut *tirer* et alors les jetons sont retirés des places d'entrée (ancien état) et ajoutés à toutes les *places de sortie* (Nouvel état).

Une *règle de franchissement* est une simple relation de transition qui définit le changement d'état dans un réseau marqué lors de l'exécution d'une action. Afin de définir une règle de franchissement, il est nécessaire de formaliser quand le réseau peut exécuter une action: on dit qu'une transition  $t \in T$  peut être franchie à partir d'un marquage  $M$  (qui représente l'état du system à un instant donné) si et seulement si chaque place d'entrée  $p \in {}^*t$  de la transition  $t$  contient au moins un nombre de jetons qui est supérieur ou égale au poids de l'arc reliant cette place d'entrée  $p$  avec la transition  $t$  tel que:  $M(p) \geq W(p, t) \quad \forall p \in P$ .

Une règle de franchissement est définie par  $M'(p) = M(p) - W(p, t) + W(t, p)$  pour tout  $p \in P$ , ce qui veut dire que lorsque la transition  $t$  est franchie à partir d'un marquage  $M$ , il faut saisir  $W(p, t)$  jetons à partir de chaque place d'entrée à la transition  $t$  et déposer  $W(t, p)$  jetons dans chaque place de sortie de la transition  $t$  ce qui permet de produire un nouveau marquage  $M'$ .

Le franchissement d'une transition  $t$ , dénoté par  $M[t \rangle M'$ , est dit l'occurrence de  $t$ . On dit que deux transitions  $t_1, t_2$  (pas certainement distinctes) sont franchies en concurrence par un marquage  $M$  si et seulement si  $M(p) \geq W(p, t_1) + W(p, t_2)$  pour toute  $p \in P$ .

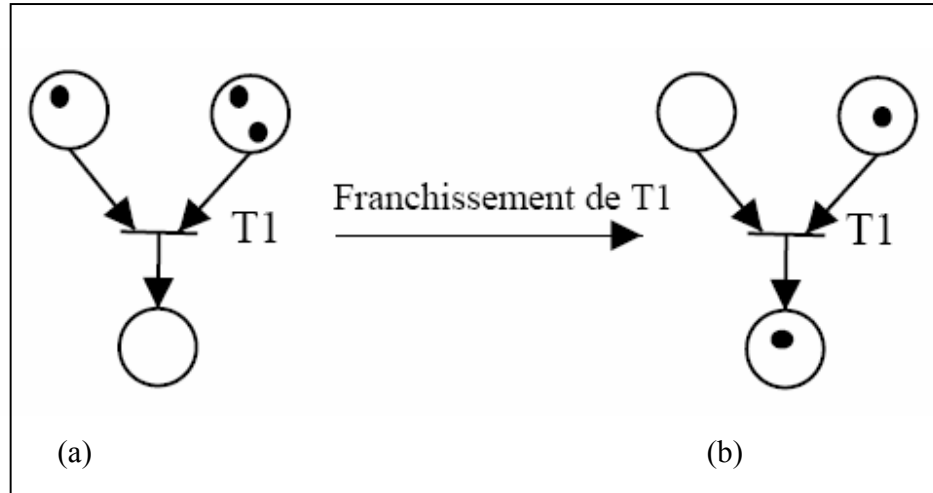
Cette vision de l'exécution concurrente de deux transitions dans un RdP est contradictoire avec celle qui impose que deux occurrence de transition sont parallèles si et seulement si : elles sont causalement indépendantes et n'ont pas une relation de conflit entre elles. Deux occurrences sont en conflit si l'une des deux peut avoir lieu mais pas toutes les deux.

### 2.7.3 Exécution d'un réseau de Petri : Notion de marquage

L'exécution d'un réseau de Petri est définie par un ensemble de séquences d'occurrence. Une séquence d'occurrence est une séquence de transitions franchissables dénotée par  $\sigma = M_0 t_1 M_1 t_2 \dots$  tel que  $M_{i-1} [t_i \rangle M_i$ . Une séquence  $t_1 t_2 \dots$  est une séquence de transitions (commencée par le marquage  $M$ ) si et seulement si il existe une séquence d'occurrence  $M_0 t_1 M_1 \dots$  avec  $M = M_0$ . Si la séquence finie  $t_1 t_2 \dots t_n$  conduit à un nouveau marquage  $M'$  à partir du marquage  $M$ , on écrit  $M[t_1 t_2 \dots t_n \rangle M'$  ou simplement  $M[t_1 t_2 \dots t_n \rangle$  si on ne veut pas spécifier le marquage résultat.

L'ensemble de marquages accessibles d'un réseau marqué  $(P, T, W, M_0)$  est défini par  $[M_0] = \{M \mid \exists t_1 t_2 \dots t_n: M_0 [t_1 t_2 \dots t_n] M\}$ .

**Exemple**



**Figure 2.3 :** (a) RdP marqué avant franchissement de T1 (b) RdP après franchissement de T1.

**Représentation matricielle**

Une représentation matricielle d'un RdP est offerte afin de simplifier les tâches d'analyse et de vérification effectuées sur un modèle RdP. Agir sur une représentation graphique d'un modèle RdP est une tâche délicate en la comparant avec une représentation matricielle.

Il est possible de représenter la fonction  $W$  (fonction de poids) par des matrices.

**Définition**

Soit Un réseau de Petri  $R = (P, T, W)$  avec  $P = \{p_1, p_2, \dots, p_m\}$  et  $T = \{t_1, t_2, \dots, t_n\}$ . On appelle matrice des pré-conditions *pré*, la matrice  $m \times n$  à coefficients dans  $\mathbb{N}$  telle que  $pré(i,j) = W(p_i, t_j)$  indique le nombre de marques que doit contenir la place  $p_i$  pour que la transition  $t_j$  devienne franchissable. De la même manière on définit la matrice des post-conditions *post*, la matrice  $n \times m$  telle que  $post(i,j) = W(t_j, p_i)$  contient le nombre de marques déposées dans  $p_i$  lors du franchissement de la transition  $t_j$ . La matrice  $C = post - pré$  est appelée matrice d'incidence du réseau ( $m$  représente le nombre de places d'un réseau de Petri et  $n$  le nombre de transitions.)

Le marquage d'un réseau de Petri est représenté par un vecteur de dimension  $m$  à coefficients dans  $\mathbb{N}$ . La règle de franchissement d'un réseau de Petri est définie par :

$$M'(p) = M(p) + C(p, t).$$

### Exemple

Pour le réseau de la figure 2.2 :

$$P = \{p_1, p_2, p_3, p_4\} \quad T = \{t_1, t_2, t_3\}$$

|   |   |
|---|---|
| $Pré = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$  | $Post = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ |
| <p><b>La matrice d'incidence C est :</b></p>  | <p><b>Le vecteur de marquage M est :</b></p>  |
| $C = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ | $M = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$                                    |

Figure 2.4: Matrice d'incidence et vecteur de marquage du RdP de la figure 2.2.

## 2.7.4. Propriétés des Réseaux de Petri Places/Transitions

### 2.7.4.1. Séquences de franchissement [Murata89]

Une séquence de franchissement "s" est une suite de transitions  $\langle t_1, t_2, \dots, t_n \rangle$  qui permet, à partir d'un marquage «M», de passer au marquage «M'» par le franchissement successif des transitions définissant la séquence.

### 2.7.4.2. Propriétés des séquences de franchissement

- **Existence d'un marquage**

Pour toute séquence de transitions "s", il existe un marquage M tel que celle-ci soit franchissable.

- **Monotonie**

L'augmentation de jetons dans les places d'un marquage préserve la possibilité de franchissement d'une séquence de transitions.

- **Séquence répétitive**

Une séquence de transitions est dite répétitive si pour tout marquage M tel que :

$$M[s^> \text{ alors } M[s^*>$$

La notion de séquence répétitive permet de définir une condition nécessaire et suffisante pour qu'un réseau marqué ait la possibilité d'être infiniment actif.

### 2.7.4.3. Propriétés comportementales

Les propriétés comportementales dépendent du marquage initial. Si l'on change ce marquage, rien ne garantit (sauf dans certains cas précis) que les propriétés tiennent encore.

- **Caractère borné**

Cette propriété définit et caractérise la possibilité pour une place d'accumuler une quantité bornée ou pas de jetons au cours de l'évolution d'un réseau.

- **Place k-bornée, non bornée**

Pour un réseau  $R$  et un marquage  $M_0$ , une place «  $p$  » du réseau marqué  $(R, M_0)$  est  $k$ -bornée si pour tout marquage  $M$  accessible depuis  $M_0$ ,  $M(p) \leq k$ . Dans le cas contraire la place " $p$ " est dite non-bornée.

- **Réseau borné**

Un réseau marqué est borné si toutes ses places sont bornées. Les réseaux 1-bornés sont appelés des réseaux "saufs".

### Exemple [Murata89]

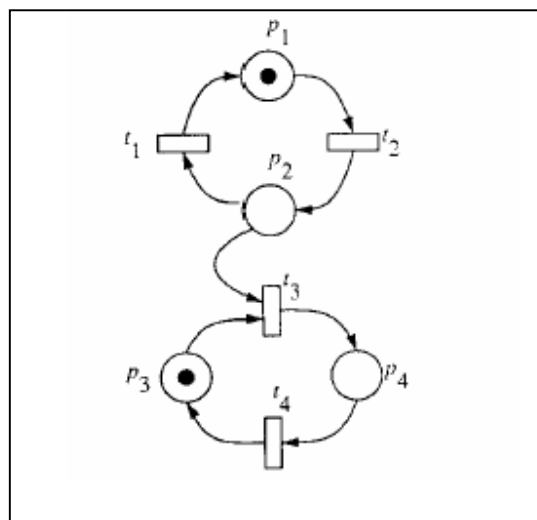


Figure 2.5 : RdP borné.



- **Activité d'un réseau**

La notion d'activité d'un réseau recouvre deux classes de définitions. La première concerne l'activité individuelle des transitions, la seconde concerne l'activité globale d'un réseau (Indépendamment de transitions particulières).

- **Pseudo-vivacité**

Un réseau de Petri  $(R, Mo)$  est dit pseudo-vivant si pour tout marquage accessible depuis le marquage initial, il existe toujours une transition "t" qui puisse être franchie.

- **Quasi-vivacité d'une transition**

La quasi-vivacité d'une transition signifie que depuis le marquage initial, cette transition peut être franchie au moins une fois. Par conséquent, une transition qui n'est pas quasi-vivante est inutile.

- **Quasi-vivacité d'un réseau**

Un réseau est quasi-vivant si toutes ses transitions le sont.

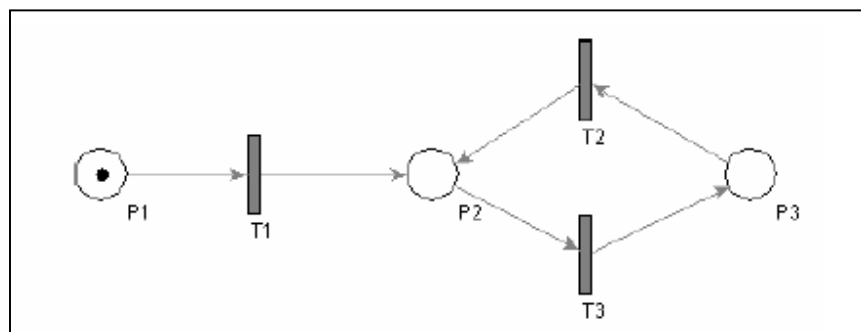


Figure 2.6 : RdP quasi-vivant, pseudo-vivant

Ce RdP est quasi-vivant car T1 est tirable au moins une fois. Il est pseudo-vivant car il existe toujours une transition tirable (T2 ou T3).

- **Monotonie et quasi-vivacité**

La propriété de monotonie présentée plus haut, implique qu'une transition quasi-vivante pour  $(R, M)$  le reste pour  $(R, M')$  où  $M \leq M'$ .

- **Vivacité**

Les deux propriétés précédentes assurent une certaine correction du système mais elles ne permettent pas d'affirmer que, dans n'importe quel état atteint, le système dispose encore de toutes ses fonctionnalités. Autrement dit, si toute transition peut toujours être ultérieurement

franchie à partir d'un état quelconque du système [Diaz]. Par exemple, dans un réseau quasi-vivant une transition pourra être franchie une seule fois.

- **Vivacité d'une transition**

La vivacité d'une transition exprime le fait que quelque soit l'évolution du réseau à partir du marquage initial, le franchissement à terme de cette transition est toujours possible.

- **Vivacité d'un réseau**

Un réseau est vivant si toutes ses transitions le sont. Autrement dit, le réseau de Petri  $(R, Mo)$  est vivant si, pour tout marquage  $M \in A(R, Mo)$ , le réseau  $(R, M)$  est quasi-vivant :

$$\forall M \in A(R, Mo), \forall t \in T, \exists M' \in A(R, M) \text{ tel que } M' [t]$$

Dans un tel réseau, on garantit que chaque transition soit éventuellement tirable peu importe l'état du système.

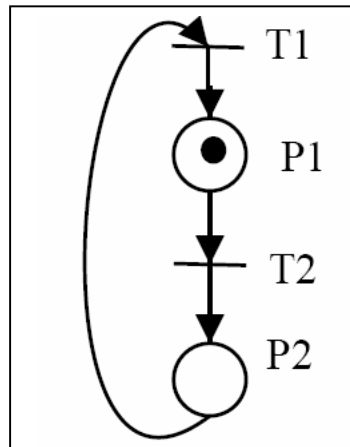


Figure 2.7: RdP vivant.

Ce RdP est vivant car il est toujours possible de tirer T1 ou T2 mais le RdP de la figure 2.6 n'est pas vivant car à un certain moment, il n'est plus possible de tirer T1.

- **État d'accueil**

Un RdP possède un état d'accueil  $M_a$  pour un marquage initial  $Mo$  si pour tout marquage accessible  $M_i$  il existe une séquence "s" telle que  $M_i [s] M_a$ .

- **RdP réversible**

Un RdP est réversible pour un marquage initial  $Mo$  si  $Mo$  est un état d'accueil.

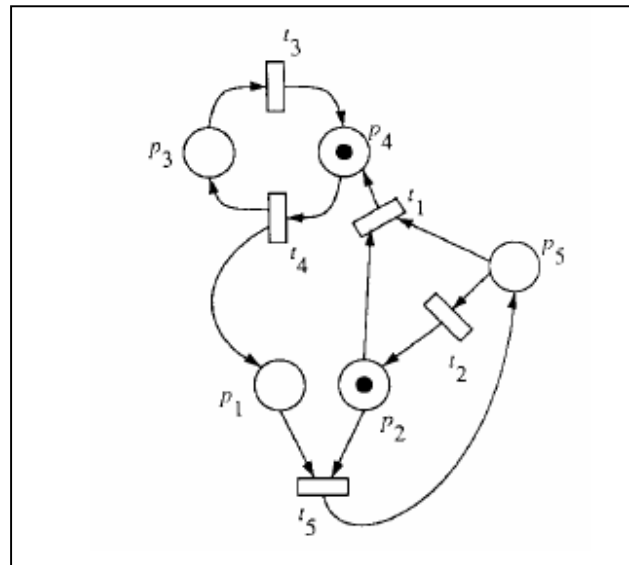


Figure 2.8: RdP réversible

- **Absence de blocage**

Cette propriété est plus faible que celle de vivacité. Elle implique seulement que le réseau a toujours la possibilité d'évoluer.

- **Marquage puit**

Un marquage puit est un marquage à partir duquel aucune transition n'est tirable. Un réseau marqué est sans blocage si aucun de ses marquages accessibles n'est un marquage puit.

### 2.7.5 Méthodes d'analyse des réseaux de Petri

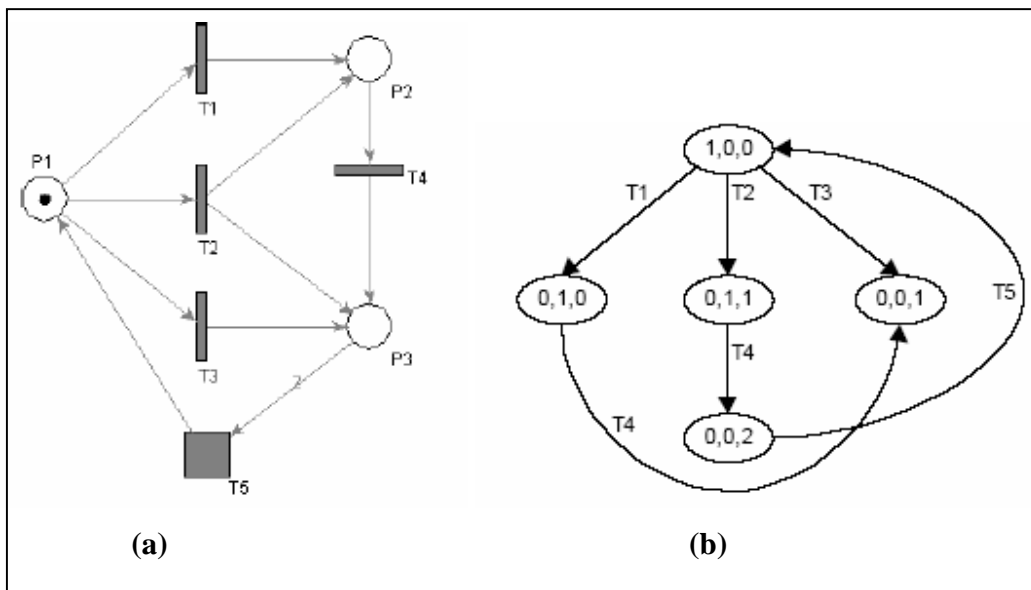
La modélisation d'un système n'est utile que si elle permet d'analyser ses propriétés. La théorie des réseaux de Petri offre des techniques d'analyse puissantes pour valider des modèles de comportement de systèmes à événements discrets (analyser les propriétés d'un réseau de Petri). Parmi ces techniques nous pouvons citer l'analyse grâce aux graphes des marquages, l'analyse par algèbre linéaire, l'analyse structurelle, et l'analyse par réduction.

- **Analyse par graphes des marquages**

L'idée la plus naturelle pour étudier les propriétés d'un RdP est de construire le graphe de tous ses marquages accessibles. Le graphe des marquages accessibles est un graphe dont chaque sommet correspond à un marquage accessible et dont chaque arc correspond au franchissement d'une transition permettant de passer d'un marquage à l'autre. Deux situations peuvent alors se présenter :

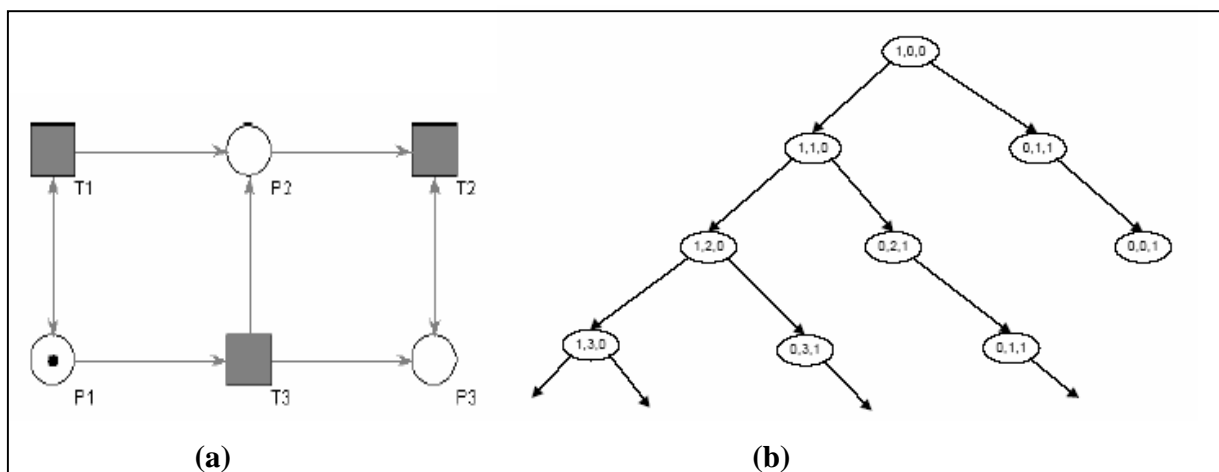
1. *Le graphe est fini.* C'est la situation la plus favorable car dans ce cas toutes les propriétés peuvent être déduites simplement par inspection de celui-ci. Nous avons déjà vu plusieurs exemples de cette utilisation.
2. *Le graphe est infini.* Dans ce cas, on construit un autre graphe appelé "graphe de couverture" permettant de déduire certaines propriétés.

**Exemple 1**



**Figure 2.9 :** (a) Réseau de Petri (b) Graphe des marquages fini

**Exemple 2**



**Figure 2.10 :** (a) Réseau de Petri (b) Graphe des marquages infini

- **Analyse par algèbre linéaire**

L'analyse par algèbre linéaire permet d'étudier des propriétés d'un réseau (caractère borné, vivacité) indépendamment d'un marquage initial. De ce fait, on parlera de propriétés structurelles du réseau. Par exemple, on pourra dire qu'un réseau est *structurellement borné* s'il est borné pour tout marquage initial fini. De la même façon, si pour tout marquage initial, le réseau est vivant, on dira que le réseau est structurellement vivant.

- **Analyse par réduction**

Pour l'analyse des propriétés d'un RdP, des difficultés peuvent apparaître dans l'utilisation du graphe de marquages, et même dans l'algèbre linéaire dans le cas d'un RdP de taille significative. L'objectif de la réduction est de présenter des règles permettant d'obtenir à partir d'un RdP marqué, un RdP marqué plus simple, c'est-à-dire avec un nombre réduit de places et un nombre réduit de transitions (règles de réduction). Ce dernier doit être :

1. équivalent au RdP marqué de départ (pour les propriétés étudiées).
2. suffisamment simple pour que l'analyse de ses propriétés soit simple.

En général, le RdP simplifié ne peut pas s'interpréter comme le modèle d'un système.

On peut distinguer deux types de règles de réduction:

1. règles de réduction préservant la vivacité et la bornétude (et leurs propriétés associées),
2. règles de réduction préservant les invariants de marquage. Pour plus de détail, voir [Murata89].

### **2.7.6. Outils d'analyse des RdP**

La puissance des réseaux de Petri réside dans l'existence d'une batterie d'outils d'analyse tels que INA ( Integrated Net Analyzer) [INA], PEP (Programming Environment based on Petri nets) [Pep], TINA [Tina], etc ...

#### **INA (Integrated Net Analyser)**

L'INA est un outil d'analyse largement utilisé par la communauté des réseaux de Petri. Il a été développé par le prof. Dr. Peter H. Starke [INA] [Roch02]. C'est un programme interactif dirigé par les menus qui permet à l'utilisateur d'éditer (dans une forme textuelle), de réduire, d'exécuter et d'analyser les modèles des RdPs.

L'exemple que nous présentons concerne trois programmeurs se partageant deux terminaux. Chaque programmeur travaille sur un terminal ou attend. Le programmeur 1 a besoin de deux terminaux en même temps pour pouvoir travailler (figure 2.11).

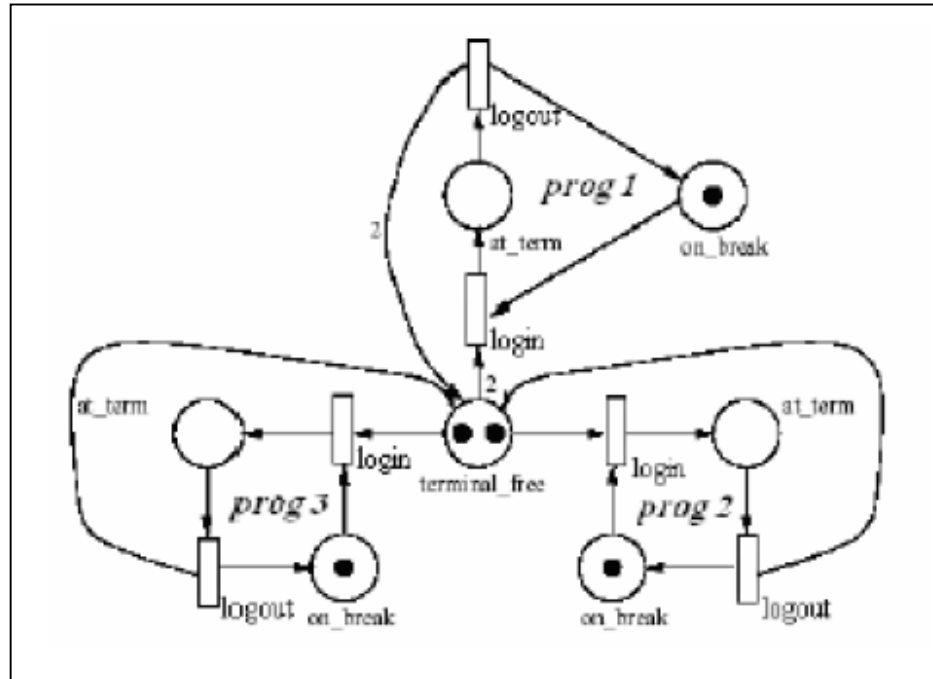


Figure 2.11 : Trois programmeurs partageant deux terminaux [Roch02]

Pour analyser un réseau de Perti en utilisant INA, il doit être transformé vers la spécification INA équivalente. C.à.d passer d'une représentation graphique vers une description textuelle. La figure 2.12 montre la description textuelle du réseau de Petri de la figure 2.11.

| P | M | PRE,POST           | NETZ | 1:3_prog_2_term |
|---|---|--------------------|------|-----------------|
| 0 | 2 | 4: 2 5 6, 1: 2 2 3 |      |                 |
| 1 | 0 | 1, 4               |      |                 |
| 2 | 0 | 2, 5               |      |                 |
| 3 | 0 | 3, 6               |      |                 |
| 4 | 1 | 4, 1               |      |                 |
| 5 | 1 | 5, 2               |      |                 |
| 6 | 1 | 6, 3               |      |                 |

| place nr. | name           | capacity | time |
|-----------|----------------|----------|------|
| 0:        | terminal_free  | 00       | 0    |
| 1:        | prog1_at_term  | 00       | 0    |
| 2:        | prog2_at_term  | 00       | 0    |
| 3:        | prog3_at_term  | 00       | 0    |
| 4:        | prog1_on_break | 00       | 0    |
| 5:        | prog2_on_break | 00       | 0    |
| 6:        | prog3_on_break | 00       | 0    |

| trans nr. | name         | priority | time |
|-----------|--------------|----------|------|
| 1:        | login_prog1  | 0        | 0    |
| 2:        | login_prog2  | 0        | 0    |
| 3:        | login_prog3  | 0        | 0    |
| 4:        | logout_prog1 | 0        | 0    |
| 5:        | logout_prog2 | 0        | 0    |
| 6:        | logout_prog3 | 0        | 0    |

Figure 2.12 : La spécification INA de l'exemple de la figure 2 .11

## 2.8 Extensions des réseaux de Petri

La modélisation d'un système réel peut mener à des réseaux de Petri de taille trop importante rendant leur manipulation et/ou leur analyse difficile. La question est alors de modifier (étendre) la modélisation par RdP de façon à obtenir des modèles RdP de plus petite taille. Cette question a motivé la nécessité de faire des extensions des réseaux de Petri. En plus de ça, les points suivants ont motivé cette extension:

- Certaines propriétés ne peuvent pas être exprimées à l'aide des réseaux usuels.
- Besoin d'avoir une information plus précise sur les jetons transitant dans le réseau.

Ceci a donné naissance à plusieurs extensions de réseaux de Petri tels que les réseaux de Petri colorés [Jensen 97, Jensen98], Les ECATNets [Bettaz92, Bettaz93], les réseaux de Petri algébriques, les réseaux de Petri Predicate/Transition [Genrich81], etc. Nous donnerons dans la section suivante un aperçu sur les ECATNets qui seront utilisés dans le cadre de cette thèse.

## 2.9. Présentation des ECATNets

Les ECATNets sont une catégorie de modèle de réseau/données combinant les forces des réseaux de Petri avec ceux des types abstraits de données. Les places sont identifiées par des multi-ensembles des termes algébriques. Les arcs entrants dans chaque transition  $t$ , c.-à-d.  $(p, t)$ , sont marqués par deux inscriptions  $IC(p, t)$  (Input Conditions) et  $DT(p, t)$  (Destroyed Tokens). Les arcs sortants de chaque transition  $t$ , c.-à-d.  $(t, p')$ , sont marqués par  $CT(t, p')$  (Created Tokens). Finalement, chaque transition  $t$  est marquée par une inscription  $TC(t)$  (Transition Condition) (voir la figure 3.3).

- $IC(p, t)$  indique une condition qui valide une transition  $t$ .
- $DT(p, t)$  indique le marquage (un multi-ensemble) qui doit être enlevé de  $p$  lors du franchissement de  $t$ .
- $CT(t, p')$  indique le marquage (un multi-ensemble) qui doit être ajouté à  $p'$  lors du franchissement de  $t$ .
- $TC(t)$  représente un terme booléen qui indique une condition supplémentaire pour le franchissement de la transition  $t$ .

L'état courant d'un ECATNets est donné par l'union des termes ayant la forme  $(p, M(p))$ . Par exemple, soit un réseau contenant une transition  $t$  ayant une place d'entrée  $p$  marquée par le multi-ensemble  $a \oplus b \oplus c$  et une place de sortie vide  $p'$ , l'état distribué  $s$  de ce réseau est donné par le multi-ensemble :  $s = (p, a \oplus b \oplus c)$ .

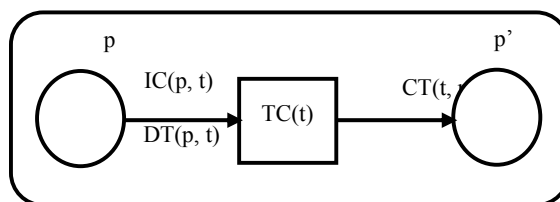


Figure 2.13: Un ECATNets générique.

### Franchissement d'une transition

Une transition  $t$  est franchissable quand les conditions suivantes sont vraies :

- ✓ Chaque  $IC(p, t)$  pour chaque place d'entrée  $p$  est tirable.
- ✓  $TC(t)$  est vraie. Enfin l'addition de  $CT(t, p')$  à chaque place de sortie  $p'$  ne doit pas avoir comme conséquence d'excéder sa capacité quand cette capacité est finie.



Le franchissement de  $t$  va mener à :

- ✓  $DT(p, t)$  est complètement enlevé (cas positif) de la place d'entrée  $p$
- ✓  $CT(t, p')$  est ajouté à la place de sortie  $p'$ .

Notons que dans le cas non positif, on enlève les éléments en commun entre  $DT(p, t)$  et  $M(p)$ .

D'un point de vue syntaxique, la différence entre un ECATNets simplifié et un réseau de Petri ordinaire est que les jetons dans les premiers ne sont pas imperceptibles car ils sont exprimés en matière de termes algébriques possédant une identité, une syntaxe et une sémantique. D'ici les jetons peuvent porter des informations complexes grâce à leur nature en tant que termes algébriques. Du point de vue sémantique, Le franchissement d'une transition et les conditions de ce franchissement sont formellement exprimés par des règles de la logique de réécriture [Meseguer96], [Meseguer00], où Les axiomes sont en réalité des règles de réécriture conditionnelles décrivant les effets des transitions comme des types élémentaires de changement. Les règles de déduction nous permettent d'avoir des conclusions valides des ECATNets à partir des changements.

La logique de réécriture offre aux ECATNets une version textuelle simple, intuitive et pratique pour leur analyse sans perdre leur sémantique formelle (rigueur mathématique et raisonnement formel). Plusieurs travaux ont été réalisés dans le contexte de l'implémentation des ECATNets dans un système basé sur cette logique. L'implémentation dans le système Maude en est un exemple [Bettaz93]. Maude fournit une plate-forme permettant un développement facile et une exécution efficace des outils des ECATNets.

### 2.9.1 Formes des Règles de Réécriture [Bettaz92]

- **Cas positif sans Arcs Inhibiteurs**

$IC(p,t)$  est de la forme  $[m]_{\oplus}$

Cas 1.  $[IC(p, t)]_{\oplus} = [DT(p, t)]_{\oplus}$

La forme de cette règle est :  $t : (p, [IC(p, t)]_{\oplus}) \rightarrow (p', [CT(t, p')]_{\oplus})$

Cas 2  $[IC(p, t)]_{\oplus} \cap [DT(p, t)]_{\oplus} = \phi_M$

Cette situation correspond à vérifier que  $IC(p, t)$  est inclus dans  $M(p)$  et, retirer  $DT(p,t)$  de  $M(p)$ . La forme de cette règle est :  $t : (p, [IC(p, t)]_{\oplus}) \otimes (p, [DT(p, t)]_{\oplus}) \rightarrow (p, [IC(p, t)]_{\oplus}) \otimes (p', [CT(t, p')]_{\oplus})$ .

Cas 3  $[IC(p, t)] \cap [DT(p, t)] \neq \phi_M$

Cette situation correspond au cas le plus général. Elle peut cependant être résolue d'une manière élégante en remarquant qu'elle pourrait être apportée aux deux cas précédents. Ceci est réalisé en remplaçant la transition  $t$  de ce cas par deux transitions  $t1$  et  $t2$ . Ces transitions, une fois franchies concurremment, donnent le même effet global que la transition  $t$ .  $IC(p, t)$  est éclaté en deux multi-ensembles  $IC1(p, t1)$  et  $IC1(p, t2)$ . De même,  $DT(p, t)$  sera éclaté en deux multi-ensembles  $DT1(p, t1)$  et  $DT1(p, t2)$  :

$$IC(p, t) = IC1(p, t1) \cup IC1(p, t2), DT(p, t) = DT1(p, t1) \cup DT1(p, t2).$$

Les quatre multi-ensembles obtenus doivent réaliser  $IC1(p, t1) = DT1(p, t1)$  et  $IC2(p, t2) \cap DT2(p, t2) = \phi_M$ .

Le franchissement de la transition  $t$  est identique au franchissement en parallèle des deux transitions  $t1$  et  $t2$ .

- **Cas Général**

$IC(p,t)$  est de la forme  $[m]_{\oplus}$

Cas 1  $[IC(p,t)]_{\oplus} = [DT(p,t)]_{\oplus}$

La forme de cette règle est :  $t : (p, [IC(p, t)]_{\oplus}) \rightarrow (p', [CT(t, p')]_{\oplus})$

Cas 2  $[IC(p, t)]_{\oplus} \cap [DT(p, t)]_{\oplus} = \phi_M$

Cette situation est équivalente à celle de vérifier que  $IC(p, t)$  est inclus dans  $M(p)$  et retirer  $DT(p, t)$  de  $M(p)$ . La forme de cette règle est :  $t : (p, [IC(p, t)]_{\oplus}) \otimes (p, [DT(p, t)]_{\oplus}) \rightarrow (p, [IC(p, t)]_{\oplus}) \otimes (p', [CT(t, p')]_{\oplus})$

Cas 3.  $[IC(p, t)]_{\oplus} \cap [DT(p, t)]_{\oplus} \neq \phi_M$

Ce cas peut être résolu aussi en combinant les deux cas précédents. Ceci peut se faire en remplaçant la transition de ce cas par deux transitions, une fois franchies concurremment, donnent le même effet global. En réalité, ce remplacement montre comment spécifier une situation donnée en deux niveaux d'abstraction.

**$IC(p, t)$  est de la forme  $\tilde{[m]}_{\oplus}$**

La forme de cette règle est:

$t : (p, [DT(p, t)]_{\oplus} \cap [M(p)]_{\oplus}) \rightarrow (p', [CT(t, p')]_{\oplus})$  if  $([IC(p, t)]_{\oplus} \setminus ([IC(p, t)]_{\oplus} \cap [M(p)]_{\oplus})) = \phi_M$   
 $\rightarrow [false]$

$IC(p, t) = empty$

Cette règle est de la forme :

$$t : (p, [DT(p, t)]_{\oplus} \cap [M(p)]_{\oplus}) \rightarrow (p', [CT(t, p')]_{\oplus}) \text{ if } [M(p)]_{\oplus} \rightarrow \phi_M$$

telle que la capacité  $C(p)$  est finie, la partie conditionnelle de règle de réécriture va inclure le composant suivant :

$$[CT(p, t)]_{\oplus} \oplus [M(p)]_{\oplus} \cap [C(p)]_{\oplus} \rightarrow [CT(p, t)]_{\oplus} \oplus [M(p)]_{\oplus} (\text{Cap})$$

Dans le cas où il y a une condition de transition  $TC(t)$ , celle-ci est représentée dans la logique de réécriture par une règle non conditionnelle.

### 2.9.2 ECATNets et Maude

Considérons la version textuelle des ECATNets dans Maude. Le module générique qui décrit des opérations de base d'un ECATNets est le suivant :

fmod GENERIC-ECATNET is

sorts Place Marking GenericTerm .

op mt : -> Marking .

op <\_;> : Place GenericTerm -> Marking .

op \_\_ : Marking Marking -> Marking [assoc comm id: mt] .

endfm

#### Exemple 1

La figure 2.14 présente un ECATNets du problème du routeur [Bettaz, 93].

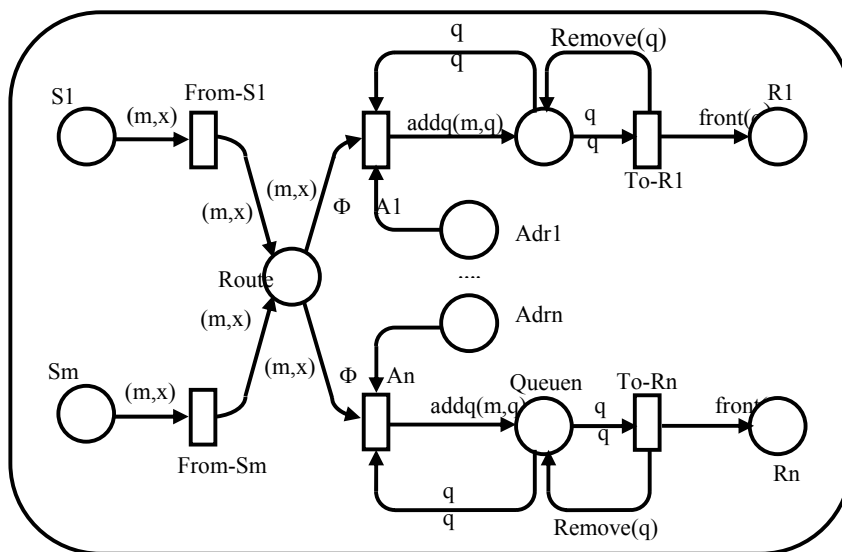


Figure 2.14 : ECATNets modélisant le problème du routeur.

Dans [Kerkouche08], les auteurs ont proposé un outil permettant d'éditer graphiquement un ECATNets puis le transformer automatiquement vers sa spécification Maude. Dans un premier temps, Ils ont utilisé leur outil pour éditer l'exemple du routeur (voir figure 2.15)

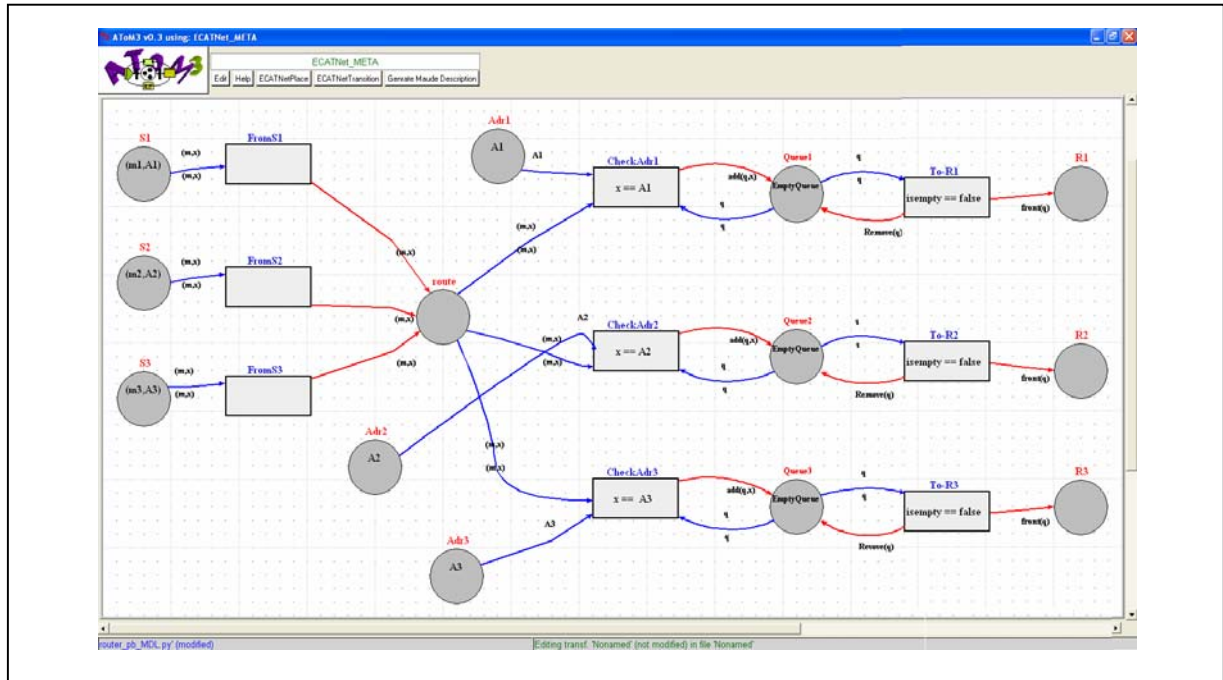


Figure 2.15: ECATNets du problème du routeur dans l'outil de transformation des ECATNets vers Maude

Puis ils ont appliqué leur outil de transformation automatique des ECATNets vers Maude sur cet exemple et ils ont obtenu la spécification Maude de la figure 2.16 :

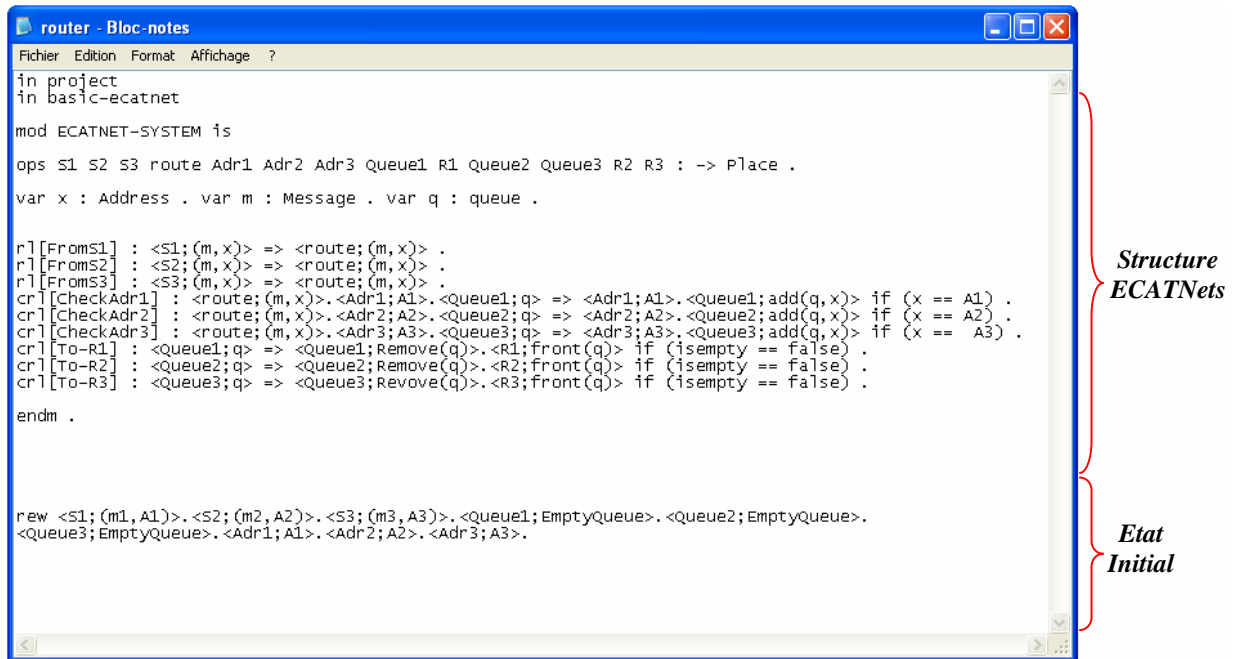


Figure 2.16 : Spécification Maude de l'exemple du routeur

### 2.9.3. Méthodes et outils d'analyse des ECATNets

La puissance des ECATNets réside dans leur sémantique basée sur la logique de réécriture et son langage de spécification Maude. Du moment qu'un ECATNets peut être transformé en une spécification Maude, le langage Maude peut être alors utilisé comme moyen pour l'analyse des systèmes modélisés par des ECATNets. Il existe maintenant un outil permettant d'éditer puis transformer automatiquement un ECATNets vers sa spécification Maude [Kerkouche08]. En plus de l'utilisation de Maude, beaucoup d'outils d'analyse des ECATNets ont été proposés. Nous pouvons citer ici l'analyse structurelle basée sur les verrous et les trappes [Bettaz96], la technique d'analyse par graphe de couverture [Boudiaf04] et la technique de réduction des ECATNets [Boudiaf 06].

### 2.10. Conclusion

Dans ce chapitre nous avons présenté les concepts fondamentaux des techniques formelles de vérification et validation des systèmes. Nous avons insisté sur les réseaux de Petri, qui sont un formalisme de modélisation des systèmes. Ils représentent un cadre formel de spécification des systèmes grâce à leur sémantique. Un grand avantage de l'utilisation des réseaux de Petri est leur force expressive due à leur aspect graphique. Comme extension des réseaux de Petri, nous

avons présenté les ECATNets qui sont l'union du concept des réseaux de Petri et des types abstraits algébriques. Les ECATNets seront utilisés dans le cadre de notre travail comme un formalisme cible dans la transformation de modèles présentée dans le chapitre 5.

## Chapitre 3

### Transformation des modèles à l'aide de transformation de graphes

#### 3.1 Introduction

L'Ingénierie Dirigée par les Modèles (IDM, ou MDE : *Model Driven Engineering*) offre un cadre méthodologique et technologique qui permet d'unifier différentes façons de faire dans un processus homogène. Il est ainsi possible d'utiliser la technologie la mieux adaptée à chacune des étapes du développement du logiciel, tout en ayant un processus global de développement qui soit unifié dans un paradigme unique.

L'IDM permet cette unification grâce à l'utilisation importante des modèles (qui peuvent être exprimés dans des formalismes différents) et des transformations automatiques entre les modèles. L'utilisation intensive de modèles permet un développement souple et itératif, grâce aux raffinements et enrichissements par transformations successives. Par ailleurs, les transformations permettent de passer d'un espace technique à un autre (par exemple de UML vers les graphes ou vers du XML). Ainsi, les transformations permettent de choisir l'espace technique et le formalisme le plus adapté à chaque activité.

Dans le cadre de cette thèse nous nous intéressons à la modélisation et à la vérification des processus métiers. Les *modèles* des PM seront vérifiés par les réseaux de Petri (un autre *modèle*). Le passage du modèle des processus métiers vers le modèle réseaux de Petri sera réalisé par la transformation de graphes à l'aide de l'outil ATOM<sup>3</sup>.

Le but de ce chapitre est de donner une idée sur la transformation de modèles, notamment la transformation de graphes, sur les grammaires de graphes et un aperçu sur l'outil utilisé à savoir l'ATOM<sup>3</sup>.

#### 3.2. L'architecture dirigée par les modèles

*L'architecture dirigée par les modèles ou MDA* ("Model Driven Architecture") est une démarche de réalisation de logiciel, proposée et soutenue par l'OMG. C'est une variante particulière de l'ingénierie dirigée par les modèles.

##### 3.2.1. Notion de modèle

L'approche MDA (Model Driven Architecture), définie par l'OMG en 2000, est basée sur l'utilisation des modèles et essaye de répondre aux comment, quand, quoi et pourquoi

modéliser. MDA inclut la définition de plusieurs standards, notamment UML (Unified Modeling Language), MOF (Meta Object Facility) et XMI (XML Metadata Interchange). Le principe clé de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement d'une application. Plus précisément, MDA préconise l'élaboration de modèles d'exigences (CIM : Computation Independent Model), d'analyse et de conception (PIM : Platform Independent Model) et de code (PSM : Platform Specific Model). L'objectif majeur de MDA est l'élaboration de modèles indépendants des détails techniques des plateformes d'exécution (J2EE, .Net, PHP ou autres), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité [blanc05].

#### **a) Le modèle des besoins CIM**

La première tâche à réaliser lors de la construction d'une nouvelle application est de *spécifier les besoins* (exigences) du client. L'objectif est de créer un modèle des besoins de la future application. En UML, un diagramme de cas d'utilisation peut être utilisé comme modèle des besoins. Ce modèle doit définir les fonctionnalités de l'application et les autres entités avec lesquelles elle interagit. La création d'un modèle des besoins est d'une importance capitale. Cela permet d'exprimer clairement les liens de traçabilité avec les modèles qui seront construits dans les autres phases du cycle de développement de l'application, comme les modèles d'analyse et de conception [Blanc05].

#### **b) Le modèle d'analyse et de conception abstraite PIM**

Le travail d'*analyse et de conception* peut commencer à partir du modèle des besoins. Ce travail utilise aussi un modèle. L'analyse et la conception sont les étapes où la modélisation est la plus présente, d'abord avec les méthodes Merise et Coad/Yourdon puis avec les méthodes objet Schlear et Mellor, OMT, OOSE et Booch. Ces méthodes proposent toutes leurs propres modèles. Aujourd'hui, le langage UML s'est imposé comme la référence pour réaliser tous les modèles d'analyse et de conception [Blanc05].

#### **c) Le modèle de code PSM**

Le travail de génération de code peut commencer à partir des modèles d'analyse et de conception. Cette phase, la plus délicate du MDA, doit elle aussi utiliser des modèles. MDA considère que le code d'une application peut être facilement obtenu à partir de modèles de code. La différence principale entre un modèle de code et un modèle d'analyse



et de conception réside dans le fait que le modèle de code est lié à une plate-forme d'exécution. Ces modèles de code sont appelés des PSM. Les modèles de code servent essentiellement à faciliter la génération de code à partir d'un modèle d'analyse et de conception. Ils contiennent toutes les informations nécessaires à l'exploitation d'une plate-forme d'exécution. Pour MDA, le code d'une application se résume à une suite de lignes textuelles, comme un fichier C++, alors qu'un modèle de code est plutôt une représentation structurée incluant les concepts de boucle, condition, instruction, composant, événement, etc. L'écriture de code à partir d'un modèle de code est donc une opération assez triviale [Blanc05].

### 3.2.2. Transformation de modèles

Elle est définie comme étant le processus de convertir un modèle d'un système à un autre modèle du même système [OMG03b].

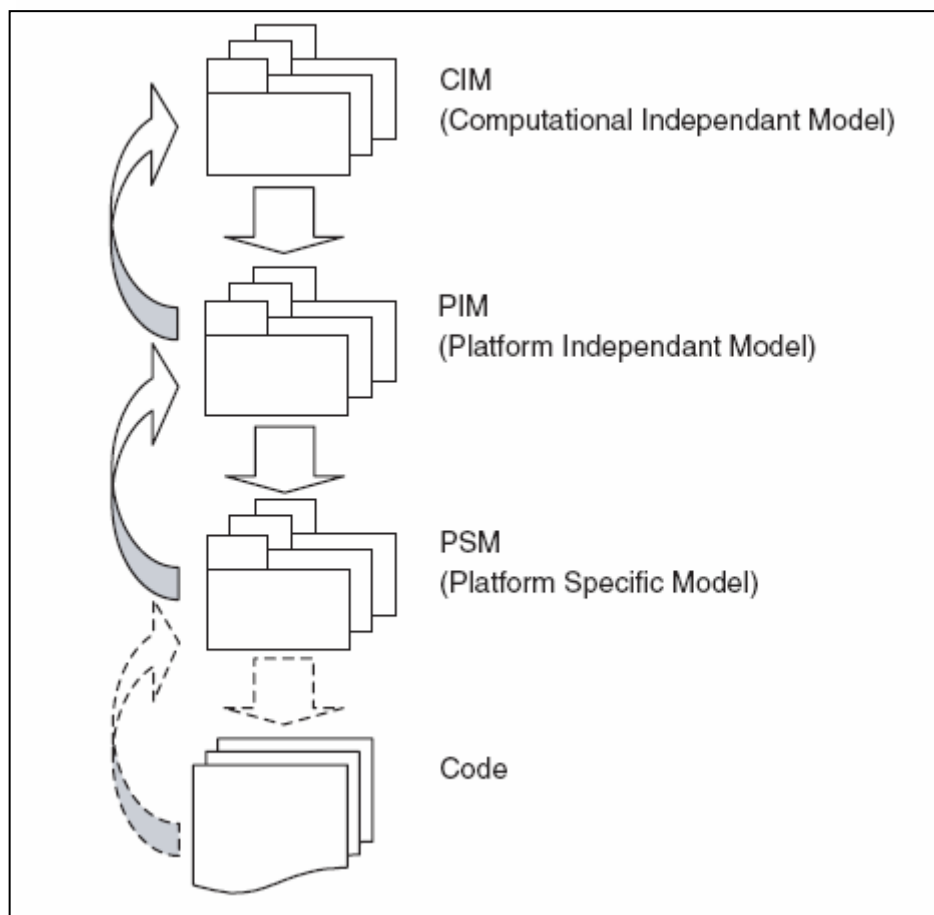


Figure 3.1 : Aperçu globale du processus de transformations de modèles de l'approche MDA

## Exemples

- Transformer un automate d'état fini non déterministe (AFN) vers un automate d'état fini déterministe (AFD).
- Transformer un diagramme UML vers un réseau de Petri.
- Transformer un processus métier vers un réseau de Petri,
- ...

### 3.2.3. Méta-modélisation et transformation

Tout comme un programme, un modèle destiné à être traité par une machine ne doit pas avoir une interprétation ambiguë, et doit être exprimé en respectant des règles bien définies. Il s'agit là de la démarche de méta-modélisation qui s'attache à définir des formalismes pour les langages de modélisation. Une fois le formalisme défini, on peut garantir que tout modèle conforme à ce formalisme pourra être traité correctement.

Les différents niveaux d'abstraction liés au concept de méta-modélisation sont résumés dans le tableau de la figure 3.2.

|                         |   |
|-------------------------|---|
| <b>Méta-méta-modèle</b> | Langage de spécification des méta-modèles             |
| <b>Méta-modèle</b>      | Définition du langage utilisé pour exprimer le modèle |
| <b>Modèle</b>           | Abstraction du système                                |
| <b>Système</b>          | Information et flux de contrôle d'un domaine          |

**Figure 3.2** : Niveaux d'abstraction de la méta-modélisation. [OMG04]

*Au premier niveau*, se trouve le système à étudier: il peut s'agir selon le cas d'un système d'information d'une entreprise, ou d'une tâche que l'on souhaite automatiser.

*Au second niveau*, se trouve le modèle qui décrit certains aspects du système que l'on veut étudier: il peut s'agir d'un diagramme de classes UML, d'un modèle conceptuel de traitement MERISE, ou de tout schéma qui représente une vue abstraite des objets modélisés.

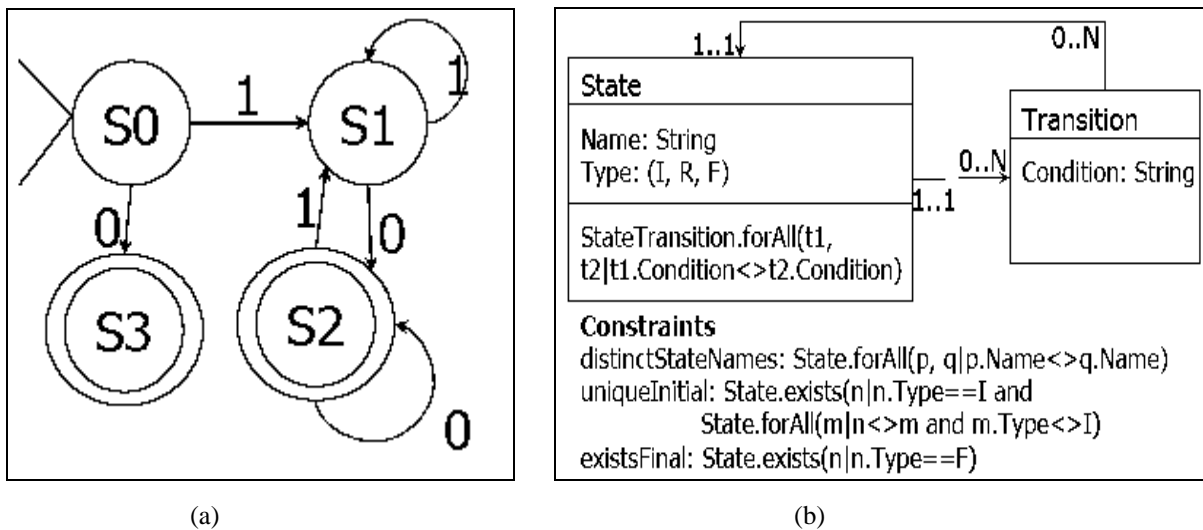
*Au troisième niveau* se trouve le langage de modélisation ou méta-modèle : par exemple les définitions d'un modèle de conception de données MERISE, d'un diagramme d'objet UML ou d'une spécification dans le langage B. C'est ce langage qui doit faire l'objet d'une spécification formelle.

*Au quatrième niveau*, se trouve le méta-métamodèle: il s'agit d'un langage qui doit être assez générique pour définir les différents langages de modélisation existants et assez précis

pour exprimer les règles que chaque langage doit respecter pour pouvoir être traité automatiquement.

### Exemple

La figure 3.3.(a) montre un modèle AFD (Automate d'état Fini Déterministe) des nombres binaires pairs et la figure 3.3.(b) montre son méta-modèle qui consiste en un diagramme de classes UML et les contraintes sont exprimées dans le langage OCL (Object Constraint Language) [OMG04].



**Figure 3.3 :** (a) Modèle AFD des nombre binaires pairs (b) Méta-modèle : Formalisme AFD (Diagramme de classe UML + OCL) [DeLara02]

Dans l'architecture proposée par l'OMG, UML se situe au troisième niveau; un langage générique de description des modèles est proposé : le MOF (Meta Object Facility).

La figure 3.4 montre une présentation générale des principaux concepts impliqués dans la transformation de modèles. La figure 3.4 présente un simple scénario d'une transformation avec un modèle en entrée (source) et un modèle en sortie (cible). Les deux modèles sont conformes avec leurs méta-modèles. Une transformation est définie en respectant les méta-modèles. Cette définition est exécutée sur des modèles concrets par un outil de transformation.

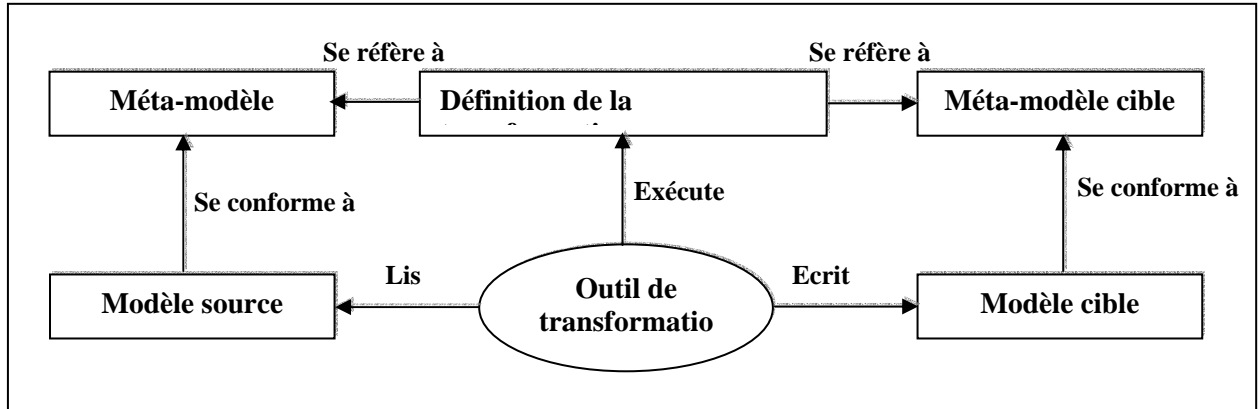


Figure 3.4 : Concepts de base de la transformation de modèles [Czarnecki06]

Généralement, une transformation peut avoir des modèles sources et cibles multiples. En outre, les méta-modèles source et cible peuvent être les mêmes dans certaines situations.

Il existe dans la littérature trois types de transformations :

1. **Les transformations verticales.** La source et la cible d'une transformation verticale sont définies à différents niveaux d'abstraction. Une transformation qui baisse le niveau d'abstraction est appelée un raffinement (PIM vers PSM). Une transformation qui élève le niveau est appelée une abstraction (PSM vers PIM).
2. **Les transformations horizontales.** Une transformation horizontale modifie la représentation source tout en conservant le même niveau d'abstraction (PIM vers PIM) ou (PSM vers PSM). La modification peut être l'ajout, la modification, la suppression ou la restructuration d'informations.
3. **Les transformations obliques.** Une transformation oblique combine une transformation horizontale et une verticale. Ce type de transformation est notamment utilisé par les compilateurs, qui effectuent des optimisations du code source avant de générer le code exécutable.

De manière orthogonale à cette catégorisation, on peut distinguer les transformations de type *modèle vers code* et celles de type *modèle vers modèle*. En effet, même si les premières peuvent être considérées comme un cas particulier des secondes (il suffit de fournir un méta-modèle pour le langage de programmation cible).

### 3.2.4. Classification des approches de transformation

Les approches de transformation de modèle ont été classées en se basant sur plusieurs points de vue. Chaque point de vue permet une classification particulière. Dans la littérature on distingue la classification faite par **CZarneki** [Czarnecki03] qui se base sur les techniques de transformation utilisées dans les approches et les facettes qui les caractérisent. Une autre classification a été proposée par **Prakash** [Prakash06] qui consiste en une classification multidimensionnelle dans laquelle il se focalise sur une dimension non traitée dans les autres classifications qui est le domaine d'application et la généralité (capacité de transformer n'importe quel modèle d'entrée à n'importe quel modèle de sortie). Il prend en considération l'aspect utilisation de transformation de modèles dans l'ingénierie des méthodes.

Selon **CZarneki** [Czarnecki03], il existe deux types de transformation de modèles : les transformations de type *modèle vers code* qui sont aujourd'hui relativement matures et les transformations de type *modèle vers modèles*.

#### 3.2.4.1 Transformations de type Modèle vers code

On distingue deux approches de transformations de type *modèle vers code* : les approches basées sur le principe du *visiteur* (*Visitor-based approach*) ou celles basées sur le principe des *patrons* (*Template-based approach*).

- a. Les approches reposant sur le principe du *visiteur* consistent à traverser le modèle en lui ajoutant des éléments (mécanismes visiteurs) qui réduisent la différence de sémantique entre le modèle et le langage de programmation cible. Le code est obtenu en parcourant le modèle enrichi pour créer un flux de texte.
- b. Les approches basées sur le principe des *patrons* sont actuellement les plus utilisées. Le code cible contient des morceaux de méta-code utilisés pour accéder aux informations du modèle source. La majorité des outils MDA couramment disponibles supporte ce principe de génération de code à partir de modèle. Parmi les outils basés sur ce principe, on peut citer : *OptimalJ*, *XDE* (qui fournissent la transformation modèle vers modèle aussi), *JET*, *ArcStyler* et *AndroMDA* (un générateur de code qui se repose notamment sur la technologie ouverte *Velocity* pour l'écriture des patrons).

### 3.2.4.2. Transformations de type modèle vers modèle

Les transformations de type *modèle vers modèle* sont aujourd'hui moins maîtrisées, cependant, elles ont beaucoup évolué au cours de ces dernières années, et plus particulièrement depuis l'apparition du MDA [OMG04].

#### a)- Utilité des transformations modèle vers modèle

Quand on trouve un grand espace d'abstraction entre un PIM et un PSM, il serait plus facile de générer des modèles intermédiaires au lieu d'aller directement vers le PSM cible. Ces modèles peuvent être utiles pour l'optimisation ou bien pour des buts de débogage. Ces transformations sont utiles pour le calcul des différentes vues du système et leurs synchronisation et aussi pour des buts de vérification et de validation.

#### b)- Structure d'une transformation

Une transformation de modèle est principalement caractérisée par la combinaison des éléments suivants: des règles de transformation, une relation entre la source et la cible, un ordonnancement des règles, une organisation des règles, une traçabilité et une direction.

- **Spécification**

Certaines approches de transformation fournissent un mécanisme dédié de spécifications, tel que des pré-conditions et des post-conditions exprimées en OCL. Des spécifications particulières de transformation peuvent représenter une fonction entre les modèles source et cible et peuvent être exécutables. Mais généralement, les spécifications décrivent des relations et elles ne sont pas exécutables.

- **Règles de transformation**

Une règle de transformation est composée de deux parties : un côté gauche (*LHS, Left Hand Side*) qui accède au modèle source, et un côté droit (*RHS, Right Hand Side*) qui accède au modèle cible. Une règle comporte également une logique, qui exprime des contraintes ou calculs sur les éléments des modèles source et cible. Une logique peut avoir la forme déclarative ou impérative. Une logique déclarative consiste à spécifier des relations entre les éléments du modèle source et des éléments du modèle cible. Une logique impérative correspond le plus souvent, mais pas nécessairement, à l'utilisation de langages de programmation pour manipuler directement les éléments des modèles par le biais d'interfaces dédiées. Optionnellement, une règle, peut être bidirectionnelle, comporter des paramètres, ou encore nécessiter la construction de structures intermédiaires.

- ❖ **Organisation des règles** : Elle définit comment composer plusieurs règles de transformation. Les règles peuvent être organisées de façon modulaire, avec la notion d'importation. Les règles peuvent également utiliser la réutilisation, par le biais de mécanismes d'héritage entre règles, ou la composition, par le biais d'un ordonnancement explicite. Enfin, les règles peuvent être organisées selon une structure dépendante du modèle source ou du modèle cible.
- ❖ **Ordonnancement des règles** : Les mécanismes d'ordonnancement déterminent l'ordre dans lequel les règles sont appliquées. Dans le cas d'un ordonnancement implicite, l'algorithme d'ordonnancement est défini par l'outil de transformation. Dans le cas d'un ordonnancement explicite, des mécanismes permettent de spécifier l'ordre d'exécution des règles. Cet ordre d'exécution peut être défini de manière externe ou interne : tandis qu'un mécanisme externe établit une séparation claire entre les règles et la logique d'ordonnancement, un mécanisme interne permet aux règles d'invoquer d'autres règles. Enfin, l'ordonnancement des règles peut se baser également sur des conditions, des itérations ou sur une séparation en plusieurs phases dont certaines règles ne pouvant être appliquées que dans certaines phases.
- ❖ **Relation entre les modèles source et cible** : Pour certains types de transformations, la création d'un nouveau modèle cible est nécessaire. Pour d'autres, la source et la cible sont le même modèle, ce qui revient en fait à une modification de modèle.
- ❖ **Augmentation** : Elle a un rapport avec la capacité de mettre à jour les modèles cible existants qui sont basés sur des changements dans les modèles source.
- ❖ **Direction** : Les transformations peuvent être unidirectionnelles ou bidirectionnelles. Dans le premier cas, le modèle cible est calculé ou mis à jour sur la base du modèle source uniquement. Dans le second cas, une synchronisation entre les modèles source et cible est possible.
- ❖ **Traçabilité** : Les transformations peuvent archiver les corrélations entre les éléments des modèles source et cible. Certaines approches fournissent des mécanismes dédiés pour supporter la traçabilité. Dans les autres cas, le développeur doit implémenter la traçabilité de la même manière qu'il crée n'importe quel autre lien dans un modèle.

Les différentes caractéristiques présentées dans cette section constituent des points de variation qui distinguent les différentes approches pour la définition de transformation de type *modèle vers modèle*.

- **Différentes approches**

On peut globalement distinguer cinq types d'approches :

**a) Approches par manipulation directe**

Ces approches se basent sur une représentation interne des modèles source et cible, et sur un ensemble d'APIs pour les manipuler. Elles sont généralement implémentées comme des cadres structurants orientés objets qui fournissent un ensemble minimal de concepts – sous forme de classes abstraites par exemple. L'implémentation des règles et leur ordonnancement restent à la charge du développeur.

**b) Approches relationnelles**

Ces approches sont celles qui utilisent une logique déclarative reposant sur des relations d'ordre mathématique. L'idée de base est de spécifier les relations entre les éléments des modèles source et cible par le biais de contraintes. L'utilisation de la programmation logique est particulièrement adaptée à ce type d'approche. Généralement, les transformations produites sont bidirectionnelles.

**c) Approches basées sur les transformations de graphes**

Ces approches, qui exploitent les travaux réalisés sur les transformations de graphes [Andries99]. Elles sont similaires aux approches relationnelles dans le sens où elles permettent l'expression des transformations sous une forme déclarative. Néanmoins, les règles ne sont plus définies pour des éléments simples mais pour des fragments de modèles: on parle de *filtrage de motif (pattern matching)*. Les motifs dans le modèle source, correspondant à certains critères, sont remplacés par d'autres motifs du modèle cible. Les motifs, ou fragments de modèles, sont exprimés soit dans les syntaxes concrètes respectives des modèles soit dans leur syntaxe abstraite.

**d) Approches basées sur la structure**

Ces approches distinguent deux phases. La première consiste à créer la structure hiérarchique du modèle cible. La seconde consiste à ajuster les attributs et références dans le modèle cible.



### e) Approches hybrides

Les approches hybrides sont une combinaison des différentes techniques. On peut notamment retrouver des approches utilisant à la fois des règles à logique déclarative et des règles à logique impérative. **ATL** et **XDE** sont deux exemples d'approches hybrides.

## 3.3. Transformations de graphes

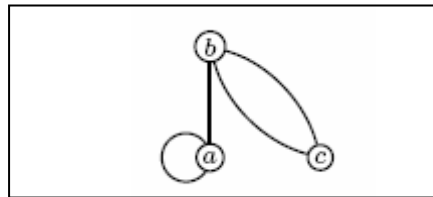
Avant d'aborder les grammaires de graphes, nous allons rappeler quelques concepts de base relatifs aux graphes

### 3.3.1. Notion de graphe

Il existe deux types de graphes : les graphes non orientés et les graphes orientés.

#### 3.3.1.1 Graphes non orientés

Un **graphe** est constitué de **sommets (nœuds)** qui sont reliés par des **arêtes**. Deux sommets reliés par une arête sont **adjacents**.

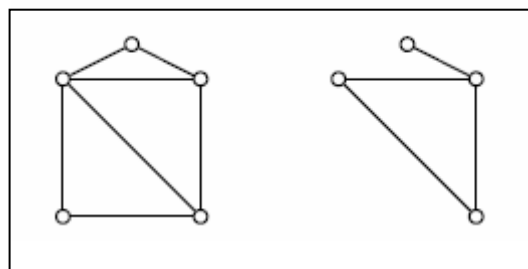


**Figure 3.5:** Graphe non orienté

Le nombre de sommets présents dans un graphe est appelé **ordre du graphe**. Le graphe de la figure 3.5 est d'ordre 3.

Le **degré** d'un sommet est le nombre d'arêtes dont ce sommet est une extrémité. Le degré du sommet c de la figure 3.5 est 2.

Un **sous-graphe** d'un graphe G est un graphe G' composé de certains sommets de G, ainsi que toutes les arêtes qui relient ces sommets.



(a)

(b)

**Figure 3.6 :** (a) graphe G (b) sous-graphe de G

### 3.3.1.2 Graphes orientés

Un **graphe orienté** est un graphe dont les arêtes sont orientées: on parle alors de l'origine et de l'extrémité d'une arête. Dans un graphe orienté une arête est dénommée **arc**.

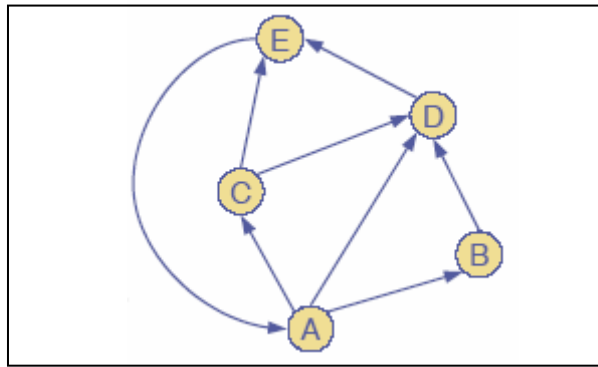


Figure 3.7 : Graphe orienté

Un **graphe étiqueté** est un graphe orienté, dont les arcs possèdent des étiquettes. Si toutes les étiquettes sont des nombres positifs, on parle de **graphe pondéré**.

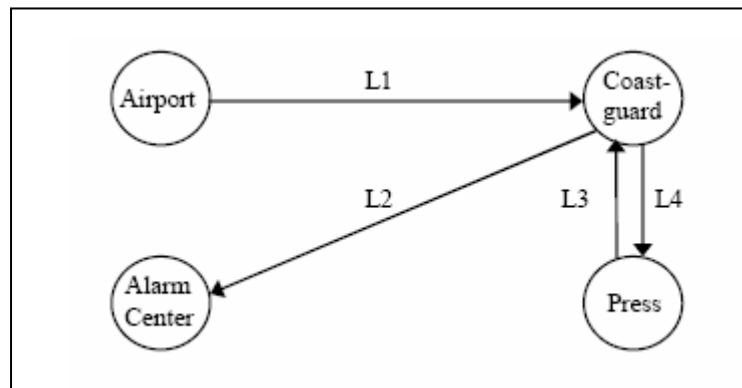


Figure 3.8 : Graphe orienté étiqueté

Un **graphe attribué** est un graphe qui peut contenir un ensemble prédéfini d'attributs.

### 3.3.2. Grammaires de graphes

Les graphes et les diagrammes sont un moyen très pratique pour la description des structures complexes et des systèmes ainsi que pour la modélisation des idées de manière directe et intuitive. Si les graphes servent à visualiser les structures complexes des modèles d'une façon simple et intuitive, les transformations de graphes peuvent être exploitées pour spécifier comment ces modèles peuvent évoluer.

Les transformations de graphes [Rozenberg99] ont évolué dans la répercussion à l'imperfection dans l'expressivité des approches de réécriture classique comme les grammaires de Chomsky et la réécriture de termes pour s'y prendre avec les structures non linéaires. Une transformation de graphe [Karsai04, Andries99, Rozenberg99] consiste en l'application d'une règle à un graphe et itérer ce processus. Chaque application de règle transforme un graphe par le remplacement d'une de ses parties par un autre graphe. Autrement dit, la transformation de graphe est le processus de choisir une règle d'un ensemble indiqué, appliquer cette règle à un graphe et réitérer le processus jusqu'à ce qu'aucune règle ne puisse être appliquée.

La transformation de graphe est spécifiée sous forme d'un modèle de grammaires de graphes, ces dernières sont une généralisation, pour les graphes, des grammaires de Chomsky. Elles sont composées de règles dont chacune est composée d'un graphe de coté gauche (**LHS**) et d'un graphe de coté droit (**RHS**).

### Définition

Une *grammaire de graphe* [Andries99] est un ensemble  $P$  de règles muni d'un graphe initial  $S$  et d'un ensemble  $T$  de symboles terminaux.

### Exemple

Règle permettant d'éliminer le non déterminisme dans un AFD.

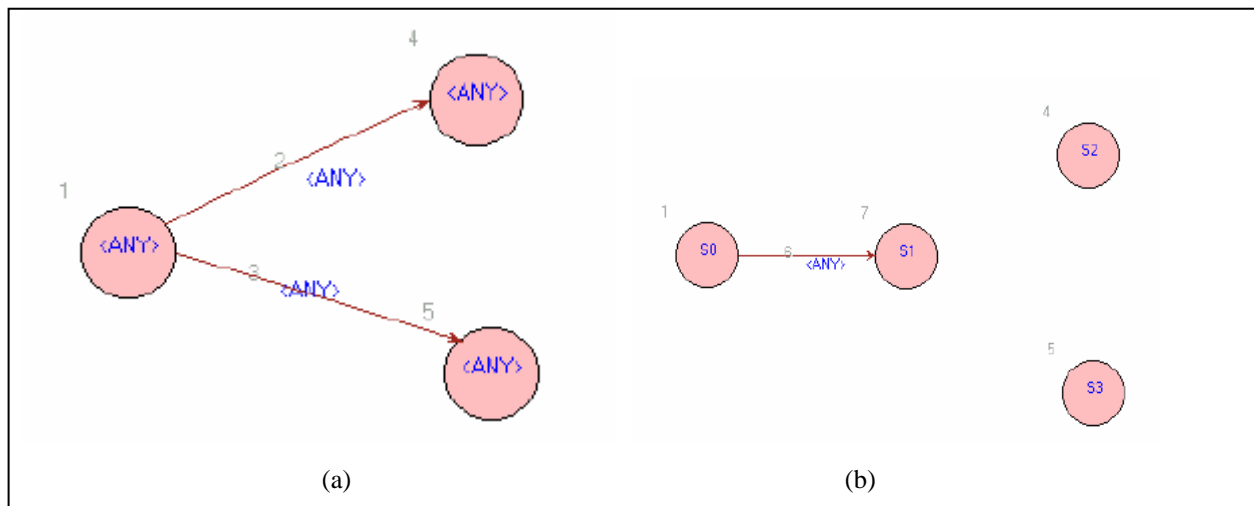


Figure 3.9 : (a) Partie gauche d'une règle (b) Partie droite d'une règle

Cette règle permet d'éliminer le non déterminisme lors de la conversion d'un automate non déterministe en un automate déterministe.

Une grammaire de graphes distingue les graphes non terminaux, qui sont les résultats intermédiaires sur lesquels les règles sont appliquées, des graphes terminaux dont on ne peut plus appliquer de règles, on dit que ces derniers sont dans le langage engendré par la grammaire. Pour vérifier si un graphe  $G$  est dans les langages engendrés par une grammaire de graphe, il doit être analysé. Le processus d'analyse va déterminer une séquence de règles dérivant  $G$ .

### 3.3.2.1. Le principe de règles

Une règle de transformation de graphe est définie par :

$$r = (L, R, K, \text{glue}, \text{emb}, \text{cond})$$

Elle consiste en:

- Deux graphes  $L$  graphe de côté gauche et  $R$  graphe de côté droit.
- Un sous graphe  $K$  de  $L$ .
- Une occurrence  $\text{glue}$  de  $K$  dans  $R$  qui relie le sous graphe avec le graphe de coté droit.
- Une relation d'enfoncement qui relie les sommets du graphe de coté gauche et ceux du graphe du coté droit.
- Un ensemble  $\text{cond}$  qui spécifie les conditions d'application de la règle.

### 3.3.2.2. Application des règles

L'application d'une règle  $r = (L, R, K, \text{glue}, \text{emb}, \text{cond})$  à un graphe  $G$  produit un graphe résultant  $H$ . Le graphe  $H$  fourni, peut être obtenu depuis le graphe d'origine  $G$  en passant par les cinq étapes suivantes :

1. Choisir une occurrence du graphe de coté gauche  $L$  dans  $G$ .
2. Vérifier les conditions d'application d'après  $\text{cond}$ .
3. Retirer l'occurrence de  $L$  (jusqu'à  $K$ ) de  $G$  ainsi que les arcs pendillé, c-à-d tout les arcs qui ont perdu leurs sources et/ou leurs destinations. Ce qui fourni le graphe de contexte  $D$  de  $L$  qui a laissé une occurrence de  $K$ .
4. Coller le graphe de contexte  $D$  et le graphe de coté droit  $R$  suivant l'occurrence de  $K$  dans  $D$  et dans  $R$ . c'est la construction de l'union de disjonction de  $D$  et  $R$  et, pour chaque point dans  $K$ , identifier le point correspondant dans  $D$  avec le point correspondant dans  $R$ .
5. Enfoncer le graphe du côté droit dans le graphe de contexte de  $L$  suivant la relation d'enfoncement  $\text{emb}$  :

L'application de  $r$  sur un graphe  $G$  pour fournir un graphe  $H$  est appelée une dérivation directe depuis  $G$  vers  $H$  à travers  $r$ , elle est dénotée par  $G \Rightarrow_r H$  ou simplement par  $G \Rightarrow H$ .

En donnant les notions de règle et de dérivation directe comme étant les concepts élémentaires de la transformation de graphe, on peut définir les systèmes de transformation de graphe, les grammaires de graphe et la notion de langages engendrés.

### 3.3.2.3. Système de transformation de graphes

Un système de transformation de graphe est un ensemble P de règles.

#### 1- Langage engendré

Supposons que nous avons un ensemble donné P de règles et un graphe  $G_0$ , une séquence de transformations de graphe successive :  $G_0 \Rightarrow G_1 \Rightarrow \dots \Rightarrow G_n$  est une dérivation à partir de  $G_0$  vers  $G_n$  par les règles de P (à condition que toutes les règles utilisées appartiennent à P).  $G_0$  est le graphe initial et  $G_n$  est le graphe dérivé de la séquence de transformation.

L'ensemble des graphes dérivés à partir d'un graphe initial S en appliquant les règles de P qui sont étiquetées par les symboles de T est dit *langage engendré* par P, S et T, et on écrit  $L(P,S,T)$ .

### 3.3.3. Outils de transformation de graphes

Il existe plusieurs outils de transformation de graphes. AGG [AGG], FUJABA [Fujaba], ATOM<sup>3</sup> [Atom3], VIATRA [Viatra], GreAT [Great], etc. sont quelques exemples d'outils de transformation de graphes existant. Après étude des différents outils, nous avons opté pour ATOM<sup>3</sup> dans le cadre de cette thèse à cause des avantages qu'il présente. Nous pouvons citer parmi ces avantages : sa simplicité, sa disponibilité, il est multi paradigmes, et il permet les deux types de transformations utilisées dans le cadre de cette thèse, à savoir les transformations de type *modèle vers code* et les transformations de type *modèle vers modèles*.

### 3.3.4. AToM<sup>3</sup> : Présentation générale

AToM<sup>3</sup> [Atom3] est un outil visuel pour la modélisation et la méta-modélisation multi-formalismes. Comme il a été implémenté en Python [Python07], il peut être exécuté, sans aucun changement, sur toutes les plateformes où un interpréteur de Python est disponible (Linux, Windows et MacOS).

Les deux tâches principales d'AToM<sup>3</sup> sont la méta-modélisation et la transformation de modèles. Pour la mété-modélisation, AToM<sup>3</sup> supporte la modélisation visuelle en utilisant

le formalisme Entité-Relation (ER) ou le formalisme de diagrammes de classes d'UML. Ceci signifie que pour méta-modéliser de nouveaux formalismes on peut utiliser le modèle ER ou le modèle des diagrammes de classes. Dans le cadre de cette thèse nous avons choisi d'utiliser le modèle des diagrammes de classes.

Afin de pouvoir spécifier entièrement les formalismes de modélisation, les méta-formalismes peuvent être étendus par l'expression de contraintes (chose qui ne peut pas être exprimée uniquement par les diagrammes de classes d'UML ou dans le modèle ER).

Les contraintes fournissent une vue sur la manière dont un constructeur est lié à un autre pour qu'il ait un sens. Les contraintes sont exprimées sous une forme textuelle. Pour ce faire, certains systèmes, dont AToM<sup>3</sup>, utilisent OCL (Object Constraint Language) d'UML. Du fait que comme AToM<sup>3</sup> est implémenté en Python, on peut également utiliser du code Python pour exprimer les contraintes.

Pour la transformation de modèles, AToM<sup>3</sup> supporte la réécriture de graphes qui utilise les règles de grammaire de graphes pour guider visuellement la procédure de transformation. Les règles sont spécifiées par l'utilisateur et ordonnées selon des critères dépendants des caractéristiques du modèle à transformer. AToM<sup>3</sup> offre la possibilité à l'utilisateur de créer une nouvelle grammaire, de charger une grammaire et de la modifier et d'exécuter une grammaire. L'exécution de la grammaire sur un modèle en entrée produit un modèle de sortie.

### 3.4. Conclusion

Dans ce chapitre, nous avons présenté le concept de transformation de modèles qui est considéré comme la clé de la démarche MDA. Nous avons présenté les différentes approches existantes en se basant sur une classification proposée dans la littérature. Une introduction aux transformations de graphes qui représentent une approche de transformation de modèles a été donnée. Nous avons également brièvement présenté ATOM<sup>3</sup>, l'outil de transformation utilisé dans notre travail. Les concepts présentés dans ce chapitre constituent un background nécessaire pour la compréhension de nos contributions dans le cadre de cette thèse et qui seront présentées dans les chapitres quatre et cinq.

## Chapitre 4

### Une approche automatique de transformation des processus métiers

#### 4.1. Introduction

Dans le premier chapitre, nous avons défini la notion de processus métier et nous avons montré l'importance de ce paradigme dans l'entreprise d'aujourd'hui. La plupart des modèles des processus métiers présentés souffrent de manque de capacités d'analyse et de vérification ce qui limite leur utilisation et réduit leur qualité.

La coordination et le contrôle automatique des processus métiers sont possibles grâce aux constructeurs de contrôle qui modélisent les comportements tels que la concurrence, l'asynchronisation et le choix. Cependant, il y a un danger réel d'introduire des anomalies dans le contrôle de flux et des inconsistances comportementales telles que l'interblocage, la vivacité, la terminaison imparfaite et la répétition des tâches multiples. Les réseaux de Petri fournissent une méthode de modélisation formelle et puissante basée sur des fondements mathématiques solides. En même temps ils possèdent une représentation graphique des modèles de systèmes et offrent plusieurs techniques d'analyse.

Dans ce chapitre nous proposons une approche totalement automatisée basée sur la méta modélisation et les grammaires de graphes afin de générer des modèles réseaux de Petri pour les processus métiers en utilisant AToM<sup>3</sup>.

Comme modèle de processus métiers, on va considérer celui proposé par E. Sivaraman et M.Kamath dans [Sivaraman02]. Une transformation automatique vers les RdPs sera présentée.

Pour analyser les RdPs obtenus, nous avons choisi d'utiliser l'outil INA. Et comme il n'est pas graphique, nous avons généré automatiquement un outil visuel de modélisation pour le traitement des modèles dans le formalisme INA. Ceci a également été réalisé à l'aide des grammaires de graphes et par le biais de l'outil de méta modélisation AToM3.

#### 4.2 Modèle de processus métiers

La figure 4.1 montre un exemple de représentation graphique d'un processus métier dans lequel les activités sont modélisées par des rectangles liés par deux types de connecteurs (and, xor) modélisés par des cercles [Sivaraman02].

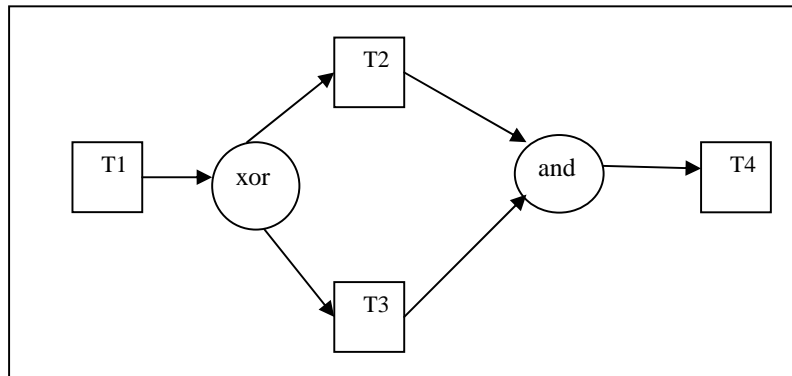


Figure 4.1 : Un exemple de processus métier.

### 4.3. Réseaux de Petri

Un réseau de Petri (comme nous l'avons rappelé dans le chapitre 2), est un graphe dirigé biparti avec deux types de nœuds appelés places et transitions. Les nœuds sont connectés par des arcs dirigés. Les connections entre deux nœuds de même type ne sont pas autorisées. Les places sont représentées par des cercles et les transitions sont représentées par des rectangles ou des bars.

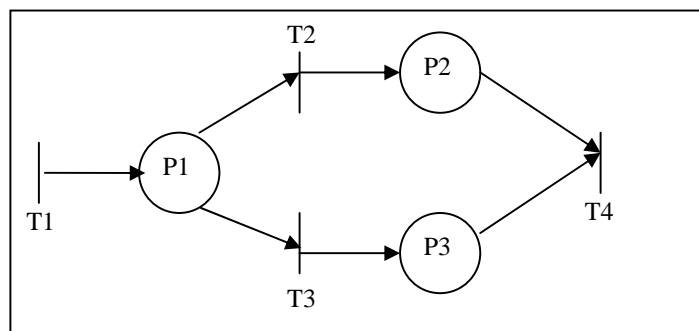


Figure 4.2: Un réseau de Petri représentant le processus métier de la Figure 4.1

### 4.4. Formalisation des processus métiers par les réseaux de Petri

Un processus peut être vu comme une collection d'événements qui sont déclenchés suite à des conditions satisfaites et rendent des conditions satisfaites après leur terminaison. Un réseau de Petri décrit cette intuition de manière idéale et sépare de manière explicite les conditions et les événements introduits dans un processus et modélise le changement d'état du système à l'aide de la simulation du mouvement des jetons. Pour transformer les modèles de processus



métiers en modèles de réseaux de Petri, nous avons utilisé les idées proposées dans [Sivaraman02].

Par exemple, le modèle réseau de Petri de la figure 4.2 est une transformation du modèle du processus métier de la figure 4.1. Nous montrons dans la section 5 comment ce modèle est généré automatiquement en utilisant notre outil.

#### **4.5 Transformation des processus métiers vers les réseaux de Petri**

Pour transformer les processus métiers vers les réseaux de Petri, nous avons fait recours à la méta modélisation, à la transformation de modèles et aux grammaires de graphes (voir chapitre 3). Nous avons utilisé AToM<sup>3</sup> pour la méta modélisation des formalismes de processus métiers et celui des réseaux de Petri. Nous avons ainsi défini une grammaire de graphe de onze règles pour réaliser cette transformation [Elmansouri07a, [Elmansouri07b, [Elmansouri08b].

##### **4.5.1 Méta-Modélisation des Processus Métiers et des réseaux de Petri [Elmansouri08b]**

Pour construire des modèles en AToM<sup>3</sup>, nous devons définir un méta-modèle pour eux. Le méta-formalisme utilisé dans ce travail est le modèle de diagrammes de classe d'UML. Les contraintes sont exprimées en code Python.

###### *4.5.1.1. Méta Modélisation des Processus métiers*

Du fait que les processus métiers consistent en des tâches et en deux types de connecteurs XOR et AND et que chaque tâche peut être reliée au connecteur par un arc entrant ou sortant, nous avons proposé pour méta modéliser les processus métiers, quatre classes comme le montre la figure 4.3.

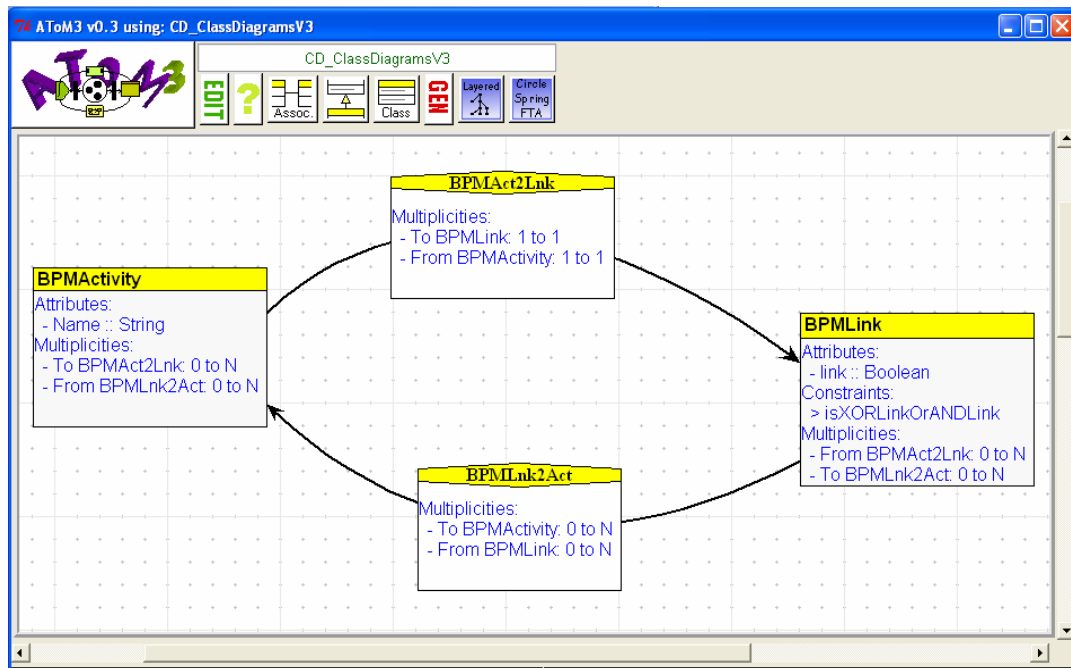


Figure 4.3 Le Meta-Model des processus métiers

#### 4.5.1.2. Méta-Modélisation des réseaux de Petri

Puisque les réseaux de Petri consistent en des places, des transitions et d'arcs de places vers des transitions et d'arcs de transitions vers les places, nous avons proposé pour les méta modéliser, quatre classes comme le montre la figure 4.4.

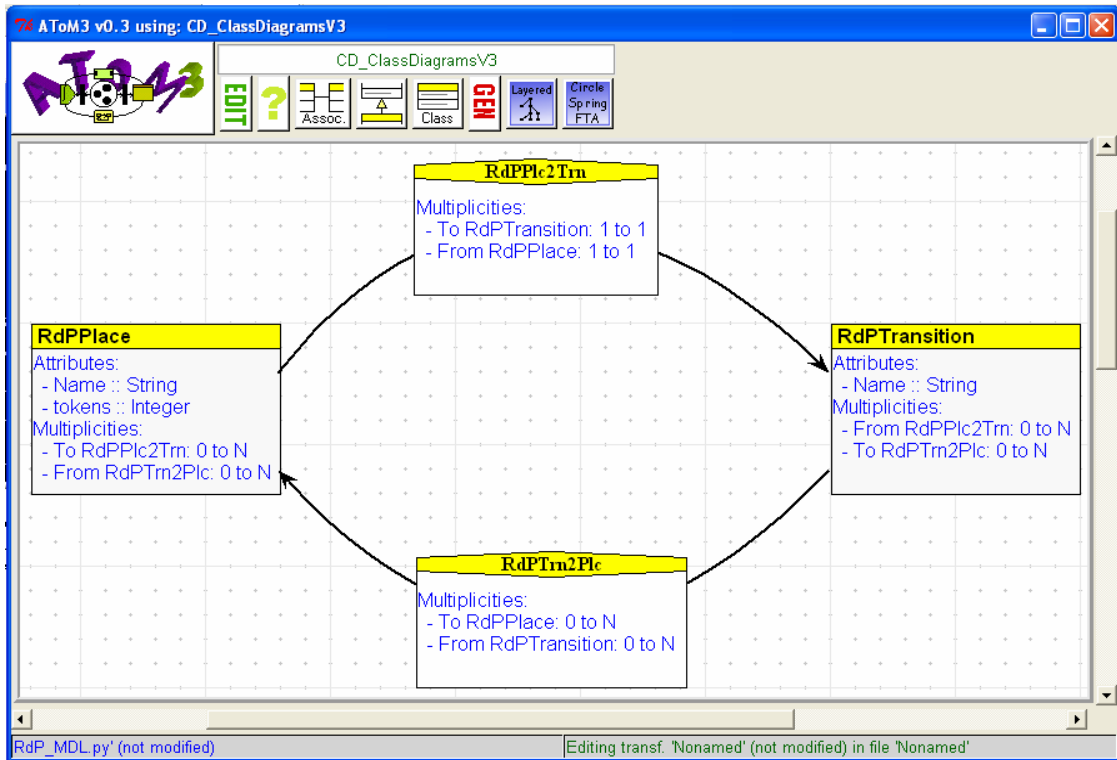


Figure 4.4 : Méta- Modél des réseaux de Petri

#### 4.5.2. Génération de l'outil des Processus Métiers

Etant donnés nos méta-modèles décrits dans les sections 4.5.1 et 4.5.2, nous avons généré, en utilisant AToM<sup>3</sup>, un environnement visuel de modélisation pour les processus métiers tel qu'il est présenté dans la figure 4.5.

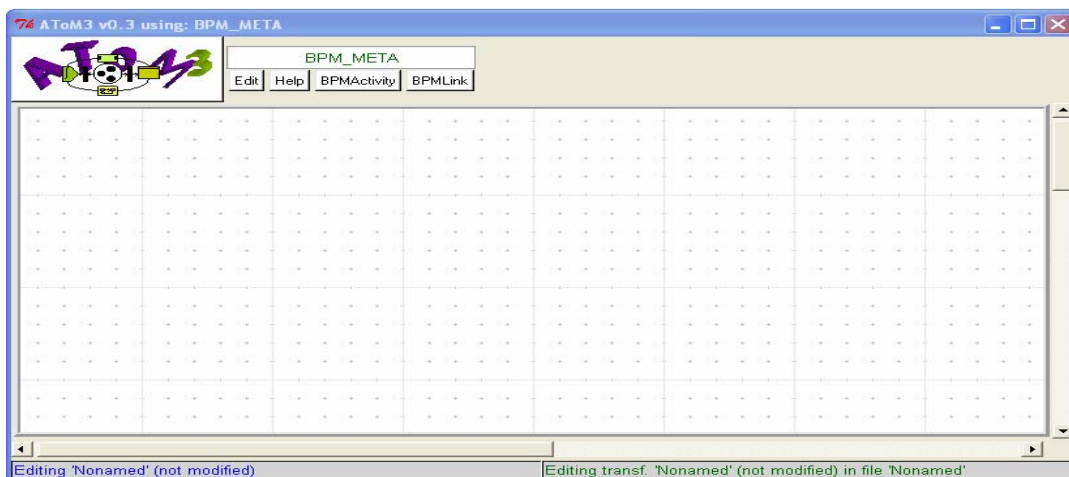


Figure 4.5 : Outil généré pour les processus métiers

### 4.5.3. Grammaire de graphes pour la transformation des processus métiers vers les réseaux de Petri

Afin de produire les formalismes des réseaux de Petri générés par notre outil, nous avons proposé une grammaire de graphes avec onze règles qui seront exécutées dans un ordre ascendant [Elmansouri08a]. L'application de cette grammaire aux processus métiers créés par notre outil (voir figure 4.5) conduit à la génération du réseau de Petri équivalent. Ces règles sont décrites comme suit :

- **Règle1: "rule\_Activity2transition"** (priorité 1): appliquée pour relier toute *Activité* (non encore traitée) dans le processus métier avec une transition du réseau de Petri et spécifier que le nom de la *transition* attachée est le même nom que celui de l'*activité* correspondante.

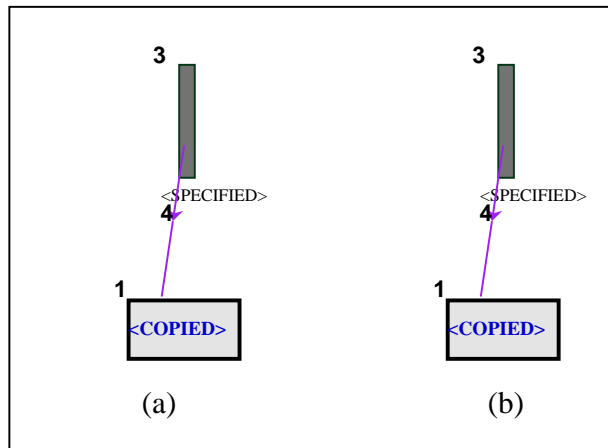


Figure 4.6: (a) LHS de la règle 1 (b) RHS de la règle 1

- **Règle2: "rule\_Xor2Place"** (priorité 2): Appliquée pour relier à chaque lien *Xor* dans le processus métier, une *place* dans le réseau de Petri.

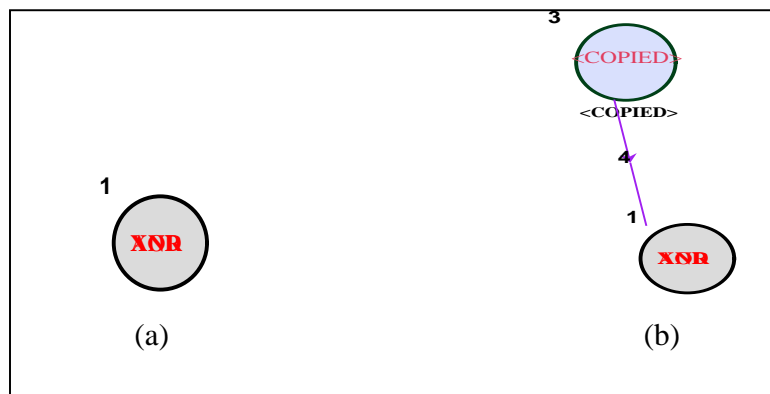


Figure 4.7: (a) LHS de la règle 2 (b) RHS de la règle 2

- **Règle3: "rule\_Xor\_trn2pl"** (priorité 3): Appliquée pour localiser un Arc de l'Activité vers un lien Xor et de générer donc un arc sortant de la Transition ( attachée à l'activité) vers la Place (attachée au lien Xor)

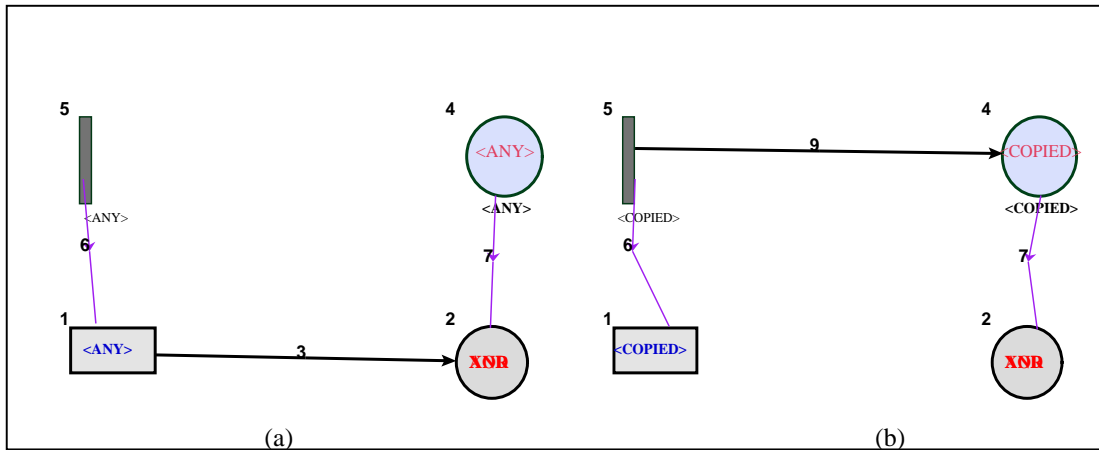


Figure 4.8: (a) LHS de la règle 3 (b) RHS de la règle 3

- **Règle4: "rule\_Xor\_pl2tran"** (priorité 4): Appliquée pour localiser un Arc sortant du lien Xor vers l'Activité et de générer donc un arc entrant de la Place(attachée au lien Xor) vers la Transition (attachée à l'activité)

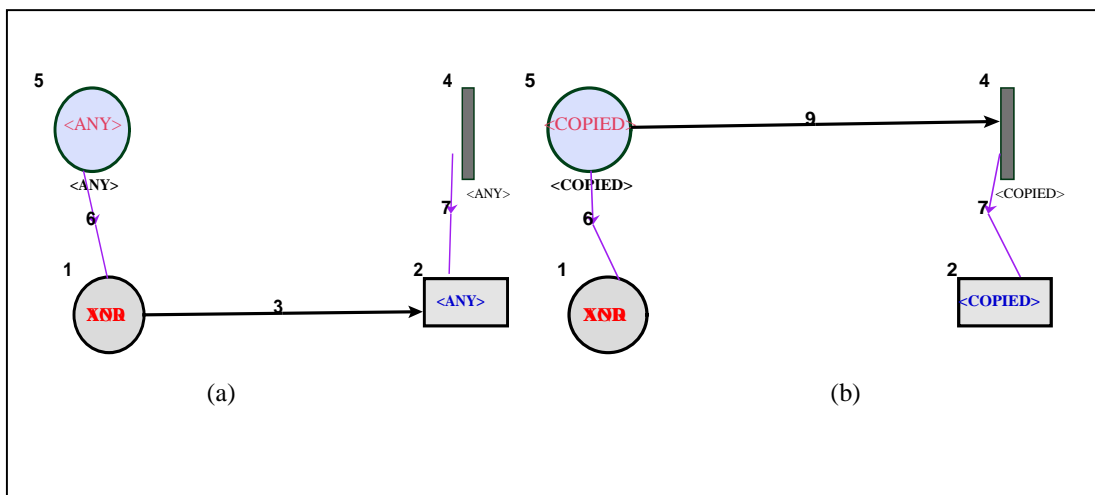


Figure 4.9: (a) LHS de la règle 4 (b) RHS de la règle 4

**Règle5: "rule\_Xor\_DelLink" (priorité 5):** Une fois tous les liens Xor sont traités, la règle 5 aura pour but de les supprimer du modèle.

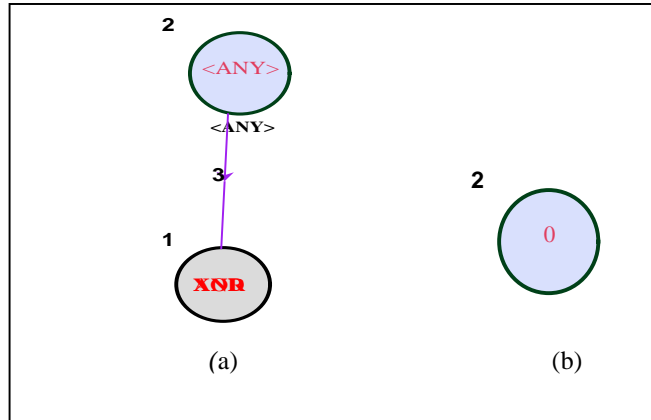


Figure 4.10: (a) LHS de la règle 5 (b) RHS de la règle 5

- **Règle6: "rule\_AND\_trn2pl" (priorité 6):** Appliquée pour localiser un Arc de l'Activité vers un lien And et de générer donc un arc sortant de la Transition (attachée à l'activité) vers la Place (attachée au lien And). Cette règle supprime l'arc de l'Activité vers un lien And.

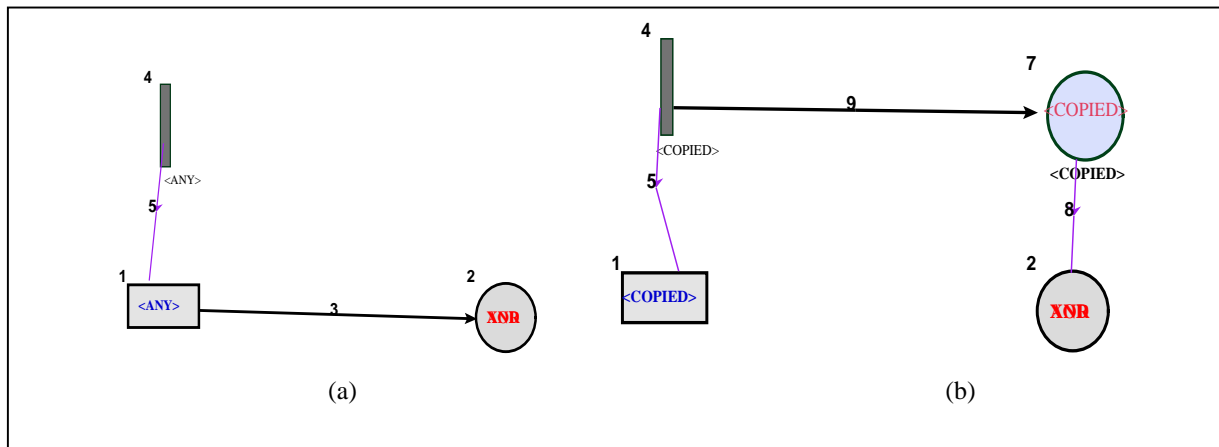


Figure 4.11: (a) LHS de la règle 6 (b) RHS de la règle 6

- **Règle 7: rule\_AND\_pl2trn (priority 7):** Appliquée pour localiser un Arc sortant du lien AND vers l'Activité et de générer donc un arc entrant de la Place(attachée au lien AND) vers la Transition (attachée à l'activité)

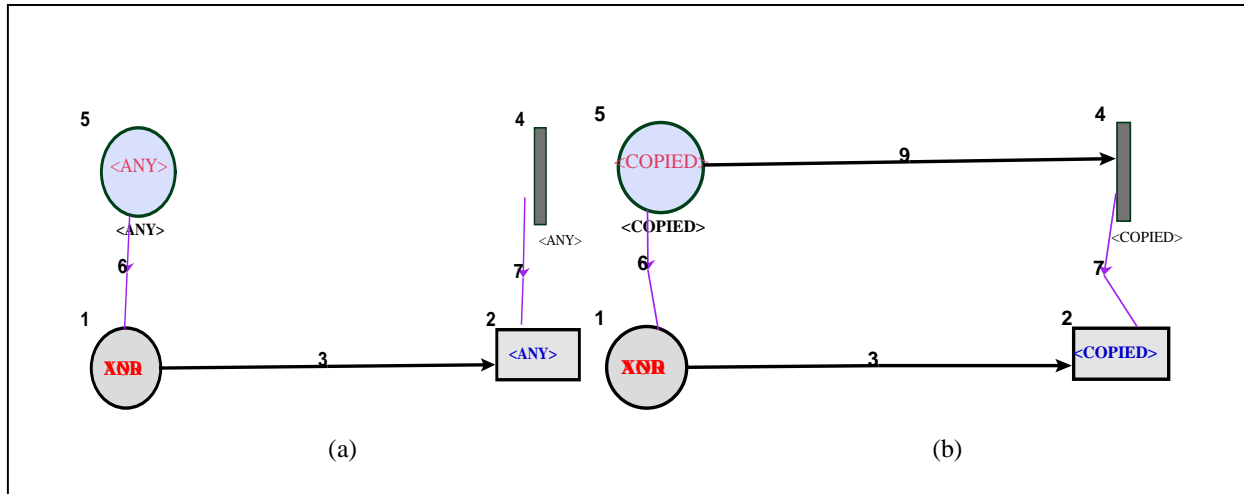


Figure 4.12: (a) LHS de la règle 7 (b) RHS de la règle 7

- **Règle 8: rule\_AND\_SeparatePIFromLink (priority 8):** Appliquée pour séparer la *Place* attachée au lien *AND*.

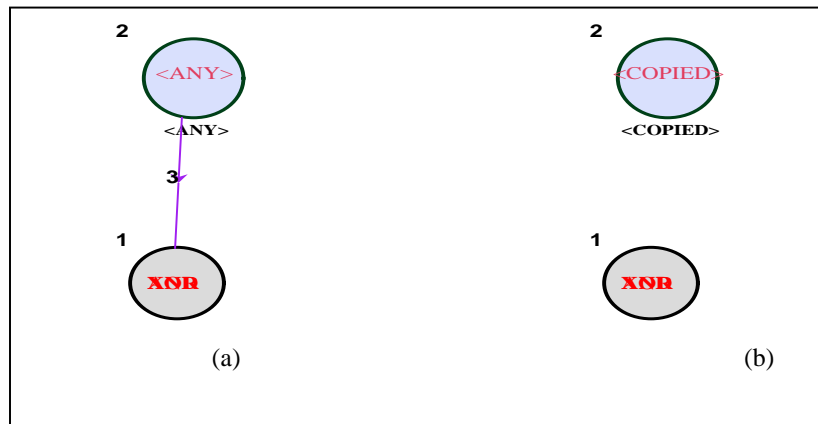


Figure 4.13: (a) LHS de la règle 8 (b) RHS de la règle 8

- **Règle 9: rule\_AND\_InitActivities (priority 9):** Utilisée pour localiser et initialiser les attributs temporaires dans les activités pour d'autres traitements.

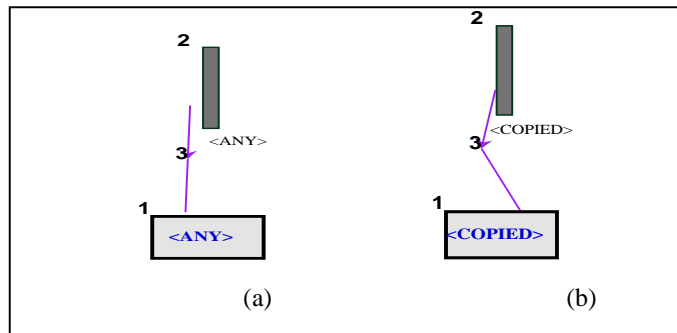


Figure 4.14: (a) LHS de la règle 9 (b) RHS de la règle 9

- **Règle 10: rule\_Del\_ANDLink (priority 10):** Une fois tous les liens AND traités, cette règle les supprimera du modèle.

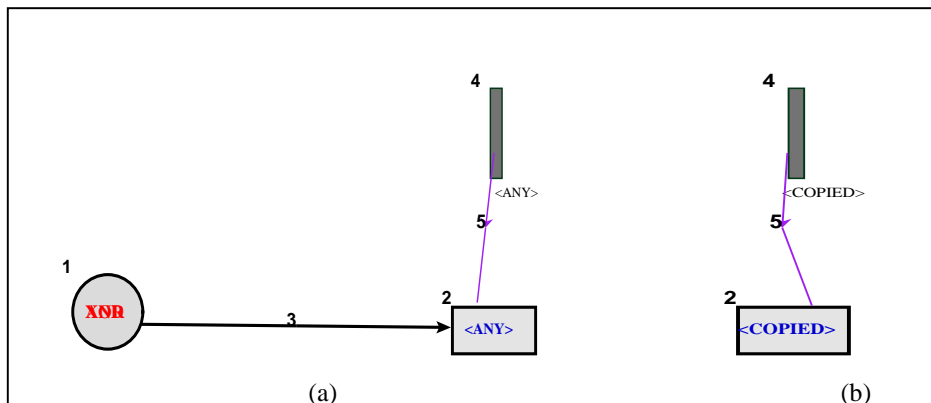


Figure 4.15: (a) LHS de la règle 10 (b) RHS de la règle 10

- **Règle 11: rule\_Del\_Activities (priority 11):** supprime toutes les activités après la fin de la transformation.

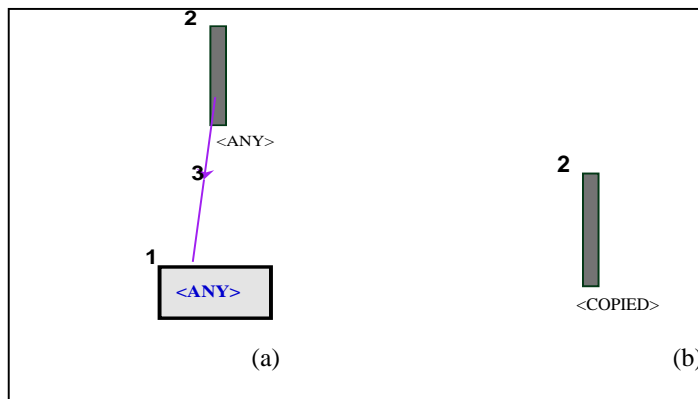


Figure 4.16: (a) LHS de la règle 11 (b) RHS de la règle 11



#### 4.5.4 Exemple

Pour illustrer notre approche, nous avons utilisé notre outil pour générer le processus métier montré dans la figure 4.1.

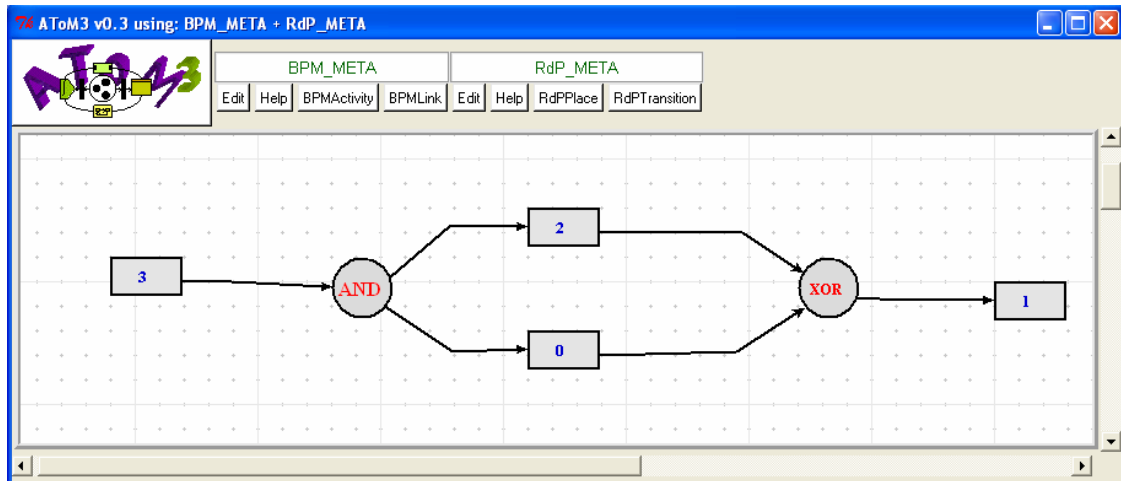


Figure 4.17: Exemple de processus métier.

Ensuite, nous avons appliqué notre méthode à cet exemple (figure 4.17) et nous avons obtenu, de manière automatique, le réseau de Petri équivalent comme c'est montré dans la figure 4.18.

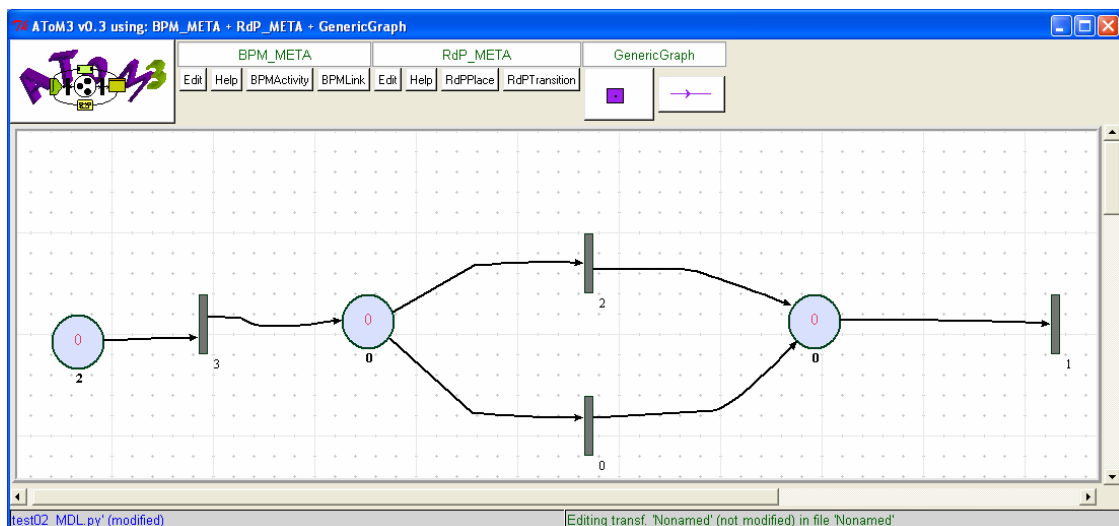


Figure 4.18: Le réseau de Petri généré du processus métier de la figure 4.17

#### 4.6. Utilisation de l'analyseur INA

Le but de transformer les processus métiers vers les RdPs est de pouvoir les analyser. Pour ce faire, nous avons choisi l'outil d'analyse INA (Integrated Net Analyser) présenté dans le chapitre 2.

Un des plus grands avantages des RdPs est leur représentation graphique des modèles. Cependant, l'outil INA n'est pas graphique ce qui oblige l'utilisateur à transformer *manuellement* la représentation graphique de ses modèles vers une description textuelle. Ceci peut être la source de nombreuses erreurs durant le processus de transformation.

Construire un outil visuel pour INA à partir de zéro est prohibitif. Pour cela nous avons fait appel à la méta modélisation qui s'avère très utile dans ce genre de problèmes puisqu'elle permet de modéliser (parfois graphiquement) les formalismes eux-mêmes [De Lara04].

Dans ce qui suit, nous proposons un Meta-modèle de l'outil INA. Nous utilisons AToM<sup>3</sup> pour la génération automatique de l'outil visuel de modélisation permettant le traitement des modèles dans le formalisme INA [Elmansouri08d]. Nous définissons une grammaire de graphes pour traduire les modèles créés dans l'outil généré vers la description textuelle dans la spécification INA. Finalement l'outil INA sera utilisé pour analyser la spécification obtenue.

##### 4.6.1 Méta modélisation des RdPs

Le méta-modèle des RdPs est déjà donné dans la section 4.5.1.2 (figure 4.4).

Etant donné ce méta-modèle, nous avons utilisé AToM<sup>3</sup> pour générer un environnement graphique pour les modèles INA. La figure 4.19 montre l'environnement graphique généré ainsi que la boîte de dialogue pour éditer une place.

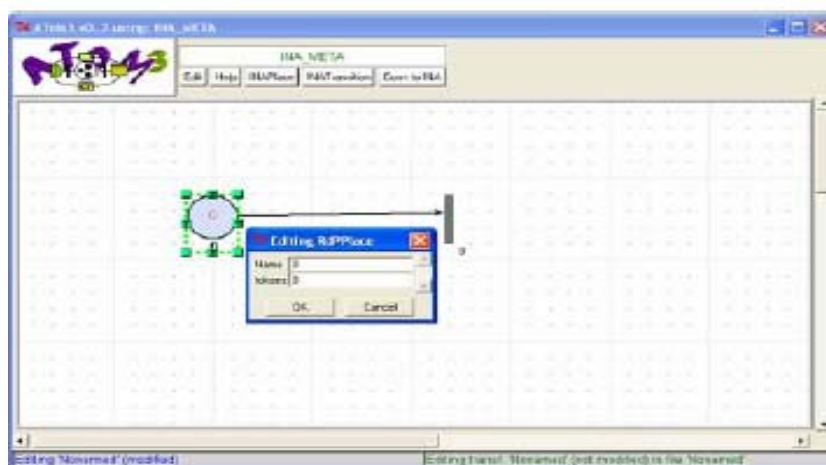


Figure 4.19 : Environnement graphique généré pour les modèles RdPs

#### 4.6.2 Génération de la spécification INA [Elmansouri08d]

Afin d'analyser les modèles INA, il est nécessaire de les traduire vers leurs équivalents dans la spécification INA. Dans cette section on va montrer comment utiliser l'environnement graphique généré dans la section précédente. Ceci est réalisé en définissant une grammaire de graphes de sept règles. L'avantage d'utiliser les grammaires de graphes pour la génération du code textuel est la représentation graphique et la programmation haut niveau.

Les règles sont appliquées dans un ordre ascendant par le système de réécriture jusqu'à l'épuisement de toutes les règles. Dans ce cas, nous sommes concernés par la génération automatique de code. Donc aucune règle ne va changer les modèles INA (c.à.d dans chacune des six règles, la partie gauche est identique à la partie droite). Les règles sont présentées dans la figure 4.21 et décrites comme suit :

- **Règle 1: *genListOfPreTransitions (priority 1)***: appliquée pour localiser une transition (non encore traitée) qui soit reliée à la place courante par un arc sortant (i.e. une pré-transition à une place courante). Elle associe un numéro à cette transition (si ce n'est pas déjà fait) et génère la spécification INA correspondante.
- **Règle 2: *genListOfPostTransitions (priority 2)***: Appliquée pour localiser une transition (non encore traitée) qui soit reliée à la place courante par un arc entrant (i.e. une post-transition à une place courante). Elle associe un numéro à cette transition (si ce n'est pas déjà fait) et génère la spécification INA correspondante.

- **Règle 3: *EndOfPlaceDescription* (priority 3):** Appliquée pour localiser une place courante dont le traitement est terminé. Elle est marquée comme "Visited".
- **Règle 4: *InitialiseTransition*(priority 4):** Appliquée pour localiser et initialiser les attributs temporaires dans les transitions pour traiter la prochaine place.
- **Règle 5: *SelectPlaceToDescribe*(priority 5):** Cette règle est appliquée afin de sélectionner une place non encore traitée. Elle associe un numéro à cette place et génère dans un fichier INA le nombre associé à cette place ainsi que son marquage.
- **Règle 6: *genPlaceInformation* (priority 6):** Appliquée pour générer la section information de la place dans la spécification INA Pour chaque place, cette règle génère le numéro associé, le nom, la capacité et le temps (s'il y a lieu).
- **Règle 7: *genTransitionInformation* (priority 7):** Appliquée pour générer la section information transition dans la spécification INA Pour chaque transition, cette règle génère le numéro associé, nom, la priorité et le temps (s'il y a lieu).

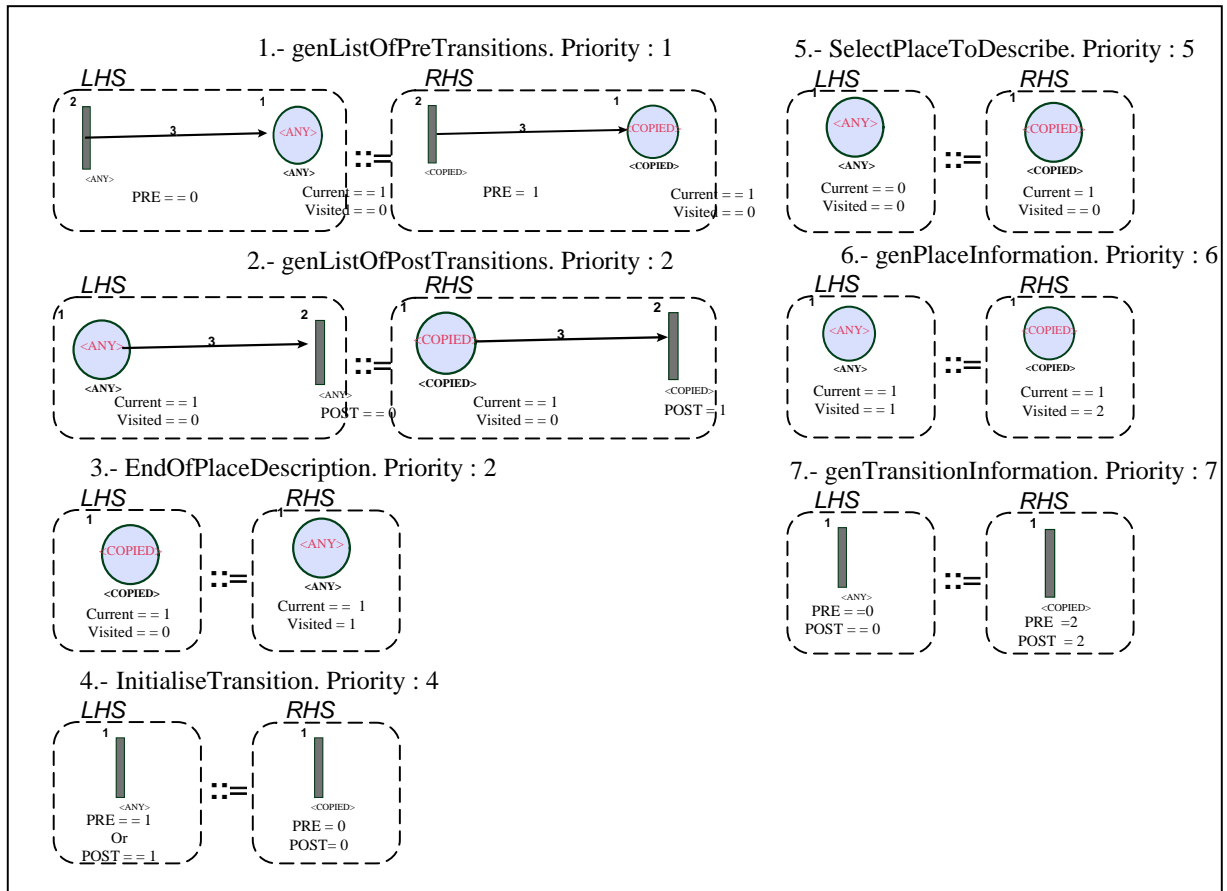


Figure 4.20: grammaire de graphes pour la génération la spécification INA à partir d'une représentation graphique

La grammaire possède également une action finale qui détruit les attributs temporaires des entités et ferme les fichiers de sortie. Finalement, nous avons assigné l'exécution de cette grammaire de graphes à un bouton étiqueté "Conv to INA" (figure 4.19).

### 4.6.3. Exemple

Pour valider notre grammaire, nous l'avons appliqué sur l'exemple des trois programmeurs présenté dans le chapitre 2 (figure 2.11).

D'abord, nous avons utilisé notre outil pour dessiner le modèle réseau de Petri des trois programmeurs (figure 4.21). Ensuite, nous avons appliqué la grammaire décrite ci-dessus et en utilisant l'outil que nous avons généré sur cet exemple et nous avons obtenu automatiquement la spécification INA suivante de la figure 4.22.

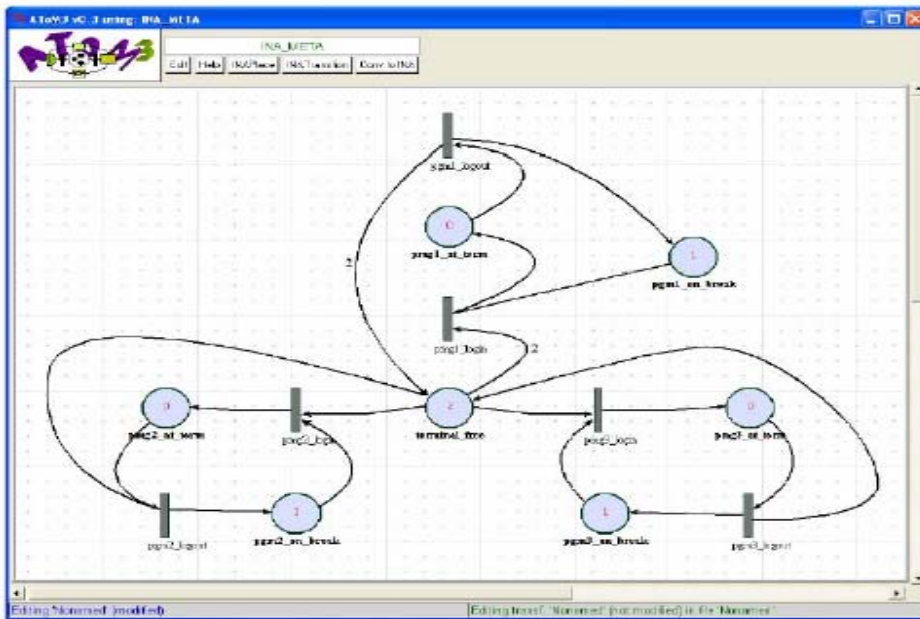


Figure 4.21: représentation graphique de l'exemple créé par notre outil

```

3_prog_2_term - Bloc-notes
Fichier Edition Format Affichage ?
P M PRE,POST NETZ 1:3_prog_2_term
0 2 0:2 1 2 , 3:2 4 5
1 0 3 , 0
2 1 0 , 3
3 0 5 , 2
4 0 4 , 1
5 1 1 , 4
6 1 2 , 5
@
place nr. name capacity time
0: terminal_free 00 0
1: pgm1_at_term 00 0
2: pgm1_on_break 00 0
3: pgm2_at_term 00 0
4: pgm3_at_term 00 0
5: pgm3_on_break 00 0
6: pgm2_on_break 00 0
@
trans nr. name priority time
0: pgm1_logout 0 0
1: pgm3_logout 0 0
2: pgm2_logout 0 0
3: pgm1_login 0 0
4: pgm3_login 0 0
5: pgm2_login 0 0
@
    
```

Figure 4.22 : Spécification INA de l'exemple de la figure 4.21.

#### 4.7. Etude de cas : Détection de la situation d'interblocage dans un processus métier

Nous avons appliqué notre approche proposée sur l'exemple du processus métier de la figure 4.1 en trois étapes consécutives:

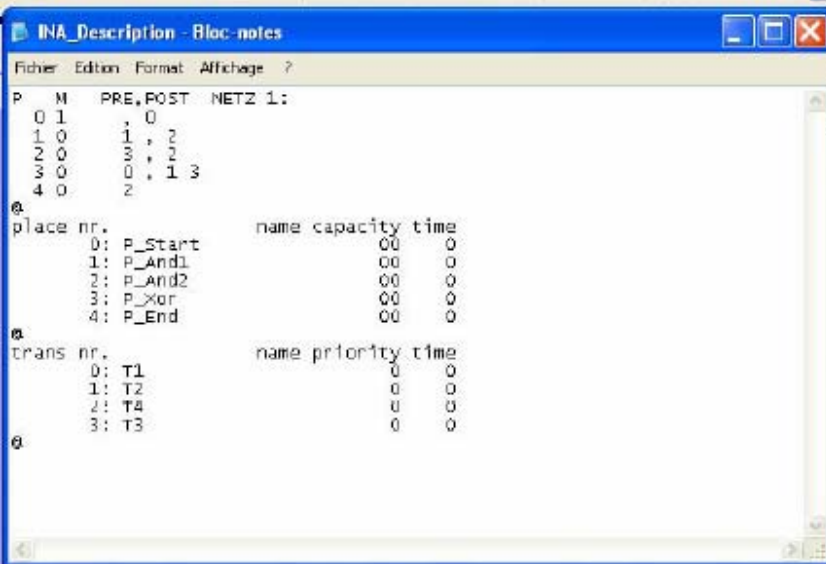
- 1) Transformation du processus métier vers son modèle RdP équivalent sous forme graphique.
- 2) Transformation du modèle graphique obtenu dans l'étape 1 en une spécification INA sous forme textuelle.
- 3) Application de l'outil INA sur la spécification obtenue dans l'étape 2 et interprétation des résultats.

- **Transformation du processus métier vers son modèle RdP équivalent**

Nous avons appliqué notre outil sur le processus métier de la figure 4.1 et nous avons obtenu automatiquement son modèle RdP équivalent (figure 4.2).

- **Transformation du modèle graphique obtenu dans l'étape 1 en une spécification INA sous forme textuelle.**

Nous avons ensuite appliqué notre approche sur le RdP de la figure 4.2 et nous avons obtenu sa spécification INA équivalente (figure 4.23).



```
Fichier Edition Format Affichage ?
P M PRE,POST NETZ 1:
0 1      0
1 0      1 2
2 0      3 2
3 0      0 1 3
4 0      2
@
place nr.      name capacity time
0: P_Start      00 0
1: P_And1      00 0
2: P_And2      00 0
3: P_Xor      00 0
4: P_End      00 0
@
trans nr.      name priority time
0: T1      0 0
1: T2      0 0
2: T4      0 0
3: T3      0 0
@
```

Figure 4.23 : Spécification INA du modèle RdP de la figure 4.2





The net is not statically conflict-free  
The net is pure.  
The net is ordinary.  
The net is homogenous.  
The net is not conservative.  
The net is subconservative.  
The net is structurally bounded.  
The net is bounded.  
There are no proper semipositive T-surinvariants.  
The net is not live.  
The net is not live and safe.  
The net is not a state machine.  
The net is free choice.  
The net is extended free choice.  
The net is extended simple.  
The net has places without pre-transition.  
The net is not state machine decomposable (SMD).  
The net is not state machine allocatable (SMA).  
The net is not strongly connected.  
The net is not covered by semipositive T-invariants.  
The deadlock-trap-property is not valid.  
The net has places without post-transition.  
The net is marked.  
The net is marked with exactly one token.

**Figure 4.24:** Résultat de l'application de l'outil INA sur le fichier de la figure 4.21

### **Interprétation des résultats**

En analysant le résultat "The net is not live and the *deadlock-trap-property* is not valid", nous pouvons déduire que le modèle du processus métier de la figure 4.1 contient une situation d'interblocage [Elmansouri08c].

#### **4.8. Conclusion**

Dans ce chapitre nous avons proposé une approche automatique basée sur la méta modélisation et les grammaires de graphe afin de générer des réseaux de Petri pour des processus métier en utilisant AToM<sup>3</sup>.

Le choix des réseaux de Petri comme modèle cible de nos transformations est justifié non seulement par leurs nombreuses techniques d'analyse, mais aussi par leur simplicité et leur capacité expressive due à la représentation graphique des modèles.

INA est un outil utilisé pour l'analyse des réseaux de Petri. Il utilise une spécification textuelle nécessitant ainsi une traduction des modèles graphiques vers du texte. La deuxième partie de ce chapitre était donc consacrée à la génération automatique d'un environnement graphique pour l'analyseur INA. Ceci a été fait aussi grâce à la méta modélisation et aux grammaires de graphes. Nous avons terminé ce chapitre par l'application de notre approche sur une étude de cas dans laquelle nous détecté une situation d'interblocage

## Chapitre 5

# Une approche intégrée UML/ECATNets pour la modélisation et la vérification des processus métiers dans les entreprises virtuelles

### 5.1. Introduction

L'entreprise virtuelle (voir chapitre 1) est un paradigme prometteur pour l'entreprise moderne. La caractéristique essentielle de l'EV est l'exécution distribuée et parallèle des processus métiers par les entités de chaque membre de l'EV. D'où le besoin d'une modélisation efficace des processus métiers avec un modèle strict d'analyse.

Le but de ce chapitre est de présenter une approche intégrée UML-ECATNets pour la modélisation des processus métiers dans les entreprises virtuelles.

Un aperçu sur les travaux similaires relatifs à la vérification et la transformation des diagrammes UML, sera donné.

Enfin, nous présenterons notre transformation automatique des *diagrammes de séquences* vers les *ECATNets* qui est basée sur les grammaires de graphes et réalisée à l'aide de l'outil AToM<sup>3</sup>.

### 5.2. Les processus métiers dans l'EV

Les processus métiers de l'entreprise virtuelle peuvent être définis comme suit :

- Des processus métiers de l'entreprise virtuelle (PM-EV): un ensemble d'activités liées qui sont distribuées aux membres de l'entreprise virtuelle et qui réalisent collectivement l'objectif de l'entreprise virtuelle.
- Sous-processus métiers des membres de l'entreprise (SP-ME-EV): un ensemble d'activités effectuées par un membre de l'entreprise.
- Relation entre le PM-EV et les SP-ME-EV: le sous-processus métier de chaque membre est une partie du processus métier global de l'entreprise virtuelle. Combinés tous ensemble, ils font émerger le processus métier global de l'entreprise virtuelle.

L'entreprise virtuelle est un cas particulier des organisations. Le caractère distribué des processus métiers de l'EV, requiert aussi bien des membres indépendants que coopérants. Les processus métiers possèdent les caractéristiques suivantes [gou03] :

- 1- *Distribués versus interdépendants*: D'une part, les activités de la VE-DBP sont distribués et exécutés par les différentes entreprises membres de l'entreprise qui sont géographiquement dispersés et organisationnellement séparés. D'autre part, il existe des relations logiques et temporelles entre certaines activités distribuées aux différents membres de l'entreprise.
- 2- *Indépendants versus coordonnés*: D'une part, les membres d'une entreprise virtuelle sont des entités indépendantes et économiquement autonomes. Non seulement ils possèdent leurs propres méthodes et procédures afin d'exécuter leurs activités de façon autonome, mais aussi ils s'efforcent de maintenir l'indépendance et le secret de leurs propres sous-processus. D'autre part, afin d'accomplir avec succès le processus métier global et de réaliser les objectifs communs de l'entreprise virtuelle, les sous-processus métiers des membres doivent être coordonnés de manière adéquate et opportune.

### 5.3. Modélisation des processus de L'EV

En raison des propriétés de répartition et d'indépendance du BP-VE, la modélisation des sous-processus devrait être entreprise par les membres eux-mêmes. En même temps, à cause des propriétés de l'interdépendance et de coordination du BP VE, cette modélisation effectuée par les membres, doit respecter les contraintes acceptées par tous les adhérents. Du fait que les inter-opérations entre les sous-processus métiers sont les causes potentielles de diverses erreurs (telles que l'interblocage), il est très important d'analyser le processus métier avant son exécution.

La panoplie de diagrammes de modélisation qu'offre UML (voir chapitre 1) permet de satisfaire les différents niveaux de modélisation des processus métiers de l'EV, depuis la description générale jusqu'à la conception détaillée. Cependant le manque d'outils d'analyse rigoureux dans UML peut réduire sa capacité. Pour cela nous recourons aux réseaux de Petri pour remédier à cet inconvénient.

On peut constater que les réseaux de Petri et UML possèdent des caractéristiques réciproques:

- UML est convivial alors que les réseaux de Petri possèdent une rigueur formelle.
- UML peut décrire les systèmes de façon efficace pendant que les RdPs peuvent les analyser strictement.
- Le modèle UML peut être facilement implémenté tandis que le modèle des RdPs est plus adéquat pour la simulation.

Donc pour tirer profit des avantages de UML et des RdPs, nous proposons une méthode intégrée UML – ECATNets pour la modélisation des processus métiers dans les EVs.

Le choix des ECATNets est motivé par leur sémantique basée sur la logique de réécriture et leur expression dans le langage MAUDE (voir chapitre 2)

#### 5.4. Description de l'approche

Notre approche suit le cycle de vie du processus unifié (UP) et se décrit comme suit [Elmansouri08a]:

- *Modèle des cas d'utilisation* : Expose les cas d'utilisation et leurs relations avec les utilisateurs
- *Modèle d'analyse* : Détaille les cas d'utilisation et procède à une première répartition du comportement du système entre divers objets
- *Modèle de conception* : Définit la structure statique du système sous forme de sous systèmes, classes et interfaces ; définit les cas d'utilisation réalisés sous forme de collaborations entre les sous systèmes, les classes et les interfaces
- *Transformer les diagrammes UML obtenus vers les ECATNets* pour l'analyse et la simulation. Le feedback du résultat de l'analyse et de la simulation sera transmis aux concepteurs pour d'éventuelles modifications et améliorations.
- *Modèle d'implémentation*: Intègre les composants (code source) et la correspondance entre les classes et les composants. Il sera fait selon les diagrammes améliorés obtenus dans la phase 4.
- *Modèle de test* : Décrit les cas de test vérifiant les cas d'utilisation.

La méthode proposée est guidée par les cas d'utilisation puisqu'elle suit le cycle de vie UP (figure 5.1).

##### 5.4.1. Modèle des cas d'utilisation :

Tout d'abord, les diagrammes des cas d'utilisation sont utilisés pour saisir et décrire la fonctionnalité de chaque activité. Les activités sont généralement d'une abstraction élevée, afin de ne pas succomber dans les détails trop tôt.

Nous pouvons relier les cas d'utilisation par les diagrammes de séquences ou de collaboration pour décrire les relations entre les objectifs des fonctionnalités.

### 5.4.2. Modèle d'analyse

Le diagramme de classes est un modèle permettant de décrire de manière abstraite et générale les liens entre objets.

UML permet de définir trois types de stéréotypes pour les classes :

a) *les classes "frontière" ou "interface" (boundary)*: classes qui servent à modéliser les interactions entre le système et ses acteurs.

b) *les classes "contrôle" (control)* : classes qui servent à représenter la coordination, le séquençement, les transactions et le contrôle d'autres objets.

c) *les classes "entité" (entity)* : classes qui servent à modéliser les informations durables et persistantes.

Chaque cas d'utilisation est réalisé en utilisant un ensemble d'objets. Chaque objet appartient à une classe de type (Frontière, contrôle ou entité).

### 5.4.3. Modèle de conception

Un ensemble de diagrammes UML est utilisé pour spécifier et concevoir le modèle statique (par les diagrammes de classes et des paquetages) et le modèle dynamique (par les diagrammes de séquences, diagrammes de collaboration, les diagrammes d'activité et les diagrammes d'état) du processus métier de l'EV.

*Les diagrammes de classes* sont utilisés pour décrire les objets métiers (y compris les propriétés et opérations) et de leurs associations. Les paquetages sont utilisés pour décrire les modes d'organisation de l'EV. On peut soit organiser tous les objets d'un membre de L'EV participant à l'exécution du processus métier de l'EV dans un paquetage, soit mettre dans des paquetages, les objets métiers appartenant à un certain domaine dans l'EV (par exemple, la conception des produits, la fabrication de produits).

*Les diagrammes de séquence* et *les diagrammes de collaboration* sont utilisés pour décrire les interactions entre les objets métiers. Basées sur un mécanisme de messages, les transitions d'état des objets métiers sont déclenchées par des transferts de messages, qui peuvent être considérés comme des événements. Les objets métiers peuvent également s'auto envoyer des messages, donc de changer leurs propres états.

*Les diagrammes d'activité* sont utilisés pour décrire les activités métiers et leurs transitions d'un point de vue global de l'ensemble des processus métiers. Comme les activités correspondent à des opérations dans les diagrammes de classes, une telle décomposition a effectivement été achevée après la mise en place de la hiérarchie de classe. Ici les diagrammes d'activité sont concentrés sur les relations dynamiques entre les activités.

Les diagrammes d'état transition sont utilisés pour décrire les états et les transitions dans un objet métier. Dans notre cas, les états correspondent également à des opérations dans les diagrammes de classes. Tous les états des diagrammes d'états de tous les objets constituent l'ensemble des activités métiers. Donc, les diagrammes d'états décrivent les interactions des activités du point de vue de l'objet.

#### **5.4.4. Transformation des diagrammes UML vers les ECATNets**

Vu l'absence d'approches d'analyse du modèle UML, nous proposons dans notre démarche de le transformer vers le modèle des ECATNets (chapitre2) pour l'analyse et la simulation. Le mapping est exécuté niveau par niveau en fonction de la hiérarchie de classes dans le modèle UML. Le processus de transformation peut se faire de manière automatique en utilisant l'outil ATOM3 se basant sur les transformations de graphes (chapitre3). Une fois le modèle ECATNets obtenu, nous pouvons utiliser les techniques d'analyse des ECATNets pour l'analyse qualitative stricte. Ensuite, on utilise la simulation sur ordinateur pour l'analyse quantitative [Boudiaf05]. Le résultat de l'analyse et de la simulation est remis aux concepteurs. Ce processus peut être répété dans le but d'améliorer le modèle UML.

Dans le cadre de cette thèse, nous nous sommes intéressé aux diagrammes de séquence dont nous avons réalisé la transformation automatique vers les ECATNets. Ceci sera détaillé dans la section 5.6.

#### **5.4.5. Modèle d'Implémentation**

Le modèle d'Implémentation est exécuté selon le modèle UML amélioré obtenu lors de l'étape précédente. Comme UML est le standard du logiciel international, il est facile de traduire les diagrammes UML (diagrammes de classes et diagrammes de séquence par exemple) vers les codes sources de langages de programmation OO. Nous recommandons l'utilisation du logiciel Rational Rose.

#### **5.4.6. Modèle de test**

Les cas de test sont tirés à partir des diagrammes des cas d'utilisation.

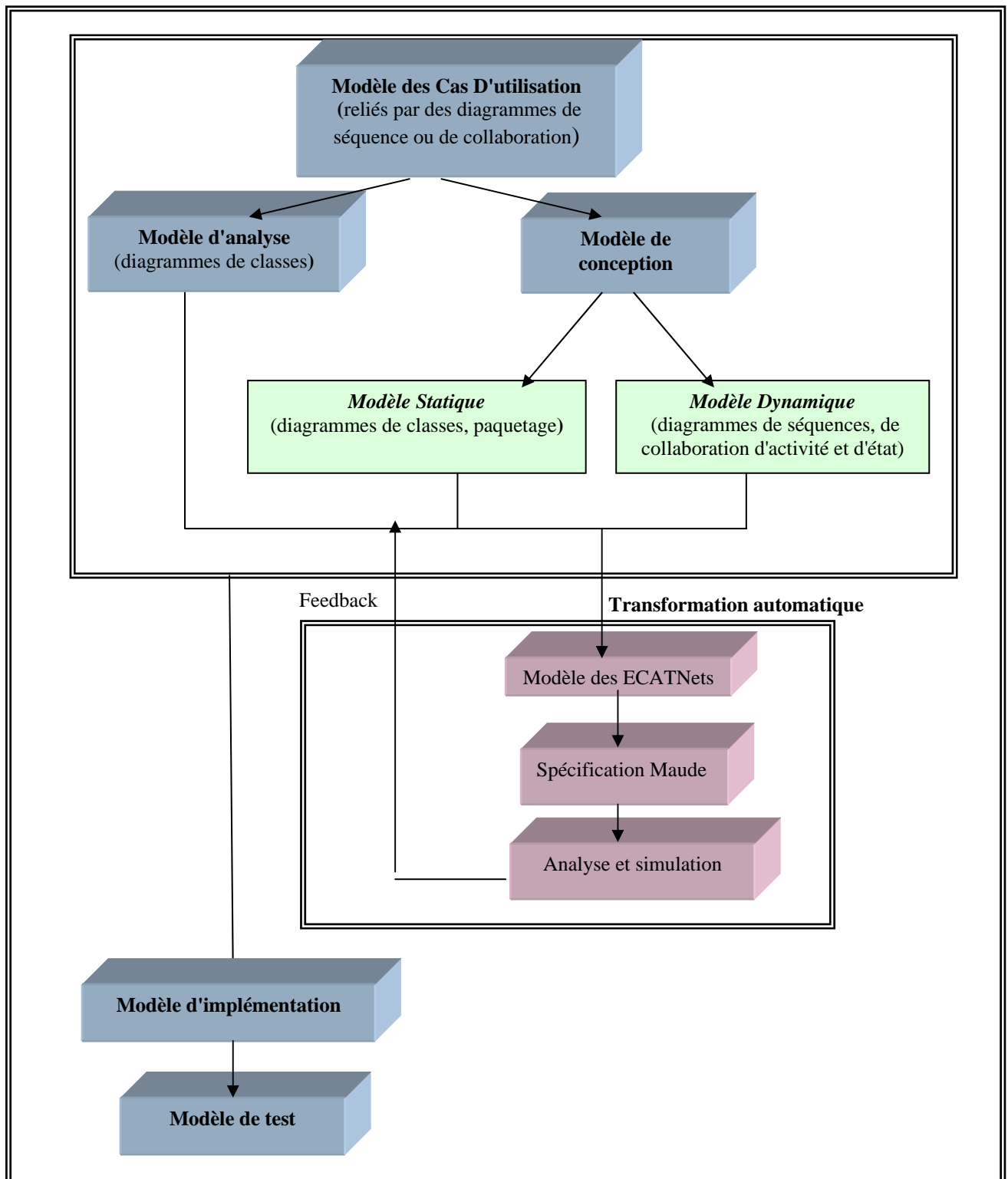


Figure 5.1: Approche de modélisation des processus métiers dans les EVs



## 5.5. Techniques formelles appliquées à UML

Avant de détailler notre transformation des diagrammes de séquences vers les ECATNets, nous présentons dans ce qui suit un aperçu sur les travaux similaires.

Le langage de modélisation unifié UML est devenu le standard de facto dans l'industrie pour la modélisation objet. Il occupe une place importante dans la modélisation des processus métiers. Cependant du fait qu'il manque d'outils d'analyse rigoureux et de techniques de preuve formelle, plusieurs travaux ont essayé d'appliquer des techniques ayant une base formelle à des spécifications UML. Les approches adoptées sont diverses. On cite ci-après quelques unes [LEGuenne01].

### 5.5.1. Génération de tests

Dans [Fleisch99] est présentée une méthodologie de test qui s'appuie sur des diagrammes décrivant la structure du système en utilisant la notation ROOM et sur un ensemble de cas d'utilisation accompagnés de scénarii sous forme de diagrammes de séquences UML. Les diagrammes décrivant le modèle sont compilés en un modèle de simulation.

Dans [Fröhlich00] une technique est proposée pour générer des tests à partir de cas d'utilisation. Les cas d'utilisation sont transformés en des diagrammes d'états-transition. Ces diagrammes sont alors eux-mêmes réécrits dans un langage de planification, comme ensemble de contraintes. L'outil graphplan permet de générer des tests en résolvant les contraintes de planification.

Offutt [Offutt99] présente une technique permettant de générer des jeux de données permettant de tester les différentes conditions apparaissant dans les gardes des transitions des machines à états.

### 5.5.2. Approches traductionnelles de la formalisation d'UML

Plusieurs formalismes ayant des bases formelles permettent de spécifier des systèmes et d'appliquer des techniques comme la vérification, la simulation, ou la synthèse de tests. Cependant, ces langages ne sont généralement pas facilement utilisables par des développeurs de logiciels pour lesquels UML est déjà largement employé. Donc il semble être très tentant de traduire un sous-ensemble d'UML vers un de ces langages. Ceci permet en effet à la fois :

- de donner indirectement une sémantique au sous-ensemble en question;
- d'utiliser les techniques formelles offertes par l'outil associé.

De nombreux travaux ont adopté cette approche traductionnelle:

- *Traduction vers Promela*

Le modèle UML est traduit dans le langage PROMELA [Holzman91] qui constitue le langage de spécification du model checker SPIN [Holzmann90], [Holzmann97]. La limite de cet outil réside dans l'incapacité du model checker à vérifier les systèmes réactifs ouverts. Cette approche a notamment été adoptée par [Paltor99], [Lilius99] et [Latella99]

- *Traduction vers SMV*

L'approche présentée dans [Kwon00] permet de faire du model-checking à partir des diagrammes d'états-transitions de UML, grâce à une traduction en SMV [McMillan92]. Cette approche ne considère qu'un seul diagramme d'états-transitions, en isolation, sans envisager de communications inter-objets.

- *Traduction vers LOTOS*

Une traduction d'un sous ensemble de UML vers le langage de spécification LOTOS [ISO85] est proposée dans l'article [Paulo00]. Cette traduction permet d'utiliser la boîte à outils CADP [Fernandez92], [Fernandez96], [Garavel98] sur les spécifications OBLOG.

- *Traduction vers SDL*

De même l'AGL Telelogic Tau (Telelogic03) permet la validation de modèles temps réel UML en offrant un passage par le langage SDL et une validation par model checking.

### 5.5.3. Les approches "méta"

Par opposition aux approches traductionnelles mentionnées ci-dessus, un autre courant vise à formaliser UML à l'aide d'UML lui-même, en modélisant le domaine sémantique d'UML en UML. On retrouve dans cette catégorie certains travaux initiés par le groupe *precise UML* (pUML), comme [Clark01] ainsi que la proposition du groupe de travail sur l'*Action Semantics* [OMG03] qui travaille sur l'élaboration de la sémantique de UML.

### 5.5.4. Utilisation de Réseaux de Petri

Une autre tendance dans la recherche, consiste à combiner UML et les réseaux de Petri. Delatour et Paludetto [Delatour99] présentent une méthodologie d'analyse et de développement des systèmes temps réel, basée sur une utilisation conjointe et complémentaire de la notation UML et des réseaux de Petri.

Peter King et Rob Pooley présentent dans [King99] une transformation depuis UML vers les réseaux de Petri stochastiques. La transformation est illustrée à travers un exemple

d'un modèle du protocole qui gère les transactions dans une base de données distribuée. Les diagrammes d'état-transition et les diagrammes de collaboration sont combinés ensemble dans un même modèle pour représenter le comportement du système dans son intégralité. Le modèle résultat est transformé en un modèle réseau de Petri stochastique qui va être utilisé comme modèle d'entrée à l'outil de vérification SPNP qui va permettre l'analyse du modèle et de ses propriétés temporelles. Une extension de ce travail peut être repérée dans [King00] où sont utilisés les réseaux de Petri stochastiques généralisés (GSPN).

[Trowitzsch05] propose une transformation des modèles UML vers des réseaux de Petri stochastiques pour des buts d'évaluation quantitative des modèles (évaluation des propriétés temporelles des systèmes temps réel). Une extension de cette approche est présentée dans [Trowitzsch06].

Dans [LopezGrao04], les auteurs ont choisi comme modèle cible une extension des réseaux de Petri stochastiques à savoir les réseaux de Petri stochastiques étiquetés généralisés (LGSPN : Labeled generalized stochastic petri nets). Cette fois c'est le diagramme d'activité qui a été sujet à la transformation dans le but d'évaluer les performances des systèmes modélisés dans des phases précoces dans le cycle de développement.

Domokos et Majzik dans [Domokos] choisissent une autre catégorie des RDP; les réseaux de Petri temporisés afin de transformer les diagrammes de classe. D'abord, le diagramme de classe est transformé en un modèle intermédiaire. Ce dernier est ensuite transformé vers un réseau de Petri temporisé. L'outil de transformation utilisé est VIATRA.

Khriss, Elkoutbi et Keller [Khriss00] proposent un outil de prototypage basé sur les cas d'utilisation de la notation UML et des réseaux de Petri colorés. Quant à Saldhana et shatz [Saldhana01], ils développent une méthodologie de dérivation d'un ensemble de réseaux de Petri objets à partir de diagrammes d'états-transitions. Pour obtenir un seul modèle, le diagramme de collaboration sera utilisé afin de réunir l'ensemble des réseaux de Petri orientés objet pour former un seul réseau de Petri coloré. Ensuite n'importe quel outil d'analyse des réseaux de Petri colorés peut être utilisé pour simuler et analyser le modèle résultat. Un outil UML-CPN pour saisir la transformation entre modèles UML et CPN est défini dans [Zhaoxia04].

### 5.5.5. UML et la transformation de graphes

UML est un langage graphique et visuel. Les différents diagrammes utilisés dans sa spécification sont des graphes. Il ne semble donc pas étonnant que plusieurs travaux fassent recours aux transformations de graphes pour la transformation de modèles. Plusieurs

travaux s'intéressent à la formalisation ou l'enrichissement de la sémantique d'UML avec des transformations de graphes pour des buts de vérification et de validation.

Dans [Holscher06], les auteurs présentent une transformation du modèle UML en un système de transformations de graphes composé de règles de transformation et d'un graphe de travail représentant l'état courant du système modélisé ce qui permet aux modélisateurs UML d'exécuter leurs modèles. La simulation de l'exécution visuelle du système est effectuée en appliquant les règles de transformation sur le graphe de travail. Le but de ce travail est la validation des modèles UML dans des phases préliminaires avant toute implémentation du système. L'approche couvre les diagrammes : de classe, de cas d'utilisation, d'objet, d'état transition et d'interaction.

Dans le même contexte, [kuske02] présente une association des règles de transformation de graphes aux opérations dans le diagramme de classe et aux transitions dans le diagramme d'état-transition. Les différentes règles de transformation sont combinées dans un même système pour obtenir une description cohérente et unique de la sémantique.

Dans [Engels00] le diagramme de collaboration est interprété par un ensemble de règle de transformation ce qui permettra de spécifier une sémantique dynamique pour UML. Dans [Baldan05] un cadre de la vérification du diagramme d'activité est présenté à l'aide des règles de transformations de graphes.

### **5.5.6. Diagrammes de séquences et réseaux de Petri**

Le diagramme de séquences est un diagramme comportemental d'UML. Il permet de représenter l'interaction entre les objets dans un ordre chronologique où chaque objet possède une période de vie précise qui dépend des messages envoyés et reçus.

Dans le contexte des transformations des modèles UML vers les réseaux de Petri, peu de travaux ont considéré les diagrammes de séquence. [Merseguer03] discute l'intégration de la modélisation de performance dans le processus de développement de logiciels en se basant sur la transformation de la majorité des modèles comportementaux d'UML vers des réseaux de Petri stochastiques généralisés, inclus le diagramme de séquence. [Fernandes07] propose une transformation des diagrammes de cas d'utilisation et de séquences vers les réseaux de Petri colorés. La transformation est décrite à l'aide d'une étude de cas d'un contrôleur d'ascenseur.

[Ouardani06] propose une approche basée méta-modèle. La transformation du diagramme de séquences vers les réseaux de Petri est définie par un méta-modèle qui est déduit d'un méta-modèle source du diagramme de séquence et du méta-modèle cible

des réseaux de Petri. L'objectif de ce travail était d'automatiser partiellement la transformation permettant ainsi la vérification et la validation des exigences.

Enfin, [Guerra03] présente un framework pour la vérification des modèles UML. L'approche consiste à construire des méta-modèles pour différents diagrammes d'UML et de les transformer vers les des réseaux de Petri. La vérification des modèles obtenus se fait par model-checking. L'implémentation est faite en utilisant l'outil ATOM3.

### 5.5.7. Discussion

D'après [Balsamo01], peu d'approches de transformation de modèles UML utilisent les réseaux de Petri comme formalisme cible [King00, King99]. En plus, plusieurs approches reposent sur des études de cas pour exprimer leurs transformations [Baresi01, Trowitzsch06, King00, King99, Fernandes07, Baldan05]. Sinon, elles sont manuelles, favorisant ainsi les risques d'erreurs dans l'application des règles de transformation. Pour remédier à ces anomalies, [Ouardani06] utilise le concept de méta modélisation dans sa transformation, mais son approche est semi-automatique. D'autres approches utilisent les transformations de graphes pour augmenter la sémantique des modèles UML [kuske02], ce qui permet d'obtenir une description cohérente de la sémantique mais n'aide pas à la validation formelle de ces modèles. Pour cela, une catégorie d'approches transforme les modèles UML vers des systèmes de transformation de graphes [Holscher06], permettant ainsi la validation de ces derniers. Enfin dans [Guerra03], on trouve une combinaison entre la méta-modélisation et les transformations de graphes ce qui va permettre l'automatisation des transformations des modèles UML et l'élargissement de l'application de ces transformations à la totalité des modèles associés à un méta modèle donné. C'est justement dans cette dernière catégorie d'approches que se situe notre travail de thèse.

## 5.6. Transformation des diagrammes de séquence vers les ECATNets

Dans cette section, nous présentons une transformation des diagrammes de séquence vers les réseaux de Petri (ECATNets) basée sur la méta modélisation et la transformation de graphes. Pour ce faire, une grammaire de graphe est définie. La transformation est exécutée à l'aide de l'outil ATOM3.

### 5.6.1. Méta-modélisation des diagrammes de séquences et des ECATNets

Pour construire des modèles en AToM<sup>3</sup>, nous devons définir un méta-modèle pour eux. Le méta-formalisme utilisé dans ce travail est le modèle de diagrammes de classes d'UML. Les contraintes sont exprimées en code Paython [Python].

#### 5.6.1.1. Méta-modélisation des diagrammes de séquences

Pour méta-modéliser les diagrammes de séquences nous avons proposé trois classes comme le montre la figure 5.2 [Chaoui09, Saadi09].

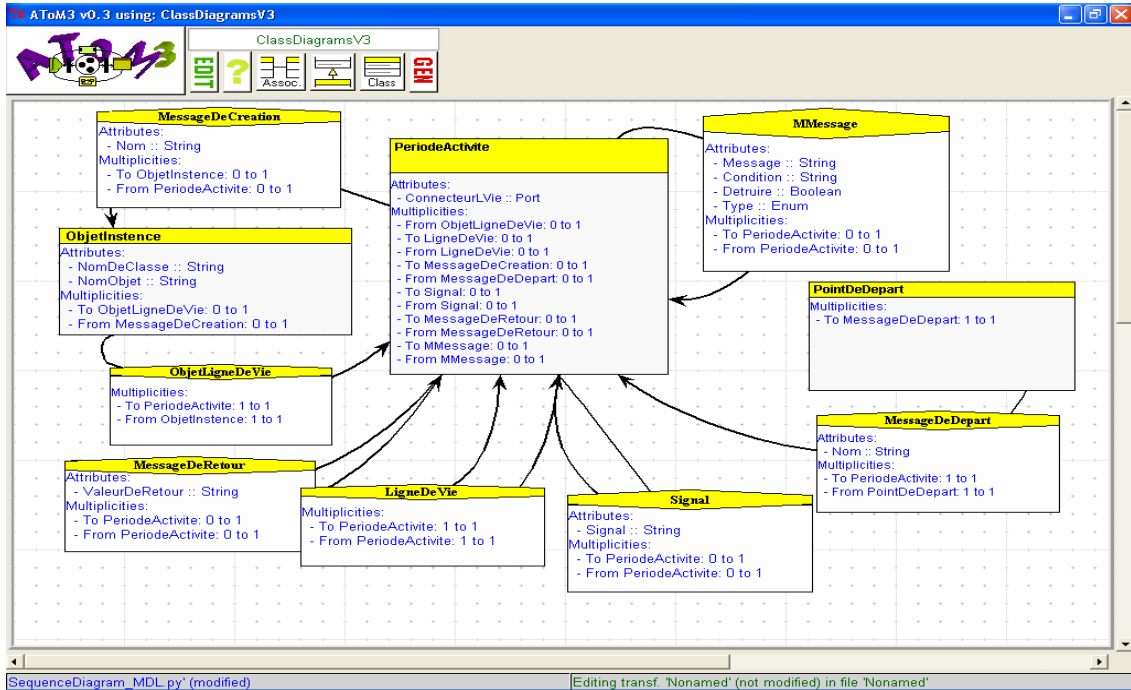


Figure5.2 : Méta-modèle des diagrammes de séquences.

- ❖ **Classe PointDeDepart** : cette classe représente le point de départ d'un diagramme de séquences. Elle est représentée visuellement par un carré de couleur grise. Elle est connectée avec la classe *PeriodeActive* par une association *MessageDeDepart*. Cette association possède un attribut Nom et relie une seule instance de la classe *PointDeDepart* avec une seule instance de la classe *PeriodeActive*. Elle est représentée graphiquement par une flèche de couleur bleue.
- ❖ **Classe ObjetInstance** : représente les objets interagissant dans le diagramme de séquences. Chaque objet possède un nom et le nom de sa classe. La classe *ObjetInstance* est reliée par l'association *ObjetLigneDeVie* avec la classe *PeriodeActive*. Elle relie un objet avec au plus une instance de la classe *PeriodeActive*. Un objet est visualisé à travers un rectangle bleu qui porte son nom et le nom de sa classe (Figure 5.3).

❖ **Classe *PeriodeActive*** : c'est la classe qui détermine la période d'activité d'un objet. Elle possède un attribut *ConnecteurLVie* de type Port qui joue le rôle d'un connecteur entre les différents segments de la période d'activité d'un objet. Deux périodes d'activités parvenant d'objets différents peuvent être reliées soit :

- Par un message (l'association *MMessage*) possédant un nom (attribut *Message*) et un type. Si le type est une condition alors il faut la définir dans l'attribut *condition*. Il peut également être un message de destruction si l'attribut *Detruire* porte la valeur "vrai". Dans notre méta-modèle, au plus deux périodes d'activités peuvent être reliées par un message.
- Par un signal (l'association *Signal*) qui possède un attribut *Signal* et qui peut avoir au plus une instance entre deux périodes d'activités.
- Par une association *MessageDeRetour* qui possède un attribut *ValeurDeRetour*. Un message de retour est une réponse à un message donné.
- Et finalement par une association *LigneDeVie*.

Une instance de la classe *PeriodeActive* est représentée graphiquement par un rectangle bleu. Elle peut être liée avec une instance de la classe *ObjetInstance* à travers un message de création (association *MessageDeCreation*) qui dispose d'un nom qui doit être unique.

Le méta-modèle ci-dessus va nous permettre de générer un outil de modélisation des diagrammes de séquence. La figure 5.3 présente l'outil généré avec un exemple de modélisation d'un diagramme de séquences.

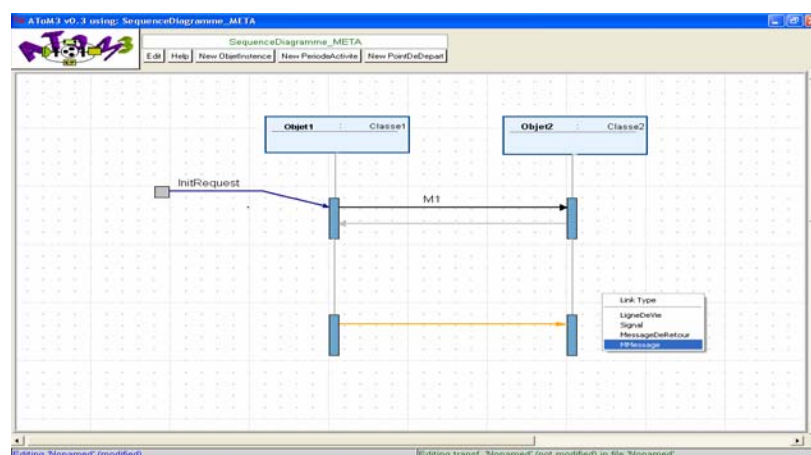


Figure 5.3 : Outil de modélisation des diagrammes de séquences.

### 5.6.1.2. Méta-modèle des ECATNets

Pour méta-modéliser les ECATNets, nous quatre classes ont été proposées comme le montre la figure 5.4 [Kerkouche08].

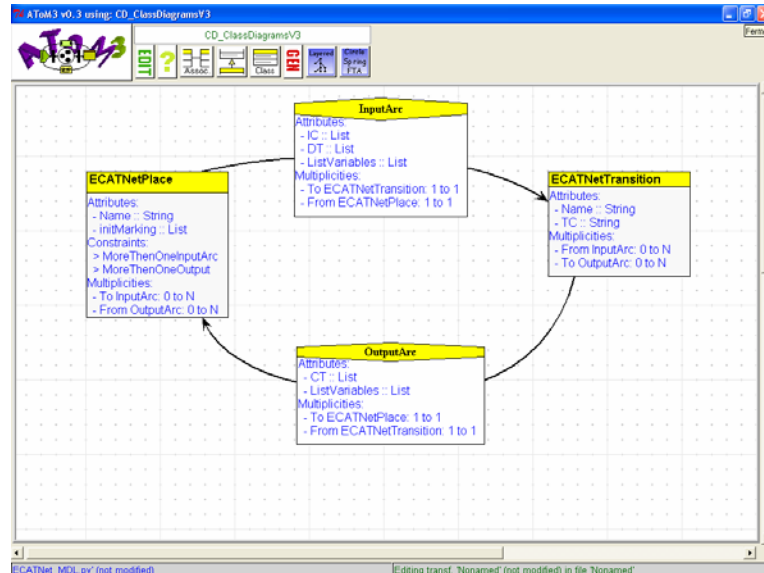


Figure 5.4 : Méta-modèle des ECATNets.

- ❖ **Classe ECATNetPlace** : indique une place visualisée à l'aide d'un cercle de fond gris. Elle possède un attribut *nom* qui est visuel et un autre attribut *initMarking* qui représente une liste de termes algébriques symbolisant le marquage de la place. Cet attribut est visualisé dans le cercle représentant la place. Pour chaque instance de cette classe, il y a deux contraintes qui interdisent l'existence de plus d'un arc d'entrée provenant de la même transition et de même pour l'arc de sortie.
- ❖ **Classe ECATNetTransition** : elle est visualisée par un rectangle de fond gris. Elle possède un attribut *nom* et un attribut *TC* (Transition Condition) qui sont visuels. Une instance de la Classe *ECATNetTransition* peut être reliée à une instance de la classe *ECATNetPlace* par un arc d'entrée ou de sortie.
- ❖ **Association InputArc** : elle relie la classe *ECATNetPlace* avec la classe *ECATNetTransition*. Elle est visualisée par une flèche bleue portant les deux attributs *IC* (Input Condition) et *DT* (Destroyed Tokens) qui représentent respectivement la liste des conditions d'entrée et les jetons détruits à partir de la place d'entrée.
- ❖ **Association OutputArc** : cette association permet de déterminer l'arc sortant d'une transition vers une place. Elle est visualisée à travers une flèche rouge portant



l'attribut *CT* (Created Tokens) où l'on trouve la liste des différents jetons créés dans la place destination de l'arc.

Grâce au méta-modèle des ECATNets, nous allons générer un outil de modélisation de ces derniers. L'outil généré va nous permettre d'éditer les différents modèles des ECATNets (figure 5.5) et de manipuler les modèles résultant de la transformation des diagrammes de séquences.

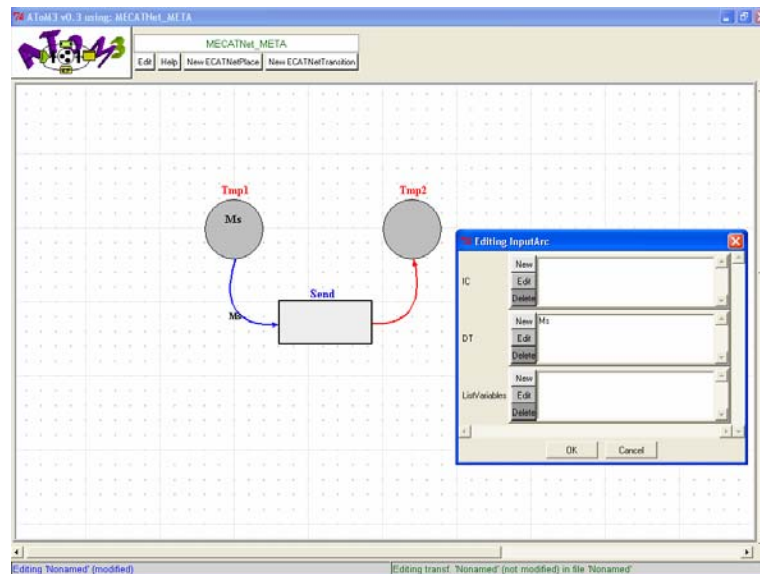


Figure 5.5 : Outil de modélisation des ECATNets

### 5.6.3. Grammaire de graphes pour la transformation des diagrammes de séquences vers les ECATNets

Afin de produire les formalismes ECATNets générés par notre outil, nous avons proposé une grammaire de graphes avec quinze règles qui seront exécutées dans un ordre ascendant. L'application de cette grammaire aux diagrammes de séquences créés par notre outil conduit à la génération du réseau de Petri équivalent. Ces règles sont réparties en cinq catégories :

1. Règles qui transforment l'ensemble des messages.
2. Règles qui renomment l'ensemble de transitions et créent la transition de réception.
3. Règles qui renomment l'ensemble de places.
4. Règles de transformation d'un message de retour.
5. Règles de suppression des périodes d'activités.

Pour plus de détails sur les règles on peut se référer à [Chaoui09, Saadi09].

## 5.7. Analyse des diagrammes de séquences

Une fois la transformation obtenue, on peut utiliser tous les outils d'analyse relatifs aux ECATNets. Pour la simulation on a choisi d'utiliser Maude. Pour cela, il est nécessaire de traduire ces modèles vers leurs modèles équivalents dans la syntaxe Maude. Une génération automatique de la spécification Maude des ECATNets se trouve dans [kerkouche08]. Les auteurs utilisent l'environnement de modélisation généré dans la section 5.6. Ceci est fait en définissant une grammaire de graphes pour générer le code Maude correspondant au modèle des ECATNets. La grammaire contient six règles [kerkouche08].

## 5.8. Exemples de transformation d'un diagramme de séquences vers un ECATNets

### 5.8.1 Exemple 1

Pour mettre en valeur la transformation réalisée et la grammaire définie, on a essayé de l'appliquer sur un exemple. La figure 5.6 présente un exemple de diagramme de séquences édité à l'aide de l'outil généré plus haut.

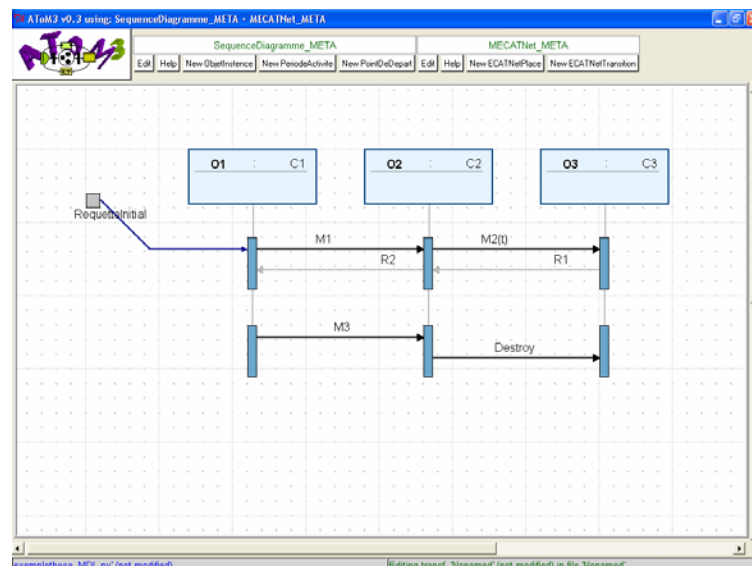


Figure 5.6 : Exemple d'un diagramme de séquences

Afin d'appliquer l'ensemble des règles sur l'exemple précédent, on doit exécuter la grammaire à l'aide de notre outil comme le montre la figure suivante.

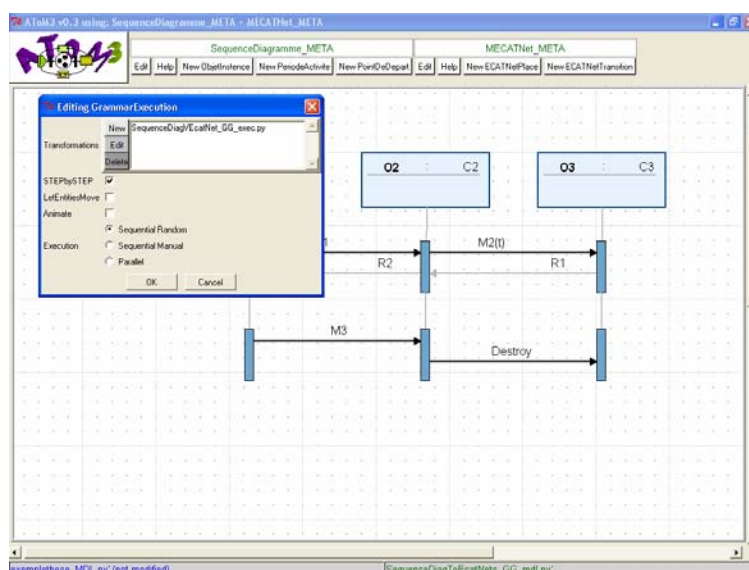


Figure 5.7 : Lancement de l'exécution de la grammaire.

L'exécution des règles est faite en respectant la priorité de chacune d'elles. La figure 5.8 présente un graphe intermédiaire contenant les éléments des deux formalismes source (*Diagramme de séquences*) et cible (*ECATNets*).

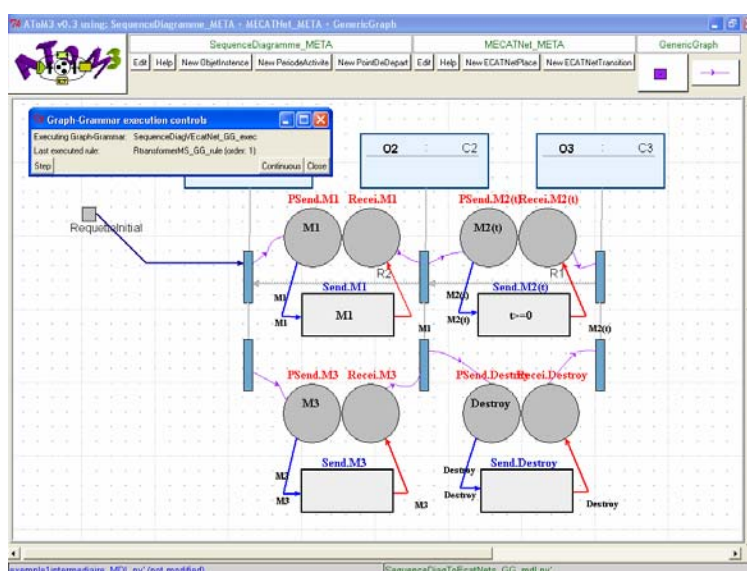


Figure 5.8 : graphe intermédiaire obtenu après l'exécution de la règle 1.

L'exécution des différentes règles nous permettra d'aboutir à un graphe final qui représente le modèle ECATNets équivalent au diagramme de séquences de la figure 5.6.

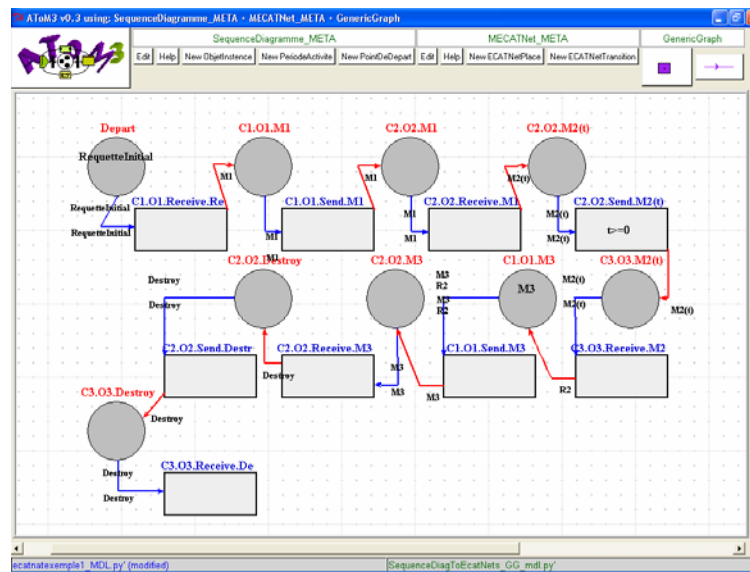


Figure 5.9 : ECATNets résultat de la transformation.

Pour analyser l'ECATNets résultant de la transformation, on va utiliser l'outil de vérification défini dans [Kerkouche08] qui génère automatiquement la description Maude équivalente. L'outil est présenté dans la figure suivante.

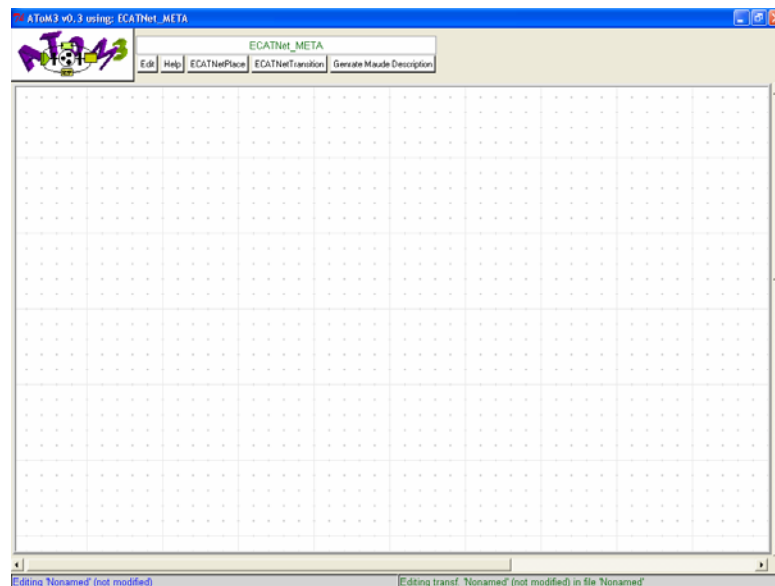


Figure 5.10 : outil de génération des spécifications Maude à partir des ECATNets

En appliquant l'outil précédent sur notre exemple, on obtient la spécification Maude présentée dans la figure 5.11.

```

in project
in basic-ecatnet

mod ECATNET-SYSTEM is

ops C1.O1.M1 C2.O2.M1 C2.O2.M2(t)
C3.O3.M2(t) C1.O1.M3 C2.O2.M3 C2.O2.Destroy
C3.O3.Destroy Depart : -> Place .
crl[C1.O1.Send.M1] : <C1.O1.M1;M1> =>
<C2.O2.M1;M1> if (M1) .
crl[C2.O2.Send.M2(t)] : <C2.O2.M2(t);M2(t)>
=> <C3.O3.M2(t);M2(t)> if (t>=0) .
rl[C1.O1.Send.M3] :
<C1.O1.M3;M3>.<C1.O1.M3;R2> =>
<C2.O2.M3;M3> .
rl[C2.O2.Send.Destroy] :
<C2.O2.Destroy;Destroy> =>
<C3.O3.Destroy;Destroy> .
rl[C1.O1.Receive.RequetteInitial] :
<Depart;RequetteInitial> => <C1.O1.M1;M1> .
rl[C2.O2.Receive.M1] : <C2.O2.M1;M1> =>
<C2.O2.M2(t);M2(t)> .
rl[C3.O3.Receive.M2(t)] :
<C3.O3.M2(t);M2(t)> => <C1.O1.M3;R2> .
rl[C2.O2.Receive.M3] : <C2.O2.M3;M3> =>
<C2.O2.Destroy;Destroy> .
rl[C3.O3.Receive.Destroy] :
<C3.O3.Destroy;Destroy> => .

endm .

rew <C1.O1.M3;M3>.<Depart;RequetteInitial>.

```

Figure 5.11: Spécification Maude équivalente à l' ECATNets présenté dans la figure 5.9

### 5.8.2 Exemple 2

Nous avons appliqué aussi notre outil sur le diagramme de séquences de la figure 5.12 et nous avons obtenu le modèle ECATNets de la figure 5.13.

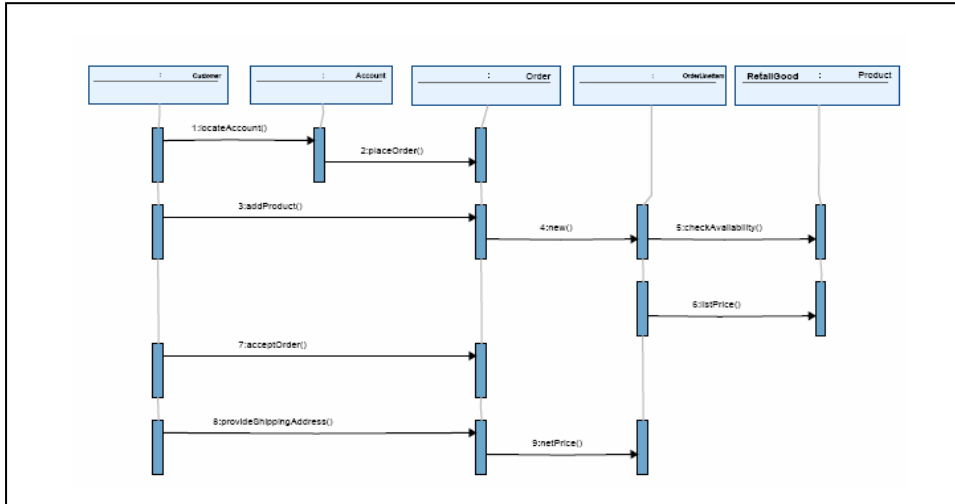


Figure 5.12 : diagramme de séquences.

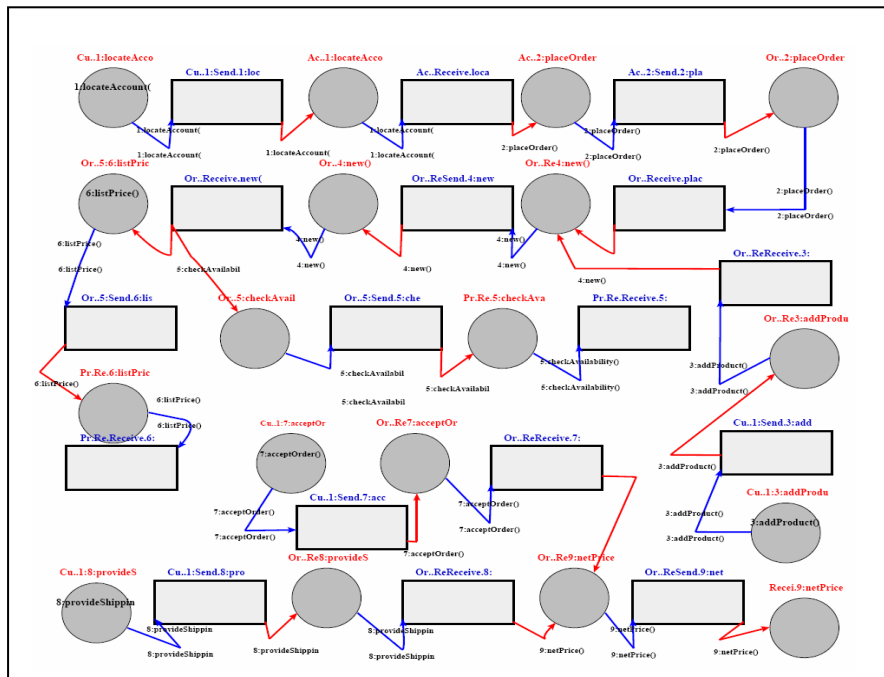


Figure 5.13: Modèle ECATNets résultat de la transformation du diagramme de séquence de la figure 5.12

## 5.9. Conclusion

Dans ce chapitre nous avons proposé une approche pour la modélisation des processus métiers dans les entreprises virtuelles. La méthode proposée suit le processus UP et utilise une intégration des réseaux de Petri (ECATNets) et d'UML. Ceci permet de tirer profit des avantages d'UML, le langage de modélisation standard, et des ECATNets comme outil d'analyse des propriétés.

Nous avons également donné un panorama sur les travaux similaires, notamment ceux concernant la vérification et la transformation des diagrammes UML.

Dans le reste du chapitre nous avons présenté une transformation automatique des *diagrammes de séquence* vers les *ECATNets*. Cette transformation se base sur les grammaires de graphes et a été réalisée à l'aide de l'outil AToM<sup>3</sup> (outils de méta-modélisation et de transformation de modèles).

Enfin, cette transformation a été illustrée par un exemple. Le résultat obtenu sous forme d'un *ECATNets* est automatiquement transformé, pour des fins de vérification, vers une spécification **Maude** à l'aide d'un autre outil de transformation qui est lui-même généré à l'aide de l'AToM3.

## Conclusion générale

Dans cette thèse nous nous sommes intéressé à la modélisation des entreprises et notamment aux processus métiers et leur vérification. Ceci se justifie parfaitement par le fait que la notion de processus a connu, durant les dernières années, un intérêt particulier dans la modélisation des entreprises afin de faire face aux défis du marché moderne. Donc pour éviter d'introduire des erreurs lors de la modélisation des processus métiers, il est nécessaire de les vérifier à une étape précoce dans le processus de développement de l'application. Le risque d'erreurs devient plus grand lorsqu'il s'agit de systèmes complexes tels que les entreprises virtuelles.

C'est dans ce contexte que se situe cette thèse. Le travail réalisé est divisé en deux parties.

Dans la première partie, nous avons proposé une approche totalement automatisée basée sur la méta-modélisation et les grammaires de graphes pour générer des modèles réseaux de Petri pour les processus métiers en utilisant ATOM<sup>3</sup>. Ensuite, nous avons généré un environnement graphique pour l'outil d'analyse des réseaux de Petri INA (qui est textuel). Ceci nous permet de passer automatiquement des processus métiers vers la spécification (textuelle) de INA éliminant ainsi les risques d'erreurs qui peuvent être causées par une transformation manuelle. L'outil INA est alors utilisé pour l'analyse des propriétés du système modélisé. L'approche proposée est illustrée par une étude de cas.

Dans la deuxième partie, nous avons proposé une approche intégrée UML/ECATNets pour la modélisation et la vérification des processus métiers dans les entreprises virtuelles.

L'approche proposée suit le processus de développement UP. UML est utilisé pour la modélisation tandis que le ECATNets sont utilisés pour l'analyse et la vérification des diagrammes obtenus. Plus précisément, nous avons proposé une transformation automatique des *diagrammes de séquence* vers la catégorie des réseaux de Petri appelée *ECATNets* basée sur les grammaires de graphes et réalisée à l'aide de l'outil ATOM<sup>3</sup>. Ensuite, le langage Maude est utilisé pour analyser les modèles obtenus.

Dans un travail future, nous comptons continuer la transformation des autres diagrammes UML (diagramme état/transition, diagramme de cas d'utilisation, diagramme d'activité, etc ...) vers les ECATNets en utilisant toujours la technique de transformation de graphes et l'outil ATOM<sup>3</sup>.



Pour mieux valider nos approches, des cas réels et plus complexes seront pris en considération.

Nous planifions également de réaliser une implémentation de nos approches en utilisant d'autres outils de transformation de graphes tels que AGG et Viatra dans un but de comparaison des performances.

Dans le cadre du reverse engineering, une perspective intéressante de ce travail consisterait en l'obtention d'interprétations automatiques des résultats d'analyse des réseaux de Petri sur les modèles des processus métiers. Par exemple, comment interpréter un interblocage dans un modèle réseaux de Petri dans un modèle de diagramme de séquences ?

D'autres modèles de processus métiers émergent, donc d'autres transformations basées sur la transformation de graphes peuvent être envisagées. Nous pouvons citer par exemple, la transformation des BPMN vers les réseaux de Petri, des 21 workflow patterns vers UML, etc ...

## Bibliographie

[AGG] AGG Home page: <http://tfs.cs.tu-berlin.de/agg/>

[Alter02] Alter, S. (1999), *“Information Systems: The foundation of E-Business”*, Prentice Hall, NJ. 2002.

[Alter99] Alter, S. *“Information Systems: A Management perspective”*, 3ème édition, Addison-Wesley. (1999).

[Andries99] M. Andries, G. Engels, A. Habel, B. Hoffmann, h.-J. Kreowski, s. Kuske, D. Pump, A. Schürr et G. Taentzer, *“Graph transformation for specification and programming”*, Science of Computer programming, vol 34, NO°1, pages 1-54, Avril 1999.

[Atom3] ATOM<sup>3</sup> (2006). Home page: <http://atom3.cs.mcgill.ca/>

[Balci02] O. balci, W. Ormsby, *“Expanding our horizons in verification, validation and accreditation research and practice”*, 2002 Winter Simulation Conference, E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes (éditeurs), 2002

[Baldan05] Paolo Baldan, andrea Corradini and fabio Gadducci, *“Specifying and verifying UML Activity diagrams Via graph Transformation”*, Lecture Notes in Computer Science, Global Computing, pages 18-33, Springer Berlin / Heidelberg 2005

[Balsamo01] S. Balsamo, M. Simeoni, *“On transforming UML models into performance models”* Workshop on Transformations in the Unified Modeling Language, 2001.  
<http://ase.arc.nasa.gov/wtuml01/submissions/balsamo-simeoni.ps>

[Bardohl04] R. Bardohl, H. Ehrig, J. DeLara and G. Taentzer, *“Integrating Meta Modelling with Graph Transformation for Efficient Visual Language Definition and Model Manipulation”*. Lecture Notes in Computer Science 2984, pp. 214-228, 2004.

[Bellon84] B. Bellon et J.M. Chevalier, *“L'industrie en France”*, Flammarion, Paris, 1984.

[Benzaken91] C. Benzaken, *“Systèmes Formels: Introduction à la logique et à la théorie des langages”*, Editions Masson, 1991.

[Berrah97] L. Berrah, *“Une Approche d'évaluation de la Performance Industrielle : Modèle d'indicateur et Techniques Floues pour un Pilotage Réactif”*, Thèse de doctorat de 3ème cycle, I.N.P.G., Septembre 1997

[Bettaz92] M. Bettaz, and M. Maouche. "How to specify Non Determinism and True Concurrency with Algebraic Term Nets". *Lecture Notes in Computer Science, N 655, Spring Verlag, Berlin*, p. 11-30, 1992.

- [**Bettaz93**] M. Bettaz, M. Maouche, M. Soualmi, and M. Boukebeche. "Protocol Specification Using ECATNets". *Networking and Distributed Computing*, pp 7-35. 1993.
- [**Bettaz96**] M. Bettaz, A. Chaoui and K. Barkaoui, "*On Finding Structural Deadlocks in ECATNets Using a Logic of Concurrency*", *Journal of Computing and Information*, Vol 2 No 1, pp. 495-506, 1996.
- [**Blanc05**] X. Blanc, "*MDA en action : Ingénierie logicielle guidée par les modèles*", Eyrolles, 2005.
- [**Bloomfield99**] R.E. Bloomfield and D. Craigen, "*Formal methods diffusion: Past lessons and future prospects*", Technical Report D/167/6101/1, Bundesamt für Sicherheit in der Informationstechnik, Bonn, Germany, December 1999.
- [**Boudiaf04**] N. Boudiaf and A. Chaoui, "*Dynamic Analysis Algorithm for ECATNets*", Proceedings of MOCA'04, pp 47-63, October 11-13, 2004, ISSN 0105-8517, Daniel Moldt (Ed.), Aarhus, University, Denmark.
- [**Boudiaf05**] Noura Boudiaf, Allaoua Chaoui, "*A Rewriting Logic Based Tool for ECATNets Analysis*", Edition and Simulation Steps Description", *European Journal of Scientific Research*, Vol 5 No 1, 2005, ISSN 1450-202X
- [**Boudiaf06**] N. Boudiaf, K. Barkaoui, and A. Chaoui, "*Implémentation des Règles de Réduction des ECATnets dans Maude*", 6<sup>ème</sup> Conférence Francophone de MOdélisation et SIMulation - MOSIM'06, Rabat, Maroc, avril 2006.
- [**Box79**] G.E.P Box, "*Robustness in scientific model building,*" *Robustness in statistics*", In R.L. Launer and G.N. Wilkinson (Editors), Academic Press, New York, 1979, pp. 201–236.
- [**Brandenburg05**] Brandenburg H, Wojtyna J.P, "*L'approche processus, mode d'emploi*", Paris: Editions d-Organisation. (2005).
- [**Brilman95**] J. Brilman, "*L'entreprise reinventée*", Editions d'organisation, 1995.
- [**Browne95**] J. Browne, P. Sackett et J. Wortmann, "*Future manufacturing systems-toward the extended enterprise*", *Computers in Industry*, Volume 25, pp. 235-255.
- [**Burlton01**] R.T Burlton, "*Business process management: Profiting from process*", Indianapolis: Sams, 2001.
- [**Butera91**] F. Butera, "*La métaphore de l'organisation : du château au réseau*", Les Edition d'Organisation, Paris, 1991.
- [**Butler99**] K.A Butler, C. Esposito et R. Hebron, "*Connecting the design of software to the design of work*", *Communications of the ACM*, 42 , 1, 39-46, 1999.

- [**Catton00**] M. Catton, "*Management des processus, une approche innovante*", Paris : Afnor, 2000.
- [**Chaoui09**] A. Chaoui, R. Elmansouri, W. Saadi, and E. Kerkouche, "From UML Sequence Diagrams to ECATNets: a Graph Transformation based Approach for modelling and analysis", to appear in the proceedings of ICIT'2009, Al Zaytoonah University, Amman, 3-5 June, 2009, Jordan.
- [**Chapulrat05**] V. Chapulrat, B.Kamsu-Foguen,F. Pruner, "Aformal verification framework and associated tools for enterprise modelling: Application to UEML", computers in industry, Novembre 2005.
- [**Chapulrat07**] V. Chapulrat, "Vérification et validation de modèles de systèmes complexes : application à la modélisation d'Entreprise"Habilitation à diriger des recherches, présentée à l'université Montpellier II, a Ecole doctorale ISS. Soutenue le 1<sup>er</sup> Mars 2007.
- [**Chatel04**] V.Chatel, C.Feliot, "*Principe de conception système certifiée par la preuve*", Journées Francophones des Langages Applicatifs, JFLA 2004.
- [**Clark01**] Tony Clark, Andy Evans, and Stuart Kent, "The metamodelling language calculus Foundation semantics for UML", In Heinrich Hussmann, editor, *Fundamental Approaches to Software Engineering. 4th International Conference, FASE 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6. 2001 Proceedings*, volume 2029 of LNCS, pages 17–31. Springer, 2001.
- [**Clavel99**] M. Clavel, and al., "*Maude: Specification and Programming in Rewriting Logic*". Internal report, SRI International, 1999.
- [**Cloutier99**] L. Cloutier, "*Une approche multi-agents par conventions et contrats pour la coordination de l'entreprise manufacturière réseau*", Thèse de doctorat, Université d'Aix-Marseille, 1999.
- [**Curtis92**] B. Curtis, M.I Kellner et J. Over, "Process modeling", *Communications of the ACM*, 35(9), 75-90, 1992.
- [**Czarnecki03**] K. Czarnecki and S. Helsen, "Classification of Model Transformation Approaches", OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.
- [**Czarnecki06**] K. Czarnecki, S. Helsen, "Feature-based survey of model transformation approaches", *IBM SYSTEMS JOURNAL*, VOL 45, NO 3, 2006.
- [**Daft04**] R. L Daft, "*Organization theory and design*", Mason, Ohio: Thompson, 2004.
- [**Davenport93**] T.H Davenport, "*Process innovation: Reengineering work through information technology*", Boston: Harvard Business School Press, 1993.

**[DeLara02]** J. De Lara and H. Vangheluwe, "*AToM<sup>3</sup>: A Tool for Multi-Formalism Modelling and Meta-Modelling*", Lecture Notes in Computer Science, No 2306, pp. 174-188, 2002.

**[DeLara04]** De Lara, J and H. Vangheluwe. "*Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM<sup>3</sup>*", Manuel Alfonseca. *Software and Systems Modelling*, Springer-Verlag, Vol 3(3), pp. 194-209. Special Section on Graph Transformations and Visual Modeling Techniques, 2004.

**[Delatour99]** J. Delatour and M. Paludetto, "*UML/PNO, a way to merge uml and petri net objects for the analysis of real-time systems*", International Workshop Conference : UML 99, Mulhouse, France, 3-5 Juin 1999.

**[Dome99]** DOME (1999). Home page: <http://www.htc.honeywell.com/dome/>

**[Domokos]** Péter Domokos And István Majzik, "*From Uml Class Diagrams To Timed Petri Nets*".

**[Doumeingts84]** G. Doumeingts, "*La méthode GRAI*", Thèse d'état, Université de Bordeaux I, France, 1984.

**[Dufresne03]** Thomas Dufresne and James Martin, "*Process Modeling for E-Business*", INFS 770 – Methods for Information Systems Engineering: Knowledge Management and E-Businessm.Spring 2003.

**[Elbert97]** J. Ebert, R. Sttenbach and I. Uhe, "*Meta-CASE in Practice: a Case for KOGGE*". Proceedings of the 9th International Conference, CAiSE'97, Barcelona. LNCS 1250, 203-216, Berlin, 1997.

**[Elmansouri07a]** R. El Mansouri, "*TOWARDS A FULL AUTOMATIC APPROACH TO USE INA PETRI NETS ENVIRONMENT FOR BUSINESS PROCESS MODELING*", PROCEEDINGS OF ACIT'2007, LATTAKIA, SYRIA, 2007.

**[Elmansouri07b]** R. El Mansouri, "*A Full AUTOMATIC APPROACH BASED ON META-MODELLING AND GRAPH GRAMMARS TO GENERATE PETRI NETS MODELS FOR BUSINESS PROCESSES*", Actes du colloque MOAD'07: Méthodes et Outils d'Aides à la Décision, Bejaia, 18-20 Novembre 2007.

**[Elmansouri08a]** R. El Mansouri.R, "*Towards a Modeling Method for Business Processes in Virtual Enterprises based on UML and ECATNets*", Information Management in Modern Organizations: Trends & Challenges, Proceedings of The 9<sup>th</sup> International Business Information Management Association Conference, January 4 - 6, 2008, Marrakech, Morocco.

**[Elmansouri08b]** R. El Mansouri.R, "*On the use of Meta-Modelling and Graph Grammars to Generate Petri Nets models for Business Processes*", IRECOS Journal, January, 2008 issue.

**[Elmansouri08c]** R. El Mansouri, “*Automatic Generation of Petri Nets models for Business Processes using Meta-Modeling and Graph Grammars*”, International Conference on Modelling and simulation AMSE08 Port Said (Egypt), 8- 10 April 2008.

**[Elmansouri08d]** R. El Mansouri, E. Kerkouche, and A. Chaoui, "A Graphical Environment for Petri Nets INA Tool Based on Meta-Modelling and Graph Grammars" , PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY VOLUME 34 OCTOBER 2008 ISSN 2070-3740, Venice, Italy.

**[Engels00]** Gregor Engels, Jan Hendrik Hausmann, Reiko Heckel, Stefan Sauer, “*Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML*”, in: Proceed. of the 3rd International Conference on the UML 2000, YORK, ROYAUME-UNI, Octobre 2000.

**[Fernandez92]** J.-C. Fernandez, H. Garavel, L. Mounier, C. Rodriguez A. Rasse, and J. Sifakis. “*A toolbox for the verification of program*”, In International Conference on Software Engineering, ICSE’14, Melbourne, Australia, pages 246–259, May 1992.

**[Fernandes07]** João M. Fernandes, Simon Jens, Bæk Jørgensen and Óscar Ribeiro, “*Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net*”. 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, IEEE Computer Society Press, Mai 2007.

**[Fernandez96]** Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. Cadp (cæsar/aldebaran development package): “*A protocol validation and verification toolbox*”, In Rajeev Alur and Thomas A. Henzinger, editors, Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA), volume 1102, pages 437–440, August 1996.

**[Fleisch99]** Wolfgang Fleisch, ”*Applying use cases for the requirements validation of component-based real-time software*”, In Proceedings of the 2nd International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’99), May 1999.

**[Fröhlich00]** Peter Fröhlich and Johannes Link, “*Automated test case generation from dynamic models*”, In E. Bertino, editor, Proceedings of ECOOP 2000, volume 1850 of LNCS, pages 472–491. Springer, 2000.

**[Fujaba]** FUJABA Home page: <http://www.fujaba.de/>

**[Garavel98]** Hubert Garavel. Open/Caesar, “*An open software architecture for verification, simulation and testing*”, In Tools and Algorithms for the Construction and Analysis of Systems (TACAS’98), volume 1384. Springer-Verlag, Lecture Notes in Computer Science, 1998.

**[GaubertMacon06]** Christine Gaubert-Macon, "Approche des processus organisationnels et modélisation", <http://www.reseaucerta.org> © CERTA - juillet 2006 – v1.0.

- [**Genrich81**] Genrich H. J. and K. Lautenbach K. “*System Modelling with High-Level Petri Nets*”, Theoretical Computer Science, vol. 13, 1981.
- [**Guerra03**] Esther Guerra and Juan de Lara, “*A framework for the verification of UML models. Example using Petri Nets*”, proceedings of JISBD’03, Alicante, Spain 2003.
- [**Grace02**] GRACE (2002). Homepage:  
<http://www.informatik.uni-bremen.de/theorie/GRACEland/>
- [**Great**] GReAT <http://www.escherinstitute.org/Plone/tools/suites/mic/great>
- [**Goranson97**] T. Goranson, M. Johnson, A. Presley et H.G Rogers, “*Metrics for the Agile Virtual Enterprise Case Study*”, Proceedings of the 6th Annual National Agility Conference, 1997.
- [**Gou00**] Gou Hongmei, Huang Biqing and Ren Shoujum, “*A UML and Petri Nets Integrated Modeling Method for Business Processes in Virtual Enterprises*”, 2000, American Association for Artificial Intelligence ([www.aaai.org](http://www.aaai.org)).
- [**Gou03**] Gou Hongmei, Huang Biqing, Wwnhuang Liu, “*A framework for virtual enterprise operation management*”, Computers in Industry, February 2003.
- [**Haberl**] S. Haberl, “*Business Process Description Languages*”.  
<http://www.cis.unisa.edu.au/~cissh/research/webflow/bpdl.html>>
- [**Hall90**] A. Hall, “*Seven myths of formal methods*”, IEEE Softw., 7(5):11–19, 1990.
- [**Hammer93**] M.H Hammer and J. Champy, “*Reengineering the corporation: A manifesto for business evolution*”, New York.: Harper Business, 1993.
- [**Harmon03**] Harmon, P. “*Business process change: A manager's guide to improving, redesigning, and automating processes*”. San Francisco: Morgan Kaufmann Publishers. (2003).
- [**Hinchey95**] M. G. Hinchey and J. Bowen, editors. “*Applications of Formal Methods*”, Prentice Hall International, 1995. isbn 0-13-366949-1.
- [**Holscher06**] Karsten Holscher, Paul Ziemann, and Martin Gogolla: “*On translating UML models into graph transformation systems*”, Journal of Visual Languages and Computing, pages 78-105, ELSEVIER, 2006
- [**Holzman90**] G.J Holzman, “*Design and Validation of Computer Protocols*”, Prentice Hall, 1990.
- [**Holzman91**] G.J Holzman, “*Design and Validation of Computer Protocols*”, Prentice Hall, 1991.
- [**Holzmann97**] G.J Holzmann, “*The model checker SPIN*”, IEEE Transactions on Software Engineering, 23(5):279–295, May 1997.

- [**ISO85**] ISO. *LOTOS, A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, ISO/ DP 8807, March 1985.
- [**INA**] INA home page : [www2.informatik.hu-berlin.de/~starke/ina.html](http://www2.informatik.hu-berlin.de/~starke/ina.html)
- [**Iso00**] Norme européenne NF EN ISO 9001 version 2000, *Systèmes de management de la qualité – Exigences*, AFNOR, 2000.
- [**Jackson01**] D. Jackson, “*Lightweight Formal Methods*”, Proceedings of International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16., 2001
- [**Jagdev98**] Jagdev.HS Brown.J. "The extended entreprise- a context for manufacturing", Production Planning & Control, Vol.9 N°.3, pp-216-219, 1998.
- [**Jensen97**] K. Jensen, "A Brief Introduction to Coloured Petri Nets", Lecture Notes in Computer Science, No 1217, Springer-Verlag, 1997, pp 203-208.
- [**Jensen98**] K. Jensen, "An Introduction to the Practical Use of Coloured Petri Nets", Lecture Notes in Computer Science, No 1492, Springer-Verlag, 1998, pp 237-292.
- [**Karsai04**] G. Karsai, A. Agrawal, “*Graph Transformations in OMG’s Model-Driven Architecture*”, Lecture Notes in Computer Science, Vol 3062, 243-259, Springer Berlin / Heidelberg, juillet 2004.
- [**Kelly96**] S. Kelly, K. Lyytinen and M. Rossi (1996). "MetaEdit+: A fully configurable Multi-User and Multi-Tool CASE and CAME Environment". In *Advanced Information System Engineering, Springer-Verlag, Lecture Notes in Computer Science*, No 1080, Berlin, 1996.
- [**Kerkouche08**] H. Kerkouche and A. Chaoui, "On the use of Meta-Modelling and Graph Grammars to process and simulate ECATNets model", proceeding of MS'2008, Port Said, April 8-10, 2008, EGYPT.
- [**Khandriche97**] Y. Khandriche, "Aide à la conduite dans un contexte perturbé: Propositions et application à un cas industriel, Etat de l'art", Juil 97.
- [**Khriss00**] I. Khriss, M. Elkoutbi, R.K Keller, "User Interface Prototyping based on UML Scenarios and High-level Petri Nets", Département d’informatique et de recherche opérationnelle, Université de Montréal, Septembre 2000.
- [**King99**] Peter King and Rob Pooley, “Using UML to derive stochastic Petri net models”, UKPEW '99, University of Bristol, Juillet 1999, UK Performance Engineering Workshop.
- [**King00**] Peter King and Rob Pooley, “Derivation of Petri Net Performance Models from UML Specifications of Communications Software”, in Haverkort, Bohnenkamp and Smith Eds, Computer Performance Evaluation - Tools 2000, Proceedings of the 11th International



Conference on Tools and Techniques for Computer Performance Evaluation, Illinois, Mars 2000, pages 262-276.

**[kuske02]** Sabin kuske, Martin Gogolla, Ralf Kollman and kreoski, “*An integrated semantics for UML Class, Object and state diagrams based on graph transformation*“, dans: M. Butler, K. Sere (Eds.), Third International Conference on Integrated Formal Methods (IFM’02), Lecture Notes in Computer Science, vol. 2335, pages 11-28, Springer, Berlin, 2002.

**[Kwon00]** Gihwon Kwon, “*Rewrite rules and operational semantics for model checking UML statecharts*”, In Andy Evans, Stuart Kent, and Bran Selic, editors, UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings, volume 1939 of LNCS, pages 528–540. Springer, 2000.

**[Lamsweerde02]** A. Van Lamsweerde, “*Formal Specification: a Roadmap, The Future of Software Engineering*”, A. Finkelstein (ed.), ACM Press, 2002

**[Latella99]** Diego Latella, Istvan Majzik, and Mieke Massink, “*Automatic verification of a behavioural subset of uml statechart diagrams using the spin model-checker*”, Formal Aspects of Computing, 11:637–664, 1999.

**[LeGuennec01]** A. LE GUENNEC, “*Génie Logiciel et Méthodes Formelles avec UML Spécification, Validation et Génération de tests*”, Thèse de doctorat soutenue le 29 juin 2001 à L’UNIVERSITÉ DE RENNES 1.

**[LeMoigne90]** J.L Le Moigne, “*La modélisation des systèmes complexes*”, Afcet-systèmes, Dunod, 1990.

**[Levi02]** M.H Levi, “*The business process (quiet) revolution: Transformation to process organization*”, Interfacing Technologies Corporation. Retrieved from <http://www.interfacing.com/rtecontent/document/CreatingProcessOrganization03.pdf>

**[Lilius99]** Johan Lilius and Ivan Porres Paltor, “*Formalising UML state machines for model checking*”, In Robert France and Bernhard Rumpe, editors, UML’99 – The Unified Modeling Language. Beyond the Standard. Second International Conference ,Fort Collins, CO, USA, October 28-30. 1999, Proceedings, volume 1723 of LNCS, pages 430–445. Springer, 1999.

**[LopezGrao04]** Juan Pablo LopezGrao, Jose Merseguer, and Javier Campos, “*From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering*”. In Proceedings of WOSP’2004, pages 25-36, 2004.

**[Lorenzoni88]** G. Lorenzoni et O.A Ornati, “*Constellations of Firms and New Ventures*”, Journal of Business Venturing, Volume 3, pp 41-57, 1988.

**[Lutherer96]** Lutherer E. “*Méthodes et outils de modélisation pour la productique*”, Thèse de 3ème cycle, INSA Lyon, 1996.

[**Marca88**] D.A MARCA and C.L Mc GOWAN, "*Structured analysis and design technique*", USA: Mc Graw-Hill, 1988, 216p.

[**Meseguer96**] J. Meseguer, "*Rewriting Logic as a Semantic Framework of Concurrency: a Progress Report*", Springer-Verlag, Lecture Notes in Computer Science, No 119, pp. 331-372, 1996.

[**McCormack01**] K.P McCormack and W.C Johnson, "*Business process orientation – Gaining the e-business competitive advantage*", Florida: St. Lucie Press, 2001.

[**McMillan92**] K.L McMillan, "*The SMV system, symbolic model checking - an approach*", Technical Report CMU-CS-92-131, Carnegie Mellon University, 1992.

[**Megarsti97**] Riad Megarsti, "*Etude comparative des méthodes d'analyse des systèmes de production*", Mémoire D.E.A. de PRODUCTIQUE et d'INFORMATIQUE, Soutenu le 23 Septembre 1997 à l'Université de Droit, d'Economie et des Sciences d'Aix-Marseille III

[**Meseguer96**] J. Meseguer, 1996, "*Rewriting Logic as a Semantic Framework of Concurrency: a Progress Report*", Seventh International Conference on Concurrency Theory (CONCUR'96), Volume 1119 of LNCS, Springer Verlag, p. 331-372, 1996.

[**Meseguer00**] J. Meseguer, "*Rewriting logic and Maude: a Wide- Spectrum Semantic Framework for Object-based Distributed Systems*", In S. Smith and C.L. Talcott, editors, Formal Methods for Open Object-based Distributed Systems, (FMOODS'2000), p. 89-117. Kluwer.

[**Meseguer 03**] José Meseguer and Javier Campos, Software "*Performance Modeling using UML and Petri nets*". In Proceedings of MASCOTS Tutorials'2003. pages.265-289

[**Montmain06**] J.Montmain, J-M. Penalva, V.Chapurlat, A-L.Courbis, B.Vayssade, M.Vinches, "*Le management du risque*", Rapport interne, Audit Risque du 21 Juin 2006.

[**Morley02**] C. Morley, "*La modélisation des processus : typologie et proposition utilisant UML*", Processus et Systèmes d'information – Journées ADELI, Paris, France, 2002.

[**Murata89**] T. Murata. "*Petri nets: Properties, Analysis and Applications*", Proceedings of the IEEE, Vol. 77, No 4, April 1989.

[**NASA 1998**] "*NASA Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems*", Volume II: A Practitioner's Companion (voir la version de 1998 à l'adresse [http://eis.jpl.nasa.gov/quality/Formal\\_Methods/document/NASA\\_gb2.pdf](http://eis.jpl.nasa.gov/quality/Formal_Methods/document/NASA_gb2.pdf) )

[**Offutt99**] Jeff Offutt and Aynur Abdurazik, "*Generating tests from UML specifications*", In Robert France and Bernhard Rumpe, editors, UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings, volume 1723 of LNCS, pages 416–429. Springer, 1999.

[**OMG03**] Updated joint initial submission against the action semantics for UML RFP, available at <http://cgi.omg.org/cgi-bin/doc?ad/00-08-03>.

[**OMG03a**] “*Object Management Group: OMG Unified Modeling Language Specification*”, Version 1.5, mars 2003.

[**OMG03b**] “*Object Management Group (OMG): MDA Guide Version 1.0.1*”, 2003.

[**OMG04**] “*Object Management Group (OMG), Model Driven Architecture (MDA)*”, site Internet, <http://www.omg.org/mda>, 2004.

[**Ouardani06**] Adel Ouardani, Philippe Esteban, Mario Paludetto, Jean-Claude Pascal: “*A Meta-modeling Approach for Sequence Diagrams to Petri Nets Transformation within the requirements validation process*”. In: Proceedings of the European Simulation and Modeling, pages 345-349, Conference (ESM'2006).

[**Paltor99**] Ivan Paltor and Johan Lilius, “*vUML: A tool for verifying UML models*”, In Robert J. Hall and Ernst Tyugu, editors, Proc. of the 14th IEEE International Conference on Automated Software Engineering, ASE'99. IEEE, 1999.

[**Paulo00**] Paulo J. F. Carreira and Miguel E. F. Costa, “*Automatically verifying an object-oriented specification of the steam-boiler system*”, In Stefania Gnesi, Ina Schieferdecker, and Axel Rennoch, editors, Proceedings of the 5th International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS'2000), pages 345–360. GMD, 2000.

[**Pep**] PEP home page: <http://peptool.sourceforge.net/>

[**Petit99**] M. Petit, “*Formal requirements engineering of manufacturing systems: a multiformalism and component-based approach*”, Thèse de doctorat de l'Université de Namur, Belgique, Octobre 1999.

[**Petri62**] C.A. Petri. “*Communication with automata*”. Supplement 1 to technical report RADCTR-65-377, Vol 1, New-York 1966, Translated from “*Kommunikation mit Automaten*” PhD Bonn 1962.

[**Poulin84**] D. Poulin, B. Montreuil et S. Gauvin, “*Bâtir aujourd'hui l'organisation de demain*”. Montréal (Québec), 1994.

[**Progres02**] PROGRES (2002). Home page:

<http://www-i3.informatik.rwth-aachen.de/research/projects/progres/>

[**Prakash6**] N. Prakash, S. Srivastava and S. Sabharwal. “*The Classification Framework for Model Transformation*”, Journal of Computer Science 2 (2): 166-170, 2006.

[**Python**] Python home page: <http://www.python.org>

[**Quanlong99**] Li Quanlong, Ye Dan, Zhan Dechen, Xu Xiaofei, “*Agile Virtual enterprise process modelling*”, Computer Integrated Manufacturing System--- CIMS Vol. 5, No. 4, 1999.

- [**Roboam93**] M. Roboam, "*La méthode GRAI, principes, outils, démarche et pratique*", Teknéa, Toulouse, 1993.
- [**Roch02**] S. Roch and P.H Starke P.H, "*Integrated Net Analyzer*", User manual, 2002.
- [**Rolstadas98**] A. Rolstadas, "*Virtual and extended enterprise-definition*" *Production Planning & Control*", Vol.9 N°.3,pp-215, 1998.
- [**Roser05**] S. Roser, B. Bauerm. "*A Categorization of Collaborative Business Process Modeling Techniques*", Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops (CECW'05) 0-7695-2384-6/05 \$20.00 © 2005 IEEE.
- [**Rozenberg99**] G. Rozenberg, "*Handbook of Graph Grammars and Computing by Graph Transformation*", Vol.1. World Scientific, 1999.
- [**Saadi09**] W. Saadi, "*Une approche de transformation des diagrammes UML vers les ECATNets*", Mémoire de Magister, soutenu à l'Université de Biskra, Janvier 2009.
- [**Saldhana01**] J.A. Saldhana and S.M. Shatz, "*UML Diagrams to Object Petri Net Models : An Approach for Modeling and Analysis*", PDCS'2001. 14 th International Conference on Parallel Systems, UML and Petri nets relations and distributed computing systems, Las Vegas, Nevada, April 2001.
- [**Scheer99**] A.W Scheer, "*ARIS - Business Process Modeling*", Springer, 1999.
- [**Sivaraman02**] E. Sivaraman and M.Kamath, "*On The Use of Petri Nets for Business Process Modeling*", Proceeding of the 11th Annual Industrial Engineering Research Conference, Orlando, FL., May 2002.
- [**Sztipanovits**] Sztipanovits, J., et al. "*MULTIGRAPH: An architecture for model-integrated computing*". *In proceedings of ICECCS'95*, pp. 361-368, Ft. Lauderdale, Florida, 1995.
- [**Talbot03**] Jean Talbot, "*Les T.I. et la réingénierie des processus*", HEC MONTreal- MBA, 2003.
- [**Tina**] Tina Home page : <http://www.laas.fr/tina/>
- [**Tham96**] K.D THAM, "*PERA, Enterprise Modelling*", *Enterprise integration laboratory of Toronto*", <http://www.ie.utoronto.ca/EIL/entmethod/PERA/introper.html>
- [**Telelogic03**] T. Telelogic, "*UML Suite and SDL Suite documentation*", 2003 <http://www.taug2.com/>
- [**Thérroude02**] F. Thérroude, "*Formalisme et système pour la représentation et la mise en œuvre des processus de pilotage des relations entre donneurs d'ordre et fournisseurs*", thèse de Doctorat de l'Institut National Polytechnique de Grenoble, 2002.

[**Touzi07**] J. TOUZI, "*Aide à la conception de Système d'Information Collaboratif support de l'interopérabilité des entreprises*", thèse doctorat spécialité : SYSTEMES INDUSTRIELS, Préparée au Centre de Génie Industriel - Ecole des Mines d'Albi Carmaux, soutenue le 09/11/2007

[**Trowitzsch05**] J. Trowitzsch, A. Zimmermann, and G. Hommel, "*Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets*", 13th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS), Denver, Colorado, Avril 2005.

[**Trowitzsch06**] J. Trowitzsch and A. Zimmermann, "*Using UML State Machines and Petri Nets for the Quantitative Investigation of ETCS*", Proceedings of the 1st international conference on Performance evaluation methodologies and tools, Octobre 2006, Pisa, Italie.

[**Valette02**] R. Valette, "*Les Réseaux de Petri*", LAAS-CNRS Toulouse, septembre 2002.

[**VanLamsweerde02**] A. Van Lamsweerde, "*Formal Specification: a Roadmap, The Future of Software Engineering*", A. Finkelstein (ed.), ACM Press, 2002.

[**Vernadat96**] F. Vernadat, "*Enterprise modelling and integration, Principles and applications*", Chapman & Hall, 1996.

[**Vernadat99**] F. Vernadat, "*Techniques de modélisation en entreprise : Applications aux processus Opérationnels*", economica 1999.

[**Vesterager99**] J. Vesterager, J.D Pedersen et M. Tille, "*Virtual Enterprise Reference Architecture and Methodology*", report IMS 95001/ESPRIT 26509 Globeman21 (Global Manufacturing in the 21st Century), 120 pages.

[**VIATRA**] VIATRA Home page:

<http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html>

[**Voster04**] Voster project (2001-2004) : VO Modelling Report (2003), Deliverable D24.1. European project IST-2001-32031, <http://cic.vtt.~/projects/voster/>.

[**Willams92**] T.J Willams, "*The Purdue Enterprise Reference Architecture*", Purdue Laboratory for Applied Industrial Control, West Lafayette, Indiana 47907, USA, 1992.

[**Willams94**] T.J Willams, "*Development of GERAM, a generic enterprise reference architecture and enterprise integration methodology*", A project of the IFAC/IFIP task force on Architectures for enterprise integration, 1994.

[**Yang**] Yang Ruiyao, Huang Biqing, Liu Wenhua, Gou Hongmei and Li Yu, "*UML in Business Process Modelling of Virtual Enterprises*".

[**Zaidat05**] A. Zaidat, "*Spécification d'un cadre d'ingénierie pour les réseaux d'organisations*", Thèse de doctorat en Génie Industriel de l'Ecole des Mines de Saint-Etienne et l'Université Jean Monnet. Septembre 2005.

**[Zhaoxia04]** Zhaoxia Hu and Sol Tjell M. Shatz, “*Mapping UML Diagrams to a Petri Net Notation for System Simulation*” Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), Banff, Canada, Juin 2004, pages. 213-219.