الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

Université 20 Août 1955-Skikda
Faculté des Sciences
Département d'Informatique
D042114002M

جامعة 20 أوت 1955 سكيكدة
كلية العلوم
قسم الإعلام الآلي

Mémoire présenté en vue de l'obtention
Du diplôme de

Magister en Informatique

Spécialité : Techniques avancées pour les systèmes parallèles et distribués

Formalismes et environnements pour la modélisation et la simulation des systèmes à évènements discrets

Présenté par :

Sofiane BOUKELKOUL

Soutenu publiquement le 01/10/2014

Devant le jury composé de :

Mr Boucheham Bachir	Maître de conférences (A)	Président	Université 20 Août 1955-Skikda
Mr Benmohammed Mohammed	Professeur	Examinateur	Université Mentouri Constantine 2
Mr Mazouzi Smaine	Maître de conférences (A)	Examinateur	Université 20 Août 1955-Skikda
Mr Redjimi Mohammed	Maître de conférences (A)	Rapporteur	Université 20 Août 1955-Skikda

A Mes parents, Ma femme et Ses parents,

A ma famille et mes amis,

A Dania, Lamis et Abd Errahmane.

A tous ceux que j'aime…

Remerciement

الحمد لله الذي بنعمته تتم الصالحات

Je tiens à remercier profondément Monsieur Redjimi Mohammed, MC'A-HDR à l'université de Skikda, pour m'avoir fait confiance depuis mon ingéniorat, et d'avoir accepté de diriger ce travail de recherche. Ainsi je lui exprime mon profond respect pour ses encouragements et son attitude d'enrichir et de mettre en valeur les modestes idées.

Mes remerciements vont également à Monsieur Boucheham Bachir MC'A-HDR à l'université de Skikda, pour l'intérêt qu'il a porté à ce travail en acceptant de présider le jury de la soutenance.

J'adresse aussi ma respectueuse reconnaissance à Monsieur Mazouzi Smaine MC'A-HDR à l'université de Skikda, d'avoir accepté d'examiner ce travail de recherche.

Je voudrais également exprimer mon respect et ma profonde reconnaissance à Monsieur Benmohammed Mohammed, Professeur à l'université Mentouri Constantine 2, de m'avoir honoré d'examiner ce travail.

Mes gratitudes vont également à mes parents pour leur prière, ma femme pour sa patience et sa confiance, mes beaux parents pour leur encouragement et toutes les personnes qui ont contribué à la réalisation de ce travail.

Résumé

La modélisation et la simulation nous permettent de représenter la structure et les comportements des systèmes. Plusieurs méthodes existent pour la modélisation et la simulation des systèmes. Nous pouvons considérer les approches mathématiques, mécaniques et informatiques. Les approches formelles telles que les réseaux de Pétri ou les machines à états finis présentent un ensemble d'outils permettant la représentation des systèmes et leur évolution dans le temps. Nous nous intéressons particulièrement au formalisme DEVS « Discret EVent System Spécification » qui est un formalisme de modélisation et de simulation initié par B. P. Zeigler dans les années 1970. DEVS est introduit comme formalisme abstrait pour la modélisation à événements discrets et est devenu un formalisme universel, grâce à sa force de joindre plusieurs modèles hétérogènes dans le but de modéliser les systèmes complexes. Le travail que nous présentons consiste, dans un premier temps, à faire un état de l'art très poussé de différentes approches, formalismes, méthodes et environnements touchant à la modélisation. Nous nous intéresserons ensuite plus particulièrement à la modélisation des systèmes à événements discrets. Aussi, les réseaux de Pétri représentent des outils formels très puissants et assez utilisés pour la représentation des systèmes et leurs évolutions dans le temps. Nous nous intéressons ainsi à la mise en œuvre de mécanismes de transformation de ces outils vers le formalisme DEVS. L'outil final de développement d'application évoluera dans un environnement de multi-modélisation que nous définirons en fin de ce travail de recherche et qui sera validé par une mise en œuvre robuste utilisant les composants JAVA et XML.

Mots clés :

Modélisation et simulation (M&S), Discrete EVent System specification (DEVS), Réseaux de Pétri (RdP), transformations entre modèles, systèmes complexes.

Abstract

Modeling and simulation allow us to represent the structure and behavior of systems. Several methods exist for systems' modeling and simulation. We can consider the mathematical, mechanical and computer approaches. Formal approaches such as Petri nets or finite state machines represent a set of tools for the representation of models and their evolution over time. We are particularly interested in DEVS "Discrete Event System Specification", which is a formalism for modeling and simulation initiated by BP Zeigler in the 1970s. DEVS is introduced as an abstract formalism for discrete event modeling and became universal formalism thanks to its strength to join multiple heterogeneous models in order to model complex systems. The work we present is, at first, to highly develop a state of the art of different approaches, formalisms, methods and environments related to modeling. We then focus more specifically on the modeling of discrete event systems. Also, Petri nets represent a formal and very powerful tool which is widely used for the representation of systems and their evolution over time. Therefore, we are interested in the implementation of

transformation mechanisms of these tools to the DEVS formalism. The final tool application development evolves in an environment of multi-modeling which we will define by end of this research, which will be validated by a robust implementation using JAVA and XML components.

Key words:

Modeling and simulation (M&S), Discrete EVent System specification (DEVS), Petri Networks (PN), model transformation, complex systems.

الملخص

النمذجة والمحاكاة تسمح لنا بتمثيل بنية و سلوك الأنظمة. توجد عدة طرق لنمذجة و محاكاة الأنظمة. إذ يمكننا اعتبار الطرق الرياضية، الميكانيكية والمتصلة بالإعلام الآلي. تعتبر النهج الرسمية مثل شبكات Petri أو آلات الحالات المحدودة مجموعة من الأدوات التي يمكنها تمثيل الأنظمة و تطورها عبر الزمن. نحن مهتمون خصوصا بالشكلية DEVS « Discret EVent System Spécification » "وصف نظام الأحداث المنفصلة" ، للنمذجة والمحاكاة التي بدأها ب. ب. زيغلر سنوات 1970. بدأ DEVS كشكلية مجردة للنمذجة الخاصة بالأحداث المنفصلة ثم أصبح شكلية عالمية بفضل قوته في وصل عدة نماذج غير متجانسة من أجل نمذجة النظم المعقدة. العمل الذي نقوم بتقديمه هو، في البداية، مدخل مفصل إلى مختلف النهج ، الشكليات ، الأساليب و البيئات المتعلقة بالنمذجة. ثم نركز بشكل خاص على نمذجة أنظمة الأحداث المنفصلة. أيضا، تعد شبكات Petri أدوات رسمية قوية جدا وتستخدم إلى حد ما في تمثيل النظم و التغييرات الخاصة بها عبر الزمن. و في هذا الصدد نحن مهتمون بتنفيذ آليات التحول من هذه الأدوات إلى شكلية DEVS. التطبيق النهائي يتطور في بيئة متعددة النماذج و الذي سنقوم بتبيينه في نهاية هذا العمل الخاص بالبحث ،حيث يتم التصديق عليه بتنفيذ صارم، معتمدين في ذلك على مكونات JAVA و XML .

كلمات الدلالة:

النمذجة والمحاكاة (M & S) ، Discrete EVent System Specification (DEVS) ، شبكات Petri ،

Table des matières

Titres **Pages**

Chapitre 2

Chapitre 3

chapitre 4

chapitre 5

Liste des figures

Liste des tables

Liste des algorithmes

Introduction générale

1. Introduction

Depuis son apparition sur terre, l'homme n'a jamais cessé de chercher la connaissance et le savoir. Il a toujours eu envie de savoir et de comprendre les choses qui l'entouraient, en posant des questions et cherchant des réponses. C'est dans cet objectif que les différentes disciplines scientifiques ont connu leur progrès. Il y a lieu de citer deux grands axes avec lesquels, les scientifiques cherchent à percer les secrets des systèmes : l'expérience directe et les solutions analytiques. Ainsi, les mathématiques, en se basant sur des théorèmes des calculs et différentes démonstrations et suppositions, ont contribué énormément dans la résolution des problèmes analytiquement. Cependant, pour les systèmes complexes, ces solutions mathématiques sont, très souvent, trop lourdes, trop couteuses en temps et elles sont parfois d'une très grande difficulté à mettre en œuvre. Ainsi, pour certains types de problèmes, on ne sait pas encore comment les formuler mathématiquement et on ne sait pas en donner des résolutions analytiques. Pour d'autres, même si l'on arrive à en trouver de telles solutions, le calcul serait anormalement prohibitif et la probabilité de tomber dans des erreurs serait considérable. D'autre part, appliquer des expériences sur des systèmes réels n'est pas toujours évident surtout quand il s'agit de systèmes complexes (tels que les écosystèmes), dangereux (tels que les usines nucléaires), hors d'atteinte de l'être humain (tels que les robots investigateurs des planètes) ou des systèmes dont les coûts de telles expériences serait extrêmement excessif (tels que les systèmes de productions). Ce contexte s'applique aussi dans les cas de la conception de futurs systèmes (tels que la conception des usines ou des infrastructures urbaines).

C'est dans ce cadre que le besoin d'adopter de nouvelles approches, autres que les solutions analytiques ou les expériences réelles, est devenu primordial. La modélisation et la simulation sont des approches qui donnent des résultats satisfaisants pour la résolution de problèmes jugés complexes. L'idée est de construire un modèle simplifié qui représente le système réel et de lui faire subir des expérimentations pour en tirer des informations à propos

d'objectifs et de questions posées préalablement, que ce soit des questions sur son fonctionnement ou ses comportements futurs.

En fait, la modélisation et la simulation « M&S » n'est pas une nouvelle discipline. Elle intervient à chaque fois que la théorie montre ses limites dans la résolution de certains problèmes. L'histoire rappelle comment la princesse tyrienne Didon a utilisé la simulation pour résoudre son problème. Après la mort de son père, le roi, elle a pris la fuite avec ses fidèles pour préparer le bâti d'un nouveau royaume. En voulant s'installer sur les rives du nord de l'Afrique (Carthage-Tunisie), le roi berbère, propriétaire des terres, lui avait proposé une aumône dérisoire : 'pas plus que peut délimiter la peau d'un bœuf'. La princesse Didon le prit au mot. Elle découpa la peau du bœuf en lanières très fines pour entourer le maximum de surface. En plus elle choisit un terrain au bout de la mer pour ne pas gaspiller de la lanière. Finalement, en faisant une simulation par sa ceinture de lin, elle prouva qu'avec un périmètre fixe, le cercle est la forme géométrique qui englobe plus de surface parmi les autres formes [Hédi et Fauqué, 2001]. Le théorème qui démontre cette vérité n'a été découvert qu'au siècle dernier.

C'est dans ce cadre que la simulation a imposé sa place dans les différentes disciplines ; là où les expériences ou les solutions analytiques montrent leurs impuissances, elle intervient.

Maintenant avec le progrès impressionnant des outils informatiques, la M&S n'a cessé d'évoluer. Elle s'est divisée même en sous-spécialités : méthodologies de conception, techniques de vérification et de validation, implémentation, …etc.

Ce mémoire rentre dans ce contexte général. Par ailleurs, nous allons présenter un état de l'art du cadre de la M&S. Nous nous focalisons sur les systèmes à événements discrets et les formalismes qui permettent leur modélisation. Nous présentons une synthèse globale des différents outils et plateformes de M&S des systèmes à évènements discrets. Ensuite nous détaillons notre contribution qui porte sur l'intégration des formalismes hétérogènes par la transformation des modèles.

2. Problématique

La M&S des systèmes complexes nécessite leur décomposition en sous systèmes qui sont relativement mieux compréhensibles. Ainsi l'assemblage des sous modèles générés, en tenant compte des interactions entre ces composants, va mener au modèle global qui représente le système étudié. D'autre part, la diversité des outils et formalismes de modélisation permettent une souplesse aux modélisateurs. On est devenu plus au moins libre face au choix des formalismes. Ainsi pour chaque catégorie de système on trouve un ensemble de formalismes qui lui conviennent le mieux. Dans le cas des systèmes complexes, on est même obligé, parfois, d'utiliser plusieurs formalismes. Du moment qu'un seul formalisme ne peut pas traiter tous les aspects d'un tel système. C'est dans ce contexte que la multi-modélisation intervient. En fait, nos travaux de recherche portent sur la réponse à la problématique qui consiste à faire coexister plusieurs modèles basés sur différents formalismes, qui peuvent être hétérogènes, en un seul modèle qui englobe le tout. Différentes approches peuvent être adoptées pour résoudre ce genre de problème[Vangheluwe, 2002].

Nous nous intéressons dans nos travaux à la technique de transformation des modèles. Cette approche consiste à choisir un formalisme pivot, vers lequel, les modèles des autres formalismes vont être transformés. Dans ce mémoire nous nous intéressons au formalisme DEVS « Discret Event System Specification » qui a été initié par B. P. Zeigler par les années 70 [Zeigler, 1984], comme étant un formalisme pivot (cible) et le formalisme des réseaux de Petri « RdP » comme étant un formalisme source. En effet, notre travail concerne la mise en œuvre d'outils algorithmiques qui permettent la transformation de modèles formalisés par des RdP en modèles basés sur le formalisme DEVS. Ainsi ces modèles peuvent être simulés dans les environnements DEVS.

3. Organisation du document

Ce document sera organisé en cinq chapitres comme suit :

Le premier chapitre sert d'introduction au domaine de la modélisation et de la simulation, nous y présentons un ensemble de concepts généraux à propos des systèmes, des modèles, de la simulation…etc.Nous y présentons aussiles notions concernant multi-modélisation et la modélisation multi-échelle. Nous abordons, aussi, de façon approfondie la théorie de la modélisation dans son cadre expérimental comme définie par Zeigler.

Dans le deuxième deuxième chapitre, nous nous focalisonssur les systèmes à événements discrets en mettant l'accent sur leur modélisation et leur simulation. Ainsi, un ensemble de formalismes qui sont utilisés dans ce type de modélisation sera défini. Parmi ces formalismes, nous donnons plus d'importance aux réseaux de Petri et DEVS qui vont faire l'objet de notre contribution en matière de transformation entre modèles. Aussi, dans ce même chapitre, nous allons parler de la modélisation modulaire et hiérarchique basée sur DEVS, tout en mettant en relief, les avantages et la puissance du formalisme DEVS en matière de cohabitation et d'unification des différents formalismes.

Par la suite, nous présenterons, dans le troisième chapitre, un panorama de différents outils et plateformes basés sur le formalisme DEVS, et qui sont utilisés dans la modélisation et la simulation des systèmes à événements discrets. Nous terminerons ce chapitre par une synthèse des différents outils présentés auparavant. Par ailleurs, la diversité des formalismes et des outils de modélisation posera une autre problématique qui a trait à la question d'intégration et de cohabitation des modèles basés sur différents formalismes. Nous tenterons d'y apporter une réponse dans le chapitre.

Le quatrième chapitre est dédié à notre contribution de transformation. A ce niveau, nous allons entreprendre la problématique de la multitude de formalismes. Nous aborderons, ainsi, la notion de transformation entre modèles. Par conséquent, nous définirons l'approche IDM (Ingénierie Dirigée par les Modèles). Ensuite, nous citons quelques transformations de modèles vers des modèles basés DEVS. Dans cette optique nous citons à titre d'exemples, les transformations des agents et des réseaux de Petri vers DEVS. Ce chapitre constitue le cœur de notre contribution. A ce niveau nous allons présenter avec détail notre approche de transformation entre les réseaux de Petri (étant le formalisme source) et le formalisme DEVS (formalisme cible). Les règles formelles de transformation vont être, bien définies. Ainsi que

l'algorithme de transformation. Nous allons nous servir dans l'explication de cette approche d'un exemple de réseaux de Pétri « producteur-consommateur », qui va être transformé en modèle couplé DEVS. Nous finirons ce chapitre en donnant quelques perspectives.

Au niveau du cinquième chapitre, nous allons présenter l'implémentation de l'approche de transformation dans une plateforme de modélisation et de simulation de modèles basée DEVS qui est aussi le produit de notre propre développement. Dans ce chapitre, nous allons mettre l'accent sur la portabilité des modèles générés par notre application. Et ce grâce aux enregistrements de ces derniers au format XML. En outre, nous allons détailler, d'avantage, les bibliothèques assurant la manipulation des fichiers XML.

Une conclusion générale ainsi que des perspectives et voies de recherche seront développées à la fin de ce mémoire.

Chapitre 1

La modélisation et la Simulation (M&S)

1.1. Introduction :

La compréhension d'un système réel passe par le filtre de notre pensée qui construit des abstractions du monde sensible. Il y a même des thèses qui disent que l'intelligence elle-même procède par modélisation et/ou simulation, c.-à-d. par construction d'une image stylisée et dynamique du réel, puis création de scenarios sur la base de ce modèle pour se plonger dans l'avenir et prendre des décisions [Dupuy, 1994].

Dans ce chapitre, nous allons illustrer les notions de modélisation et de simulation dans un cadre introductif. Nous définirons, ainsi, les concepts et entités qui ont une relation avec cette notion de telle sorte que ce chapitre aille nous servir d'introduction à la problématique de ce mémoire.

Pour bien situer ces notions de modélisation et de simulation, nous proposons de commencer par des définitions générales.

1.2. Définitions et concepts généraux :

1.2.1. Notion de système

Un système est un ensemble d'éléments en interaction, organisés dans l'objectif d'atteindre un but. Ces éléments évoluent dans un ou plusieurs environnements et chacun a des fonctions bien déterminées à accomplir. Pendant son évolution dans le temps, un système peut subir des influences exercées par le milieu extérieur comme il peut agir sur ce dernier [Boissier et al, 2004].

Selon P.A. Fishwick [Fishwick, 1995] *« Un système est une partie de la réalité ou opèrent le temps et l'espace et des relations causales entre les différentes parties de ce système. Nous posons nécessairement des frontières en fabriquant un monde fermé et en*

identifiant clairement les éléments qui font partie du système et ceux qui l'affectent de l'extérieur ».

Cette définition situe le système dans un cadre bien défini auquel on doit spécifier les frontières et déterminer les éléments qui lui appartiennent et ceux qui sont derrières ses frontières et l'affectent de l'extérieur. Les systèmes peuvent être classés suivant leur dynamique et leur évolution dans le temps en systèmes continus ou discrets. Ils peuvent aussi être classés suivant leur comportement extérieur qui définit la relation entre les entrées et les sorties, en systèmes déterministes ou indéterministes (stochastiques).

Un système est dit continu s'il comprend, parmi les variables de son état, au moins une variable dont la valeur change de manière continue. Contrairement aux systèmes discrets où l'état est considéré stable pendant des intervalles de temps donnés. Dans cette catégorie de systèmes, si l'état change par la perception d'évènements, on parle alors des systèmes à événements discrets.

La deuxième classification ne s'intéresse pas au temps et à sa manière d'écoulement. Elle est basée, en revanche, sur le comportement des systèmes. Si un système se comporte toujours de la même façon après avoir été soumis aux mêmes conditions ; alors il est dit « déterministe». Par opposition, le système indéterministe (ou stochastique) est tout système dont le comportement est imprévisible. Autrement dit, tout système dont l'état comprend au moins une variable aléatoire, est dit indéterministe.

La notion de système est très liée à celle de modèle. Ce dernier présente une 'copie conforme' du système réel ou d'une ou plusieurs parties auxquelles on s'intéresse en négligeant un certain nombre de paramètres (échelle, espace, temps). On peut alors faire subir au modèle un ensemble d'actions, de transformations et d'expérimentations dont les résultats seront projetés sur le système réel pour en tirer des conclusions. Une telle démarche permet de ne pas pénaliser, voire endommager, le système réel et prédire ses futurs comportements. D'autres systèmes sont loin de l'atteinte de l'être-humain, un moyen efficace pour en faire des expériences consiste à les modéliser.

1.2.2. Le Modèle :

Un modèle est une simplification d'un système réel dans l'objectif de répondre à des questions à propos de son comportement ou de prédire ses futurs comportements. Ainsi la modélisation est l'art de concevoir des modèles en mettant en œuvre des outils.

Pour A. Pavé [Pavé, 1994] : *« Un modèle est une représentation symbolique de certains aspects d'un objet ou d'un phénomène du monde réel ».*

Selon J. Ferber [Ferber, 1995] : *« Un modèle, en science, est une image stylisée et abstraite d'une portion de réalité ».*

Pour P.A Fishwick [Fishwick, 1995]: *« Modéliser c'est décrire la réalité sous la forme d'un système dynamique, à l'aide d'un langage de description, à un certain niveau d'abstraction ».*

Nous rajoutons aussi la définition de Minski qui est souvent citée dans les mémoires et thèse [Millischer, 2000] [Servat, 2000] [Hill, 2000]:

"To *an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A"*.

Cette définition explique l'objectif de la modélisation qui est très attaché au besoin de la compréhension de quelques aspects du système à modéliser [Hill, 2000].

1.2.3. La simulation:

La définition de la simulation par ordinateur est plus ou moins difficile tenant compte de la diversité de domaines desquels descendent différentes applications qui sont généralement hétérogènes. C'est pour cette raison que nous proposons deux définitions complémentaires permettant de construire une idée du contexte dans lequel s'inscrit ce mémoire.

D'après Shannon [Shannon, 1976]:

" The process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or a set of criteria) for the operation of the system"

Cette définition est souvent utilisée dans des travaux touchant le domaine de la modélisation et la simulation comme par exemple dans [Ingalls, 2001, Shannon, 1998].

En effet, le contexte de la simulation s'inscrit dans l'étude d'un système réel (qui peut être futur ou virtuel) afin de comprendre son fonctionnement ou éventuellement de prévoir son comportement dans certaines conditions. Cette tâche ne peut se faire qu'à travers un modèle.

La simulation informatique est en effet indissociable du processus expérimental qui se rattache à son objectif. Ce qui nous conduit systématiquement à une deuxième définition : celle de Fishwick [Fishwick, 1997] :

"Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output."

Autrement dit, la simulation est la mise en évolution dans le temps d'un modèle tout en conduisant des expériences dans l'objectif de comprendre le système ou d'évaluer des stratégies diverses à propos de ses fonctions.

La figure .1.1 dénote les cycles de modélisation et de simulation où le modèle se construit à partir du système dans un cadre expérimental pour répondre à des questions prédéfinies ou bien atteindre des objectifs bien déterminés. Après la modélisation on passe à l'élaboration des plans d'expériences suivant les objectifs de la simulation qui en suit. L'étape suivante est l'analyse des résultats de la simulation. Cette étape pourra mettre en jeu, soit le plan d'expérience ou encore le modèle lui-même.

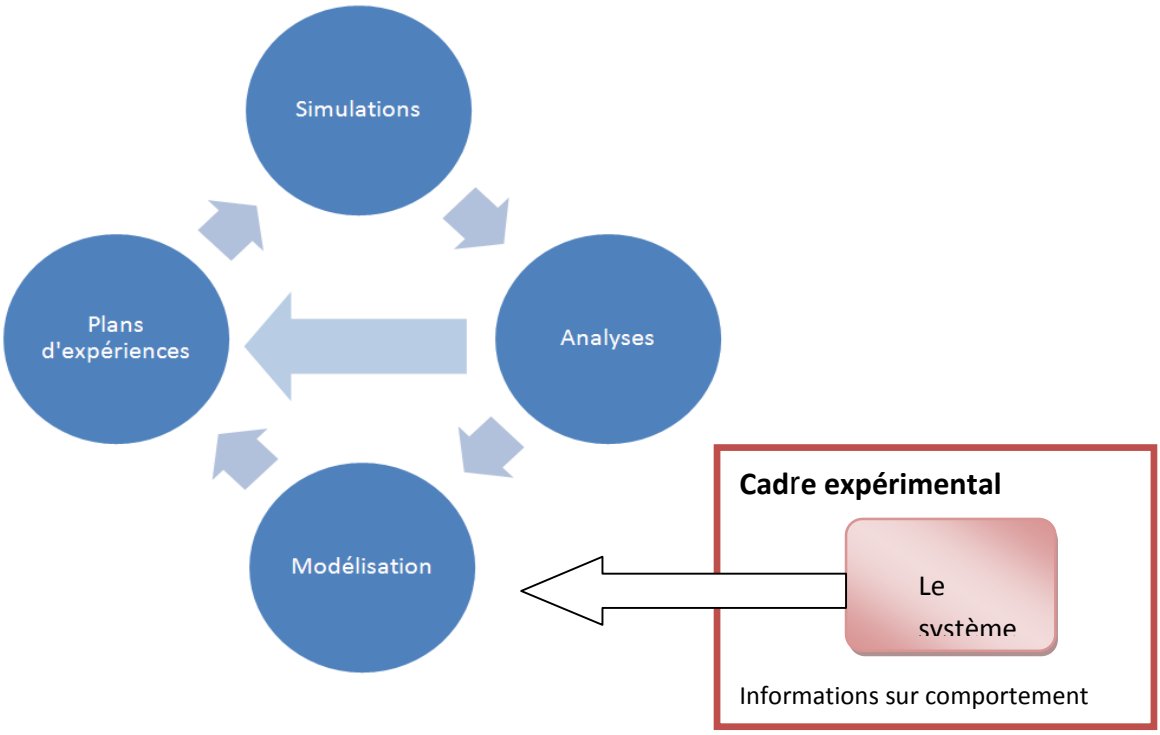

Fig.1.1 Le cycle de la modélisation & simulation

Par la simulation, on peut gagner beaucoup de choses sans pénaliser le système réel par des arrêts dans le cas de l'industrie par exemple. En outre, d'autres systèmes sont hors d'atteinte de l'être humain tels que des systèmes embarqués dans des stations nucléaires…etc. Donc la simulation sur des modèles des systèmes pareils est un outil primordial.

1.3. La simulation en tant que cadre d'expérimentation des systèmes

Pour B.P. Zeigler*« Un système est une source potentielle de données »*, mais il reste à filtrer les données importantes et les observer. Pour cela, il y a lieu d'illustrer les différentes étapes qui constituent l'élaboration d'une simulation informatique. Shannon [Shannon, 1976, Shannon, 1998] propose de distinguer ces étapes de la manière suivante :

1. Définition du problème : dans cette phase initiale, il s'agit de clairement définir les objectifs de l'étude. Quelles sont les questions auxquelles on souhaite apporter une réponse ?

2. Planification du projet : il s'agit d'être sûr que l'on disposera des ressources humaines et matérielles que nécessite l'étude entreprise.

3. Définition du système : dans cette phase, le but est de déterminer quels sont les aspects du système que l'on désire étudier de manière à pouvoir le définir de manière pertinente dans le cadre de l'expérience. Le modèle sera alors élaboré en fonction des objectifs fixés.

4. Formulation du modèle conceptuel : durant cette phase, un premier modèle est élaboré de manière graphique ou en pseudo code. Il s'agit de définir les différentes entités qui composent le système : composants, variables, interaction entre composants, etc.

5. Analyse préliminaire de l'expérimentation : il faut ici déterminer quels sont les critères qui permettront d'évaluer la qualité de l'expérimentation : quels sont les paramètres que l'on souhaite faire varier, avec quelle amplitude et sur combien d'exécutions. Combien d'expériences seront nécessaires à l'expérimentation dans son ensemble.

6. Constitution des paramètres initiaux : durant cette phase, il est question de déterminer et de collecter les données qui sont nécessaires à l'élaboration des valeurs initiales qui seront utilisées pour le paramétrage du modèle.

7. Transcription du modèle : cette étape consiste à convertir le modèle élaboré dans un langage de simulation pour permettre son implémentation sur machine.

8. Vérification et validation : il s'agit ici de vérifier dans un premier temps que le simulateur exécute correctement le modèle (debugging), pour dans un deuxième temps valider les résultats obtenus par celui-ci. Sont-ils acceptables et représentatifs du système que l'on souhaite étudier ?

9. Analyse finale de l'expérimentation : à ce stade de la conception, il convient de reconsidérer l'étape numéro cinq. En effet, il faut en mettre à jour ses conclusions étant donné que la connaissance du modèle s'est considérablement accrue.

10. Expérimentation : la simulation proprement dite est exécutée de manière à récupérer les résultats désirés et à effectuer une analyse de sensibilité du modèle aux paramètres initiaux.

11. Analyse et interprétation des résultats : une fois les simulations effectuées, il s'agit d'inférer des conclusions sur le modèle `à partir des résultats obtenus.

12. Utilisation et documentation: outre les conclusions tirées de l'expérimentation, le modèle et son utilisation doivent être clairement documentés. Ainsi que l'implémentation (pour les futurs systèmes)

La figure 1.2, résume de manière graphique les principales étapes de conception liées à cette approche.

Cependant, cette approche n'est pas toujours la plus évidente pour toute tâche de modélisation et simulation. D'ailleurs dans [Hymore, 1981] on peut trouver une critique intéressante où l'on décrit cette approche comme étant une méthodologie orientée objectifs ; par conséquence, la définition du problème se voit humiliée.

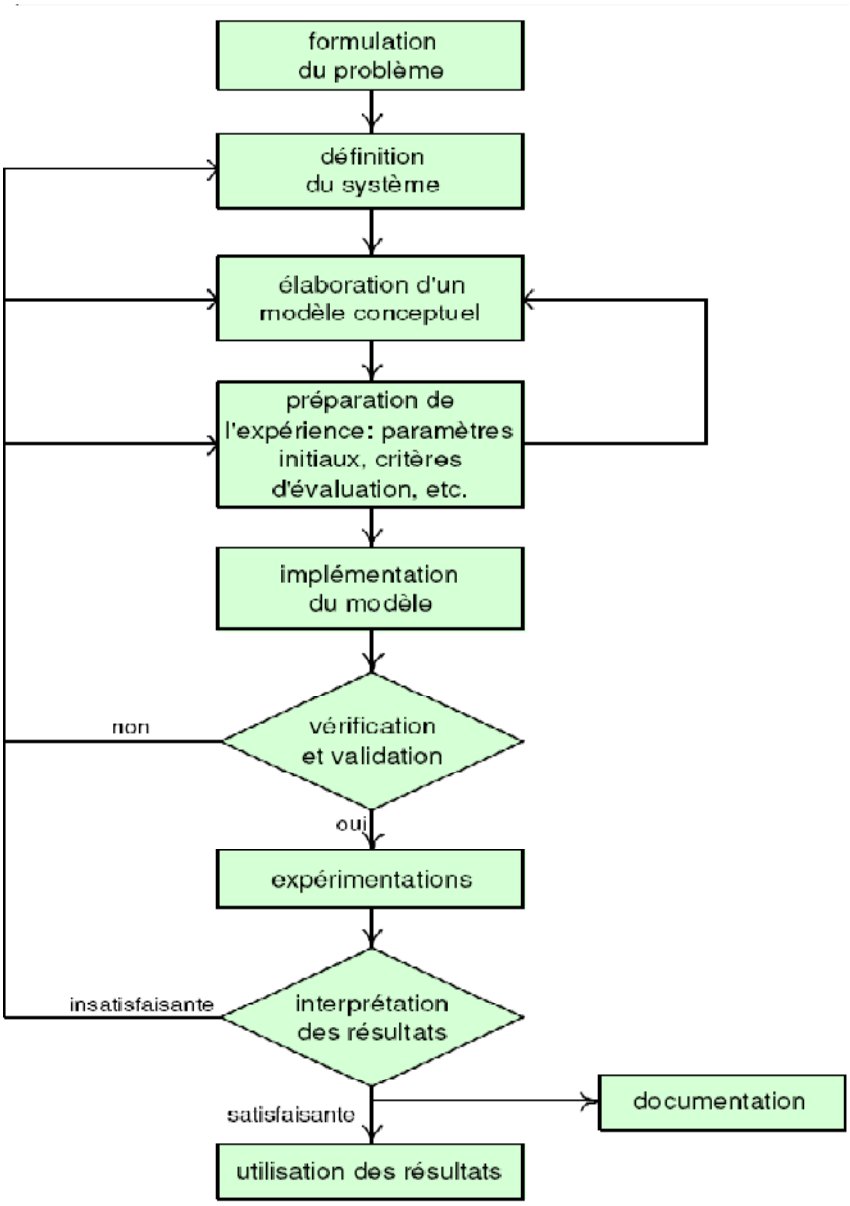

Fig. 1.2 Etapes de conception d'une simulation informatique [Shannon, 1998].

1.4. La modélisation des systèmes

1.4.1. La multi-modélisation

Les systèmes complexes ne peuvent pas être modélisés uniquement par un seul formalisme parce que beaucoup d'aspects sont présents tels que l'hétérogénéité, l'interopérabilité, le souci d'intégration…etc, (figure 1.3). Par ailleurs, un formalisme ne peut

pas traiter tous les cadres de système. Il peut être sémantiquement approprié pour un ou plusieurs aspects d'un système complexe mais pas, généralement, pour tous. Ainsi un système réel peut être modélisé par une variété de modèles issus de différents formalismes [Zeigler,1984 ; Fishwick, 1989].Ainsi, la multi-modélisation a lieu pour bénéficier de la force de représentation des formalismes multiples permettant la coexistence de plusieurs modèles basés sur différents formalismes dans un modèle qui englobe le tout.

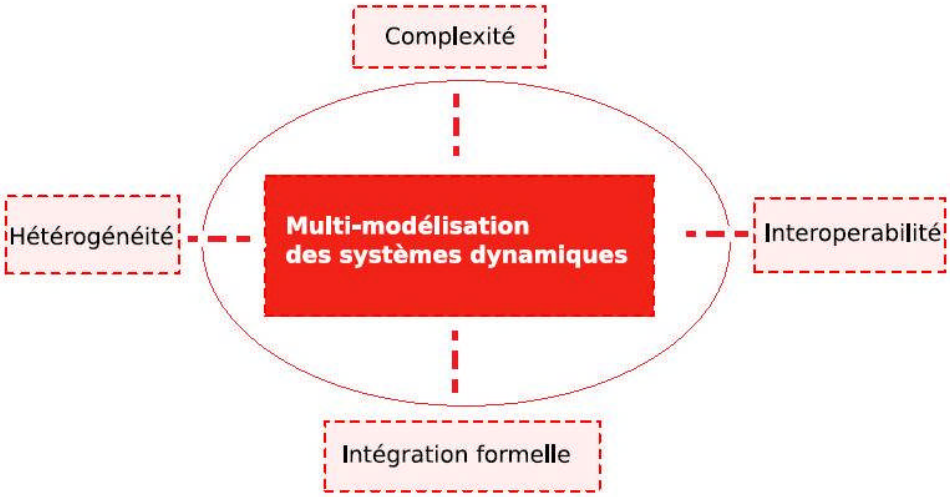

Fig. 1.3Cadre de la multi-modélisation

Selon Hans Vangheluwe [Vangheluwe, 2002], le paradigme de la multi-modélisation se base sur trois axes :

- Le couplage et la transformation des modèles décrits dans différents formalismes.

- L'Abstraction des modèles en définissant le rapport entre eux à chaque niveau d'abstraction.

- Le méta-modèle qui se concentre sur la description des classes des modèles (modèles des modèles).

Dans [Vangheluwe, 2000] nous trouvons une représentation de diverses transformations possibles entre les formalismes en utilisant le graphe de transformation des formalismes " FTG " (figure 1.4).

Le sujet de la multi-modélisation et souvent lié à la notion de paradigme et de formalisme.

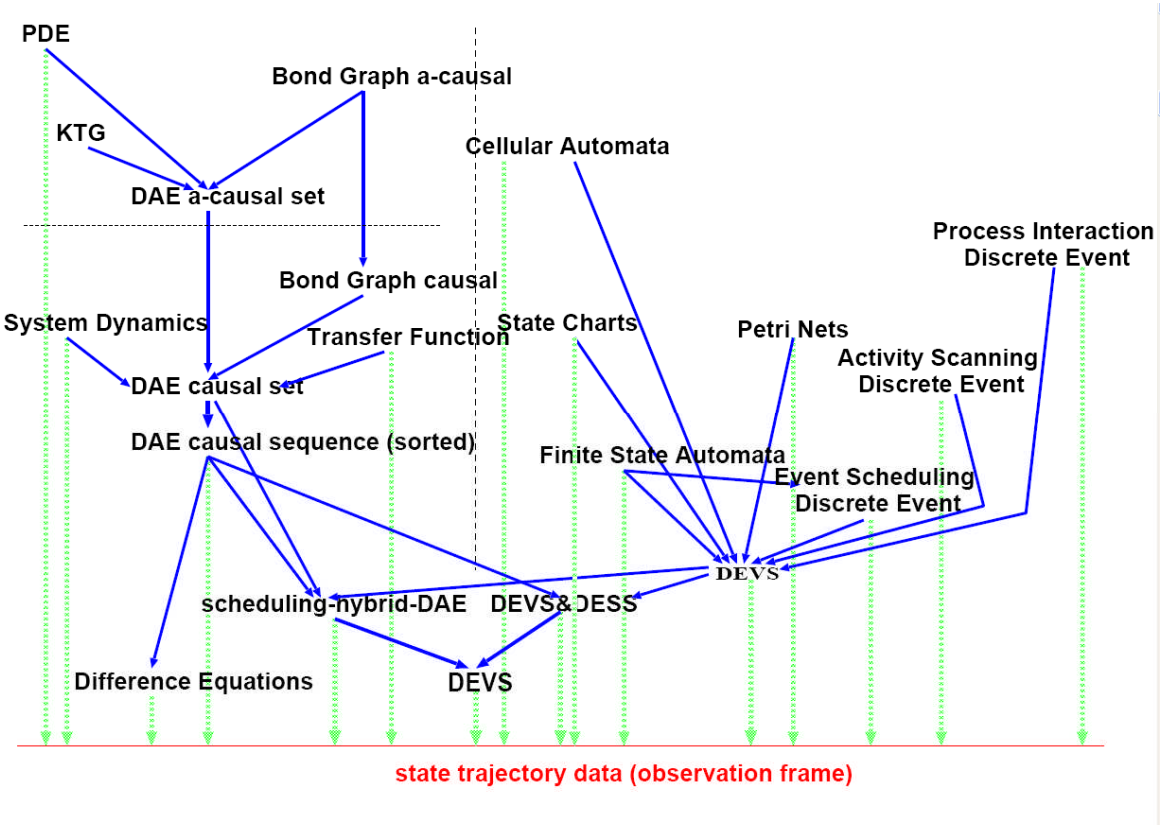

Fig. 1.4 FTG « Graphe de transformation des formalismes » [Vangheluwe ,2000]

1.4.2. La modélisation multi échelles

1.4.2.1. Problématique

La modélisation classique telle que présentée dans les niveaux d'abstraction de Zeigler, s'intéresse aux couplages et à la reconstitution de composants modulaires d'un modèle [Zeigler, 2000]. Cette étape correspond au niveau le plus bas d'abstraction. Cependant, dans les systèmes industriels, quand on cherche à prédire le comportement de tous les phénomènes, couplés entre eux dans la pratique et qui sont présents dans le système, on aura un enchainement de phénomènes divers et d'ordres de grandeur très différents étant donnée l'intégration de toute la connaissance tant théorique qu'empirique, à différentes échelles, dans des composants élémentaires extrêmement détaillées que l'on doit assembler. Ceci entraîne, ainsi, des coûts d'identification et de calcul prohibitifs. La simulation multi-échelles est une réponse à cette problématique. Elle consiste à simuler chaque phénomène à l'échelle la plus pertinente, c'est-à-dire en utilisant plusieurs modèles de tailles et de finesses différentes; cela permet de réaliser des simulations qui seraient inaccessibles par des approches plus directes.

Ce type de modélisation est souvent appelé « modélisation multi-physique » puisque sur le plan physique, elle prend en compte les couplages entre phénomènes élémentaires de nature différente. Par exemple dans le domaine de la physique des réacteurs, on couple la mécanique des structures, la neutronique et la thermo-hydraulique. Dans le domaine de la physique des

matériaux, par exemple, il s'agira de déduire les propriétés macroscopiques d'un matériau poly-cristallin à partir de sa description à l'échelle la plus microscopique (l'atome), via des niveaux de description à différentes échelles. Le défi posé est de lier ces différents niveaux de description en utilisant la bonne information pour passer d'une échelle à l'autre sans discontinuité et de manipuler de façon modulaire ces lois de comportement valables à diverses échelles.

1.4.2.2. Modèle pour chaque échelle

Le modèle multi-échelle se compose de plusieurs modèles et autant d'échelles détectées. La figure 1.5 montre deux modèles pour deux échelles différentes.

- La figure 1.5(a) illustre un modèle « macro» qui représente le produit (matériau) et son environnement extérieur. Au niveau de ce modèle, on ne s'intéresse pas aux phénomènes microscopiques mais aux comportements aperçus à l'échelle macroscopique.

- La figure 1.5(b) illustre le modèle "micro" muni d'une description géométrique et d'un maillage suffisamment fin. Dans ce modèle, on se limite à une ou plusieurs cellules de petites dimensions. Le nombre de cellules à considérer dépend de la nature et de l'étendue des phénomènes microscopiques.

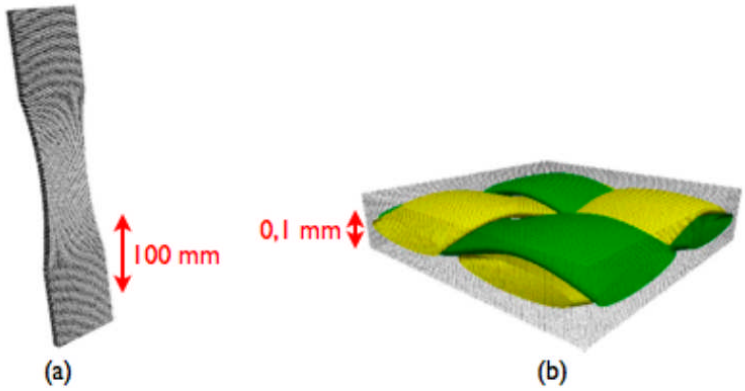

Figure 1.5 : Exemple de modélisation multi-échelles : (a) Un modèle « macro » d'un produit (une éprouvette en composite tissé) ; (b) une cellule « micro » représentant la géométrie du composite.

1.4.2.3. Couplage des échelles

Dans la modélisation multi-échelle le modèle "macro" et les modèles "micro" ne sont pas indépendants. En effet, ils modélisent la même physique à des échelles différentes. Pour que le modèle global soit pertinent, il y a lieu d'exiger des cohérences entre les sous-modèles macro et micro à chaque instant de la simulation. Pour cette raison, les modélisations multi-échelles offre des mécanismes de couplages via des modèles d'interactions, entre les échelles.

Il existe plusieurs façons de définir ces couplages. Nous citons celle qui est la plus utilisée dans le domaine de la mécanique des solides déformables. Elle est basée sur les deux principes suivants, voir Figure 1.6 :

- Le modèle de comportement "macro", qui modélise des phénomènes de déformation macroscopique doit correspondre à la relation contraintes/déformations observée sur la cellule "micro" ;

- Les conditions aux limites "micro", qui traduisent la façon dont la cellule est sollicitée par la matière qui l'entoure doivent, à leur tour, correspondre à l'état de contraintes ou de déformations appliquées sur le matériau à l'échelle macroscopique.

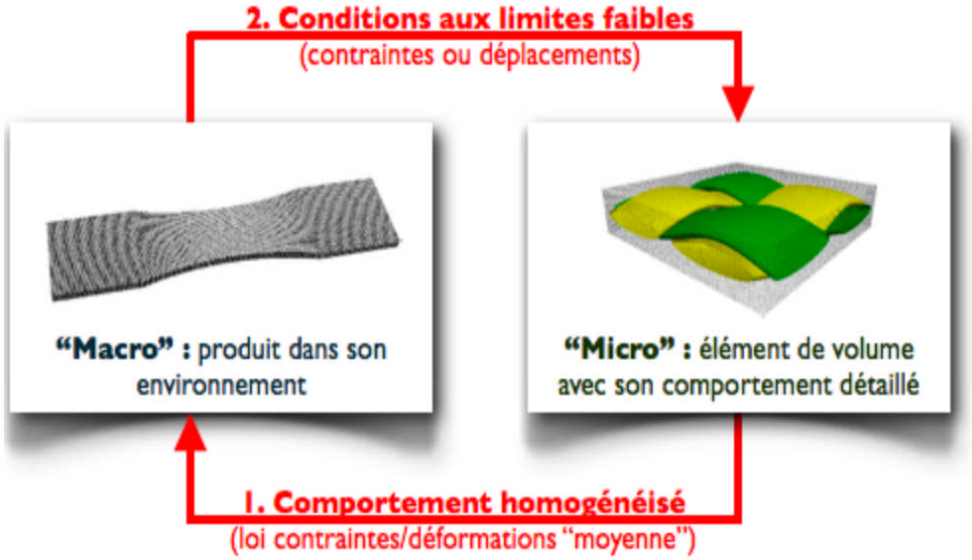

Fig .1.6 Principe général du couplage des modèle multi-échelle « macro » et « micro ».

1.5. Les systèmes dynamiques

1.5.1. La théorie des systèmes

Les bases de la théorie des systèmes ont été élaborées durant les années 60 dans l'objectif de réponde à la problématique liée à la représentation des systèmes dynamiques dans un cadre formel. Par système dynamique on entend:

"Any formal construct which provides general modeling concepts for various kind of disciplines." [Rozenblit & Zeigler, 1993]

La théorie des systèmes propose des concepts, des méthodes et des outils pour la représentation formelle de ces systèmes. Le principe de base de cette théorie est basé sur l'abstraction des systèmes à plusieurs niveaux. Le plus haut niveau considère le système comme une boite noire qui n'a aucun lien avec l'environnement extérieur sauf à travers des

ports d'entrée et des ports de sortie, comme illustré figure 1.7. Théoriquement, le système peut être spécifié suivant deux aspects essentiels [Zeigler et al, 2000]:

– le comportement du système à ses bornes (comportement externe) qui concerne les réactions observables du système depuis l'extérieur.

– la structure interne du système, c'est-à-dire son état interne et son fonctionnement intrinsèque (sa dynamique).

Le comportement externe est la manière avec laquelle le système réagit du point de vue d'un observateur externe. D'autre part c'est la réponse à la question « Quels seront les résultats si le système aperçoit telles entrées ? ». On trouve souvent dans la littérature le terme « événements » pour désigner les entrées [Zeigler, 1976].

Quant à la structure interne du système ; elle est définie selon trois paramètres :

– Le vecteur d'état interne du système qui est composé de l'ensemble des variables pertinentes décrivant formellement l'état du système.

– la dynamique selon laquelle, le système change d'état en fonction des entrées ou de manière autonome. Autrement dit, la détermination de la fonction ou du graphe de transition d'état.

– la manière selon laquelle le système génère des sorties en fonction de son état. D'une façon plus explicite, il s'agit de déterminer les sorties qui peuvent être produites si le système se trouve dans tel ou tel état.

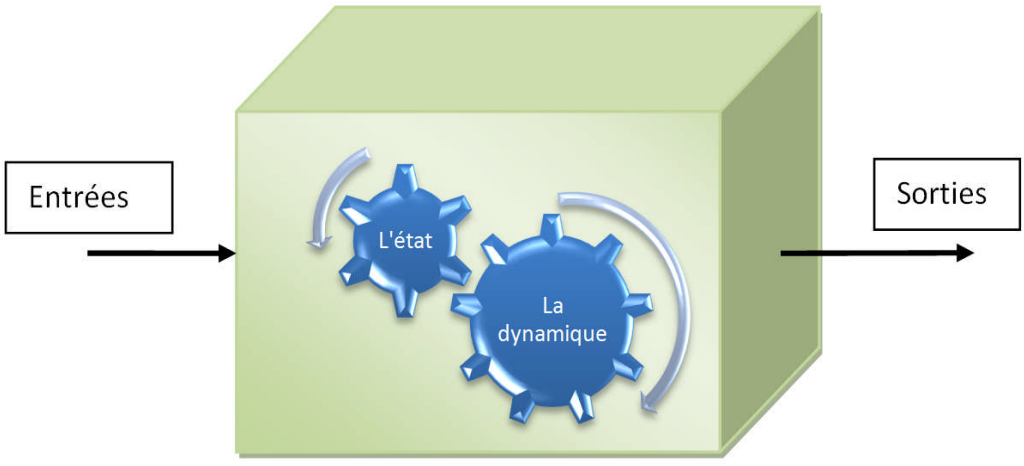

Fig. 1.7 Le système dynamique comme étant une boite noire

Dans ce qui suit, nous présenterons les formalismes qui s'inspirent de cette théorie. Nous allons voir comment ces formalismes mettent l'accent sur la spécification des fonctions qui

implémentent la dynamique des systèmes, ainsi que leur comportement en cours de l'évolution du temps.

1.5.2. Les modèles temporels

Nous avons vu les principaux aspects de la théorie des systèmes. Maintenant nous allons classer les différents modèles temporels qui représentent les systèmes dynamiques. Par ailleurs, la caractéristique la plus importante dans les systèmes dynamiques c'est bien le temps et sa manière de s'écouler. Selon cette caractéristique on peut distinguer trois grandes familles de systèmes et de modèles. Si le temps est considéré comme étant une variable continue dans l'ensemble des nombres réels IR et que l'état du système change continuellement, on parle alors de modèle continu. Si, maintenant, le modèle change d'état uniquement à des instants séparés, par une période égale souvent appelée « le pas ». Pendant cette période, l'état du modèle reste statique: là on parle de modèles discrets. On parle de modèles à évènements discrets si l'état du modèle change à des instants discrets et que l'intervalle du temps n'est pas forcément constant ; le système change d'état lors de l'occurrence d'évènements externes.

1.5.2.1. Les modèles continus

Dans les modèles continus, l'ensemble des variables constituant l'état changent souvent continuellement dans des intervalles de temps infiniment petits. La figure 1.8 montre la variation continue d'une variable d'état au cours du temps.

Par exemple, la quantité « x » d'un liquide dans un bac peut être représentée par une variable d'état qui varie continuellement. Pour calculer l'évolution de x, il est donc nécessaire de connaître son taux de variation au cours du temps, soit $\frac{dx}{dt}$. Ce taux de variation est lui-même une fonction du temps étant le débit D(t). Ce modèle va être représenté par l'équation différentielle suivante: D(t) = $\frac{dx}{dt}$. Ainsi, tous les modèles continus sont représentés par un ensemble d'équations différentielles. Pour modéliser ce genre de modèles on aura souvent recours au formalisme DESS pour « Differential Equation System Specification ». On trouvera dans [Cellier, 1991a] de plus amples détails sur ce type de modèles.

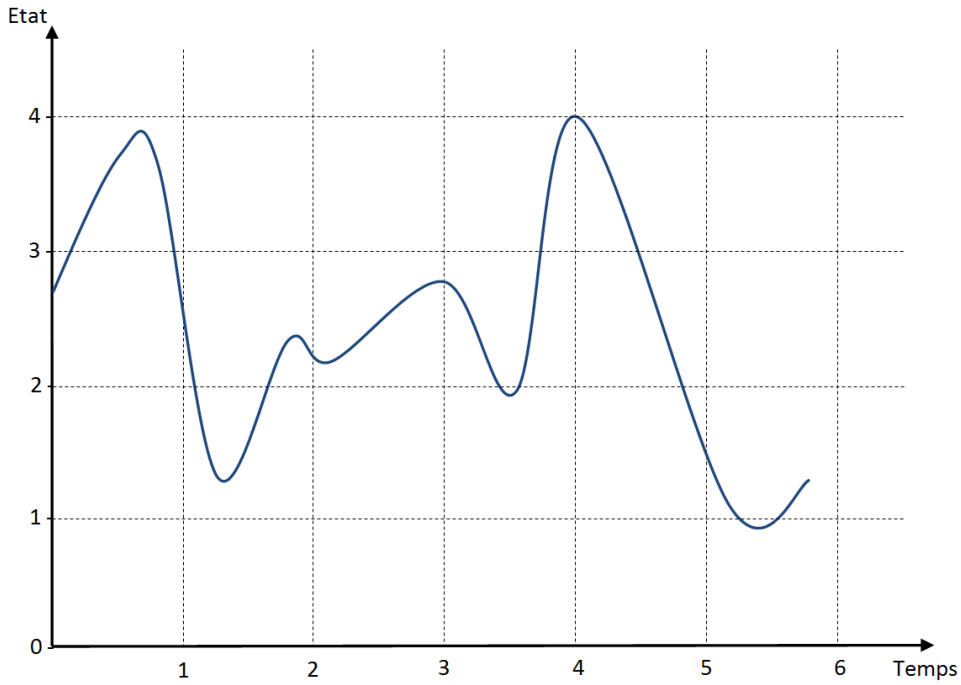

Fig.1.8 Evolution d'une variable dans un modèle continu.

1.5.2.2. Les modèles discrets

Dans ce type de modèles, l'axe du temps est discrétisé suivant une période constante appelée le pas de temps (ou pas d'échantillonnage). L'état du modèle est considéré comme étant stable durant cet intervalle de temps. Au bout de chaque fin de période, le modèle est observé et l'évolution des variables d'état se fait alors de manière discrète et instantanée. La figure 1.9 montre un exemple de l'évolution d'une variable d'état x en fonction du temps dans un modèle discret.

La modélisation de ce type de système est assurée par l'écriture des fonctions qui permettent de calculer l'état du système pour un nouvel instant t+1 à partir de son état à l'instant t. Ainsi, et à tout moment, l'état du modèle pourra être connu en calculant successivement tous les états intermédiaires (n) entre l'instant t de l'état courant et l'instant t+n. Le formalisme DEV pour « Discret EVent » est très approprié pour modéliser ce type de systèmes [Praehofer et al, 1993].

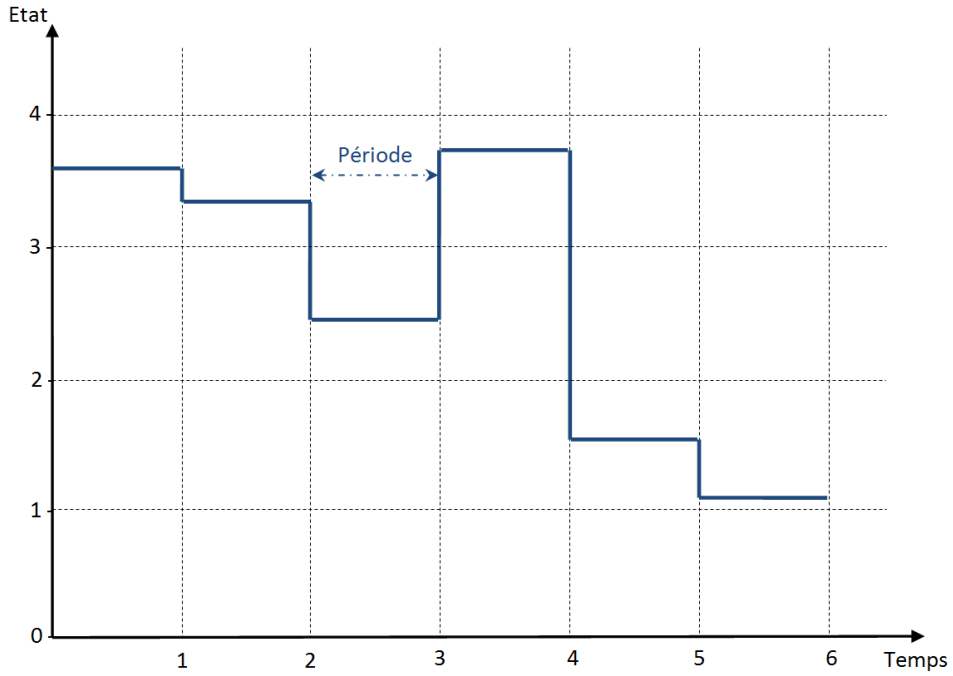

Fig.1.9 Evolution d'une variable dans un modèle discret.

1.5.2.3. Les modèles à événements discrets

Ce type de modèles est une forme de généralisation des modèles discrets, cependant dans ce cas, ce sont les événements occurrents à des instants discrets qui provoquent le changement de l'état du modèle. Etant donné que l'état du modèle est considéré stable entre deux événements successifs. Dans le cas pratique les systèmes à événements discrets changent souvent leur état d'une manière autonome sans réception d'aucun événement de l'extérieur : là on parle d'événement interne. C'est dans les années 1970 que B.P. Zeigler proposa le formalisme DEVS pour « Discret EVent System Specification » qui est l'un des formalismes fondamentaux pour la modélisation des systèmes à évènements discrets [Zeigler, 1976, 1984] [Zeigler et al, 2000] [Ramat, 2003]. Ce formalisme sera détaillé d'avantage dans les chapitres qui suivent. La figure 1.10 montre un exemple de l'évolution de l'état d'un modèle suite à l'occurrence d'évènements à des instants discrets.

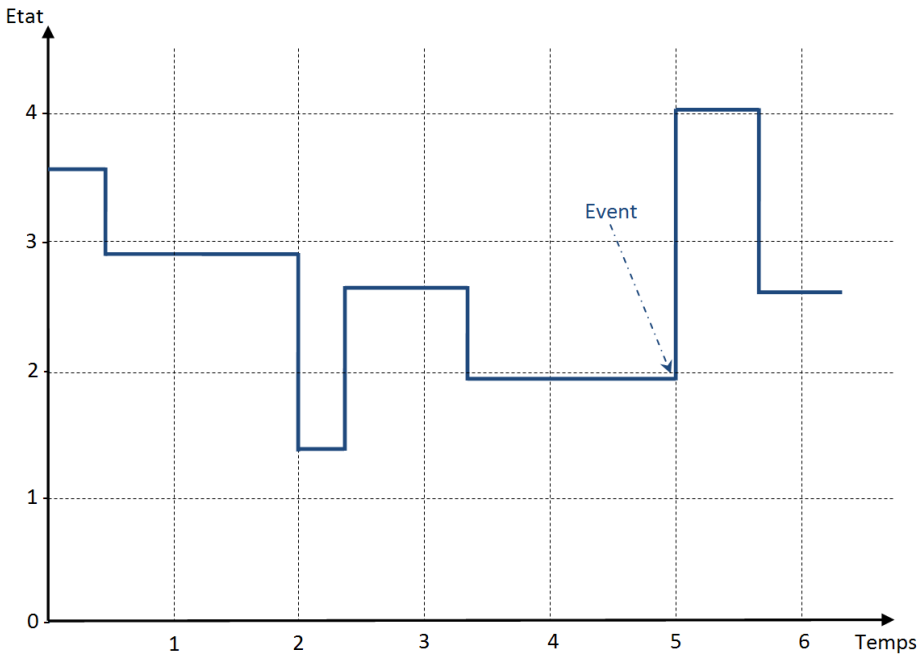

Fig.1.10 Evolution d'une variable dans un modèle à événements discrets.

1.6. Les techniques de simulation

La mise en évolution d'un modèle dans le temps sert souvent de base pour la définition de la notion de simulation. A cet effet, le temps et la manière de sa prise en compte sont des caractéristiques pertinentes à considérer lors du choix ou développement du simulateur. D'autre part, une grande partie des simulateurs est purement liée à la gestion temps. Ainsi, à chaque type de modèle temporel correspondent une ou plusieurs techniques génériques de simulation. Dans cette section nous allons présenter les principes généraux d'implémentation pour simuler les différents modèles temporels présentés précédemment.

1.6.1. Simulation des modèles continus

La présentation des techniques de simulation des modèles continus de façon détaillée est une tâche assez fastidieuse. En effet, il y a lieu de savoir que la simulation des modèles continus pose une grande problématique due à la nature digitale de l'ordinateur, ce qui rend impossible de respecter la nature de la continuité de la dynamique d'un système continu, étant donné que l'évolution du temps se fait avec des pas infiniment petits alors que la simulation informatique repose sur des calculs séquentiels et ponctuels. Cependant les méthodes de calcul numérique permettent de résoudre ce problème de continuité. Ce sont des méthodes mathématiques qui prennent en compte plusieurs paramètres (valeurs passées et/ou futures estimées, taux de changement, etc.) pour estimer la véritable valeur de l'état du système à des instants infiniment petits.

1.6.2. Simulation des modèles discrets

Les techniques d'implémentation utilisées pour la simulation des modèles discrets sont plus au moins faciles à mettre en œuvre du fait que l'évolution du temps est représentée d'une manière discontinue. A chaque pas on incrémente la variable temps après avoir appliqué les fonctions qui modélisent la dynamique du système. La figure 1.11montre le fonctionnement global d'un algorithme pour ce type de simulation.

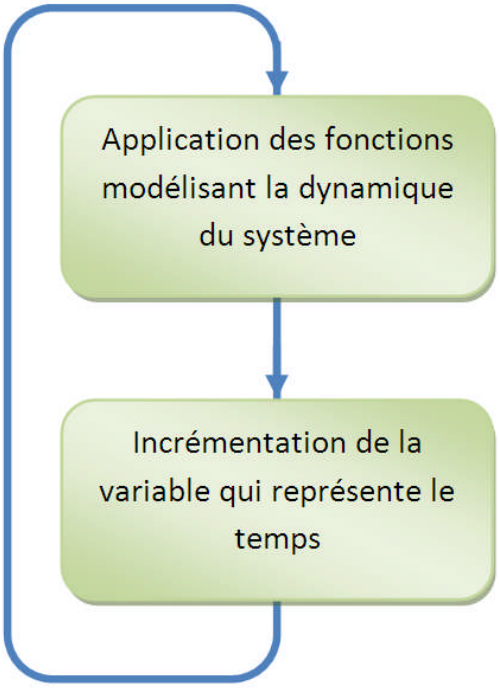

Fig.1.11 Simulation des modèles discrets.

1.6.3. Simulation des modèles à événements discrets

Il existe trois approches d'implémentation pour la mise en œuvre de la simulation de modèles à évènements discrets : par ordonnancement d'évènements (event sceduling), par activités (activity scanning) et par interaction de processus (interaction process) qui combine l'ordonnancement d'évènements et d'activités.

1.6.3.1. L'ordonnancement d'événements

La simulation par ordonnancement d'événements est souvent la technique la plus utilisée. Son principe est de déterminer au fur et à mesure de la simulation les futurs évènements qui vont éventuellement intervenir sur le système. Les événements externes sont généralement générés par ce qu'on appelle des générateurs d'événements, contrairement aux événements internes qui sont produits par les composants système eux-mêmes. Chaque événement ainsi créé est placé dans une liste d'événements, souvent ordonnée chronologiquement. Le temps dans ce type de simulation ne s'incrémente pas forcément par pas de valeurs égales. En fait,

lors de la simulation, l'horloge logique sera avancée jusqu'à la date de l'événement le plus proche. Ce dernier sera retiré de la liste d'évènements en attente et exécuté. Cette exécution pourra éventuellement engendrer la planification d'un ou plusieurs nouveaux événements. Ainsi les événements générés sont intégrés systématiquement dans la liste d'événements. On aura souvent besoin de réordonner les événements par ordre chronologique. La figure 1.12 résume l'algorithme qui correspond à ce principe de simulation. La simulation des files d'attente est un exemple simple de ce type de programmation. Par exemple, pour simuler un système "guichet de poste" on peut définir les événements suivants : E= {"arrivée d'un client dans la queue", "début de service", "fin de service", "départ d'un client"}. L'événement "arrivée d'un client" est un événement externe au système qui peut être généré par un générateur d'événement suivant une loi de distribution tandis que les autres sont des événements internes qui sont générés au fur et à mesure de l'évolution de la simulation. Lorsqu'un client débute son service, la date des événements "fin de service" et "départ d'un client" seront immédiatement calculées. Ainsi les événements générés seront alors planifiés et seront exécutés à la date prévue.

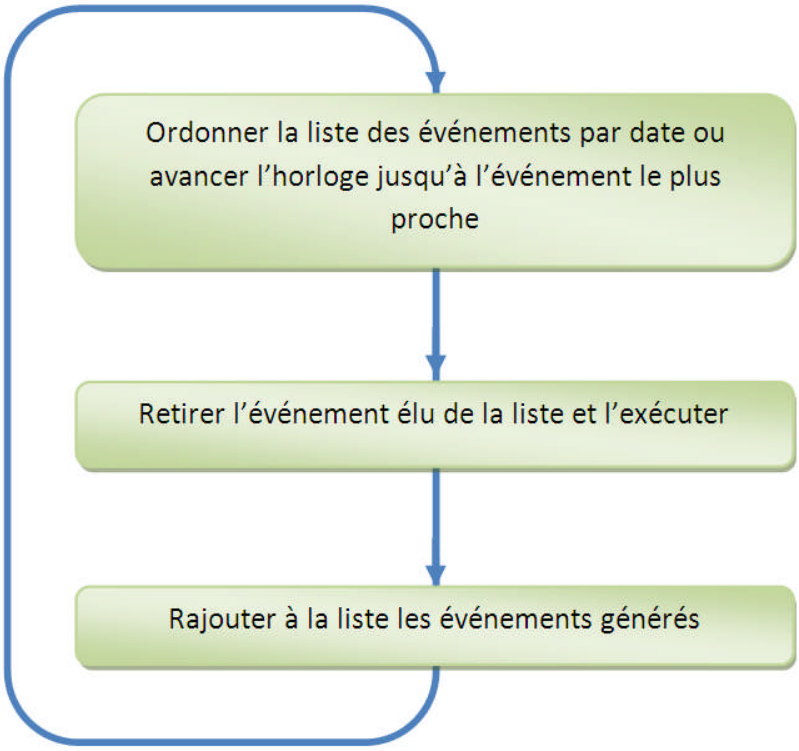

Fig.1.12 Principe de simulation par ordonnancement d'évènements.

1.6.3.2. Analyse d'activités

La simulation par activités se caractérise par le déclenchement conditionné de certains événements, qui ne sont pas planifiés au préalable, au fur et à mesure de l'évolution du temps. Cela est assuré par l'application des règles d'activation à l'occurrence de certains évènements. Si les conditions de telles règles sont vérifiées, alors d'autres événements seront exécutés

malgré que leurs dates d'exécution ne soient pas connues à l'avance. Par exemple, l'événement 'collision de deux avions' interviendra si deux appareils se trouvent en même temps à la proximité d'un point défini dans l'espace tridimensionnel. Pour implémenter cette technique, il suffit, pour chaque exécution d'un nouvel évènement, de vérifier s'il engendre un état du système qui vérifie les conditions de déclenchement de ce type d'événements.

1.6.3.3. Interaction de processus

Dans ce cas, les deux méthodes précédentes sont utilisées pour définir le comportement d'une entité qui se compose –par définition– d'un ensemble de routines capables de gérer son état d'une façon séquentielle. En fait, c'est de cette définition que les composants portent souvent la nomination de processus. D'autre part, au niveau du processus, on gère des entrées, des procédures et des fonctions conditionnelles et la génération des sorties soient sous formes d'évènements, de messages ou d'autres formes. Dans le cas d'un système composé, on distingue, alors, un ensemble de processus en interaction. Ces derniers sont soumis à une horloge globale commune afin de contrôler le rythme de leur évolution. Les événements générés par certains processus peuvent être capturés par d'autres, tout en jouant le rôle d'un stimulus. D'autres événements de type conditionnel seront créés au fur et à mesure de l'évolution de la simulation. Dans [Balci, 1988], on trouve une synthèse plus détaillée des différentes implémentations liées à l'utilisation d'un principe évènementiel. Cette synthèse intègre, aussi, une description du principe de simulation à pas de temps comme étant un cas particulier du principe de simulation par évènements. Cela est basé sur l'analyse des différentes méthodes qui peuvent être utilisées pour définir l'évolution temporelle du système (Time Flow Mechanism TFM).

1.7. La théorie de la modélisation et la simulation

Le domaine de la modélisation et de la simulation (M&S) est devenu aussi vaste que la diversité des disciplines et les centres d'intérêt de la société. Chaque discipline a développé ses propres modèles ainsi que les outils et techniques pour les étudier et analyser. Par conséquent, le besoin de maturation d'une discipline de M&S est devenu de plus en plus essentiel. Actuellement, tous les domaines profitent de la M&S. Cependant, chaque domaine propose des outils et méthodes adaptés. Certains concepts de spécification, simplification, validation et simulation des modèles, peuvent être, cependant, considères comme généraux et ne sont pas spécifiques un domaine particulier. Le professeur B.P. Zeigler a fait une contribution intéressante dans ce domaine dans les années 70 [Zeigler, 1972] actualisée dans [Zeigler et al, 2000]. La M&S reste encore fragmentée entre plusieurs domaines ce qui rend son développement et sa progression difficile, tout en perdant l'intérêt de la réutilisation des idées et techniques des autres disciplines et la collaboration dans un cadre multidisciplinaire. A cet effet, la théorie de la M&S doit répondre à diverses questions telles que le concept d'abstraction et l'intégration de modèles pluridisciplinaires ainsi que le défi d'assumer plusieurs formalismes pour profiter de la force de description de chacun. On doit donner plus d'importance à des concepts fondamentaux touchant l'interopérabilité et la crédibilité (par

exemple : la vérification, la validation, la réutilisation des modèles et sous-modèles). Plusieurs travaux concernant ces approches donnent des résultats très poussés tels que les HLA « High Level Architecture ».

La théorie de la M&S offre des concepts et des formalismes mathématiques pour la représentation des systèmes dynamiques. Ainsi d'après Zeigler [Zeigler et al, 2000] Il y a deux aspects principaux de la théorie :

- Niveaux de spécification des systèmes : Ce sont les niveaux qui servent à la description des comportements des systèmes, ainsi des mécanismes qui permettent leur fonctionnement.

- Formalismes de spécification des systèmes : Ce sont les types et styles de modélisation (continus ou discrets). Ce volet sera détaillé dans le chapitre 2.

1.7.1. Les variables quantitatives et qualitatives d'un modèle

Pour modéliser un système, nous auront besoin, non seulement, de définir la manière d'écoulement du temps mais aussi de définir l'ensemble des variables qui constituent l'état du système sous étudié. On peut ainsi distinguer deux grandes catégories de variables : les variables quantitatives et les variables qualitatives [Cellier, 1991b].

1.7.1.1. Les variables quantitatives

Les variables quantitatives sont représentées par des quantités ou des grandeurs physiques telles que le poids, la taille et la distance. Elles s'expriment en valeurs, généralement par des réels. Dans ce type de variables, on peut distinguer deux catégories : Les variables quantitatives discrètes et continues. La première catégorie concerne les valeurs que l'on peut énumérer. Par exemple, le nombre de personnes dans une salle. Les variables quantitatives continues, quant à elles, sont exprimées par des réels puisqu'elles peuvent prendre des valeurs très nombreuses dont l'énumération serait impossible. Par exemple, la distance entre deux points est une variable quantitative continue puisqu'elle peut être divisée en un nombre infini de distances plus petites.

1.7.1.2. Les variables qualitatives

Les variables qualitatives sont des variables représentées par des qualités, telles que le sexe ou l'état civil. En effet, une variable qualitative est une variable dont le domaine de valeurs est défini dans un domaine D par des valeurs qualitatives comme D={'Jaune','Vert','Rouge','Bleu','Gris'}. On peut distinguer plusieurs types de variables qualitatives, suivant le type des mesures auxquelles elles correspondent :

– Variables qualitatives nominales correspondent à un domaine de valeurs non ordonnées qui s'excluent mutuellement. Par exemple, la couleur de la peau.

– Variables qualitatives ordinales où les valeurs du domaine de définition peuvent être ordonnées. Par exemple le domaine des valeurs qui peuvent répondre à la question du degré

de satisfaction d'un client à propos d'un service fourni par un prestataire, soit D'={'Très_satisfait','Satisfait','Plutôt_satisfait','Insatisfait','Très_insatisfait'}.

– Variables qualitatives dont la mesure se fait sur des intervalles qui correspondent à une mesure ordinale où la distance entre deux valeurs peut toujours être calculée. Par exemple les catégories d'âges d'une population.

– Variables qualitatives où la mesure se fait par ratios. Ce type ressemble à celui des variables qualitatives dont la mesure se fait par intervalles. Dans ce cas, le domaine de définition contient une valeur centrale et des valeurs symétriques par rapport à cette valeur.

Ainsi, on distingue les modèles quantitatifs, qui opèrent sur des variables quantitatives, des modèles qualitatifs qui utilisent des variables qualitatives et définissent ainsi des comportements qualitatifs, étant définis par une suite ordonnée dans le temps des valeurs prises par une variable qualitative, souvent appelé scénario. On parle aussi de simulation qualitative par opposition aux simulations quantitatives. Par ailleurs, et souvent dans la plupart des cas, les modèles comportent un mélange de ces deux types de variables qui coexistent [Fishwick et Zeigler, 1992].

1.7.2. Hiérarchie des spécifications d'un modèle

Nous avons vu dans la section de la théorie de M&S que l'un de ses objectifs essentiels est de fournir une base méthodologique générale pour la conception de la simulation. Ainsi la modélisation d'un système dynamique se traduit par une spécification qui répond préalablement à la problématique posée. Cependant, le système peut être spécifié de plusieurs manières et avec des formalismes différents. A cet effet, la M&S vient nous offrir des possibilités pour organiser ces spécifications, indépendamment du formalisme utilisé, suivant les détails des informations dont on dispose sur le système, en allant du concept classique élémentaire de boite noire (voir la figure 1.7) à la description détaillée de la structure interne du système. Par ailleurs, Le niveau de spécification n'est pas le même suivant les connaissances que l'on a sur le système. Dans cette optique, Zeigler propose dans [Zeigler et al, 2000] une méthodologie pour organiser les spécifications d'un système de manière hiérarchique (table 1.1). Il y a lieu de citer aussi la hiérarchisation de Klir [Klir, 1985]. Bien que celle de Zeigler rajoute l'aspect temps. En intégrant explicitement le modèle du temps, cette hiérarchisation se voit très appropriée à la modélisation de systèmes dynamiques où la notion du temps et sa manière d'écoulement est une composante fondamentale.

Niveau	Non de la spécification	Connaissances
0	Observation	Les variables E/S essentielles et la façon de les mesurer suivant une base temporelle.
1	Comportement Entrées/Sorties (E/S)	L'ensemble de variables datées, collectées à partir du comportement du système source (Action/Réaction) : paires de valeurs E/S.
2	Fonctions E/S	Sachant un état initial, chaque entrée produit un résultat unique en sortie (cas des systèmes non stochastiques) : relations entre les E/S.
3	Transition d'état	L'évolution de l'état interne du système en fonction des entrées. La sortie générée par un état interne : la structure interne du système.
4	Couplage	Les composants système et la manière dont ils sont agencés: liaison entre ports E/S des modules : Structure hiérarchique.

Tab. 1.1 – Hiérarchie des spécifications d'un système dynamique

1.8. Conclusion

Ce chapitre présente un aperçu global de notions concernant la modélisation et la simulation (M&S). Nous y avons vu introduit un ensemble de définitions et de concepts généraux dont la compréhension est fondamentale pour ce domaine. Ainsi nous avons introduit la simulation comme étant un cadre d'expérimentation des systèmes. A cet effet, nous avons illustré les différentes étapes qui constituent l'élaboration d'une simulation informatique, en mettant en lumière la proposition de Shannon [Shannon, 1976, Shannon, 1998] d'énumération de ces étapes. Ensuite, nous sommes passés de la modélisation à la multi-modélisation où nous avons introduit la problématique de coexistence de plusieurs modèles hétérogènes dans un système. Nous avons vu, ainsi, comment bénéficier de la force de représentation des multi-formalismes. Par la suite, et par souci de couplage entre modèles, introduit dans le cadre de la multi-modélisation, nous avons cité une autre approche de la modélisation qui est la modélisation multi échelles.

Nous avons consacré, par la suite, une partie assez conséquente aux systèmes dynamiques. Ainsi, nous avons classé les différents modèles temporels qui les représentent. Pour chaque modèle temporel, nous avons identifié la technique de simulation appropriée. Nous avons vu aussi que la modélisation des systèmes dynamiques nécessite une certaine méthodologie qui offre des concepts et des formalismes mathématiques pour la représentation de cette catégorie de systèmes. C'est dans cet objectif que nous avons introduit la théorie de la M&S telle que proposée par Zeigler [Zeigler et al, 2000].

Dans le chapitre qui suit, nous allons présenter les formalismes qui s'inspirent de cette théorie et qui sont appropriés à la modélisation et à la simulation de la catégorie des systèmes à événements discrets, qui seront au centre de l'étude du chapitre suivant.

Chapitre 2

La simulation à évènements discrets

2.1. Introduction :

Nous avons passé en revue, dans le chapitre précédent ; différentes techniques de la simulation des systèmes dynamiques, qu'elles soient continues ou discrètes. Dans ce chapitre nous nous focalisons sur les systèmes à évènements discrets « SED », leurs caractéristiques, leurs exigences, la façon de leur développement et mise en œuvre et leur simulation. Nous allons aussi, présenter la notion de paradigme et de formalisme, ainsi que la définition de quelques formalismes souvent utilisés dans la modélisation des systèmes dynamiques.Aussi, l'accent sera mis sur deux formalismes qui vont faire l'objet de notre contribution (Voir chapitre 4) : Les Réseaux de Petri « RdP » et le formalisme DEVS « Discret EVents system Specification ».

2.2. La simulation discrète

La simulation discrète, comme vu dans le chapitre précédent, se distingue de celle continue du fait que l'état du modèle est statique durant des intervalles de temps déterminés, contrairement aux systèmes continus où l'état change d'une manière continue. Cela ne veut pas dire que l'état global du système discret ne change pas, mais ce qui nous intéresse là, ce sont desinstants biendéterminés et des variables d'état pertinentes. Par la notion d'abstraction, seuls les faits importants pour l'étude (certains événements) sont considérés. Ainsi, le modèle est considéré stable jusqu'à l'occurrence d'un événement (On parle alors des systèmes à événements discrets « SED »). Le mot « discrets » ne signifie ni « temps discret » ni « état discret », Ce sont les événements qui arrivent de manière discrète.

Les SED peuvent être classés selon leur comportement et la relation entre les entrées et sorties en :

- Systèmes combinatoires : où les sorties ne dépendent que des entrées.

$$\forall\, s \in S : Y = f(X)\ avec$$

$$S : le\ vecteur\ d'états\ possibles$$

$$Y: le\ vecteur\ de\ sorties$$

$$X: le\ vecteur\ d'entrées$$

- Systèmes séquentiels : où les sorties dépendent des entrées et des états antérieurs du système

$$S_{t+1}=f(S_t,X)\ \ et\ \ Y=g(S,X)$$

- Systèmes asynchrones : où le changement sur l'environnement est immédiatement signaléau système et suppose un traitement immédiat.
- Systèmes synchrones : où les événements aperçus parle système sont planifiés. Avec une horloge permettantla synchronisation et un ordonnanceur qui permet d'ordonner les événements par date d'occurrence.

Dans ce qui suit nous allons présenter les principes de la simulation des SED.

2.2.1. La simulation à événements discrets

Avant de d'entamer la détermination de la simulation des SED, il est convient de situer les modèles de cette catégorie de systèmes dans un arbre de modèle comme illustré figure 2.1.

En général, un modèle à événements discrets est caractérisé par trois caractéristiques :

- Il est stochastique : au moins quelques variables d'état sont aléatoires.
- Il est dynamique : l'évolution du temps des variables d'états est à considérer.
- Il réagit à des événements discrets : L'état change lors de l'occurrence des événements à des instants discrets.

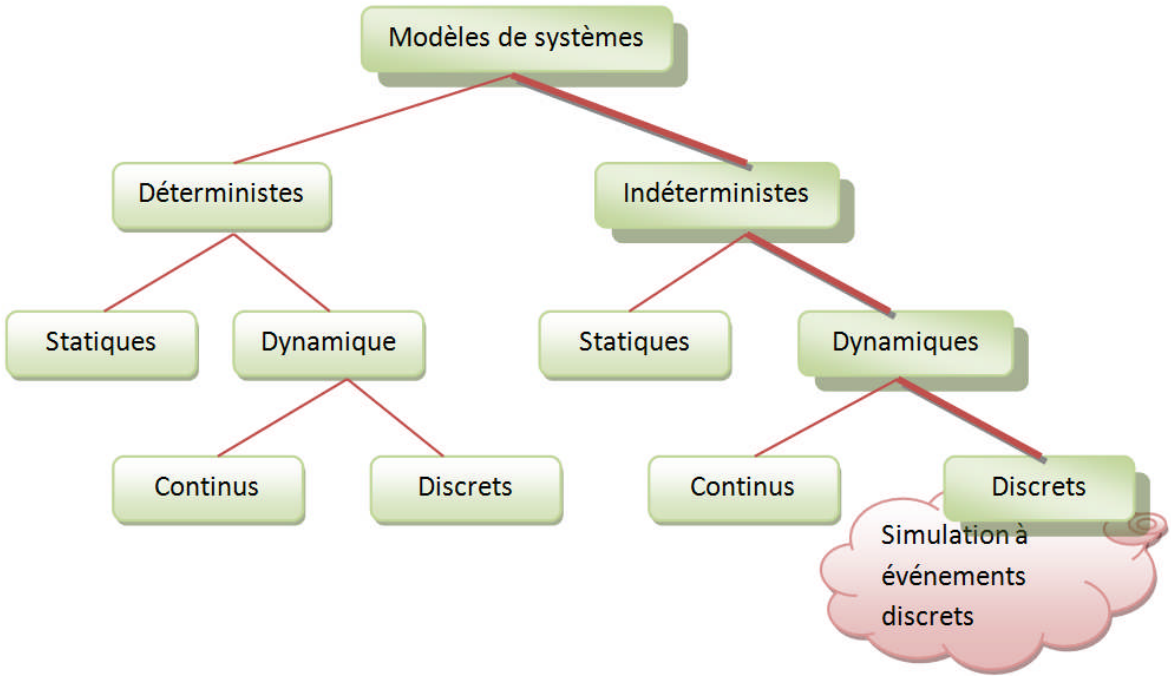

Fig 2.1 Taxonomie des modèles

Par ailleurs, la simulation des modèles représentant les SED consiste à calculer l'état du modèle à chaque occurrence (instant) d'évènement. L'état doit contenir les informations suivantes :

- Nombre d'entités dans le système.
- La date du dernier événement.
- La date du prochain évènement (initialement planifié).
- Autres informations sur le passé du système y compris les variables d'état.

2.2.1.1. Développement de modèle d'un SED

En général, le développement d'un modèle d'un SED est un processus complexe et ne se résume pas en un simple algorithme séquentiel. Il est utile de construire un croquis à haut niveau d'abstraction qui va servir d'aide au développement des modèles de SED. L'algorithme 2.1ci-dessous détermine les étapes (nous les appellerons les clés avec lesquelles on adopte les bonnes attitudes pour développer un modèle de SED). Cet algorithme est inspiré de celui de Shannon[Shannon, 1976, 1998].

- Algorithme 2.1
  Développement d'un modèle de SED
  (1) Après avoir mis les frontières du système sous l'étude, déterminer les objectifs (Des questions qui attendent des réponses).
  (2) En se basant sur (1), construire un modèle conceptuel du système (Quelles sont les variables d'état ? Comment elles sont-elles reliées les unes aux autres ? Quelles sont parmi elles les plus importantes ?
  (3) Convertir le modèle conceptuel en spécifications de modèle (Cette étape relève de la collecte des informations et l'analyse des données afin de construire le modèle à simuler. Cette étape est tellement critique que le reste du travail ne sera qu'une perte du temps si elle n'est pas bien procédée).
  (4) Convertir le modèle en un programme (La phase de l'implémentation : le choix du langage est primordial).
  (5) Vérification (Vérifier si le programme correspond bien aux spécifications préalables : Est-ce qu'on a correctement implémenté le modèle ?).
  (6) Validation (voir si le programme qui a répondu aux spécifications du modèle reflète le système sous l'étude ou non ? Autrement dit : Est-ce qu'on construit le bon modèle ?). Les étapes (5) et (6) sont très importantes pour le développement d'un modèle de SED. Si le modèle n'est pas satisfaisant, alors les étapes (2), (3) et (4) seront mises en jeu et le processus sera itératif tant que besoin).

  Notons que les étapes (5) et (6) sont présentes dans l'esprit du développeur durant tout le processus de conception du modèle. Cette attitude offre des corrections rapides plutôt que d'arriver à la fin et se trouver surpris que le

modèle –souvent couteux– ne corresponde nullement aux spécifications ou aux objectifs préalablement tracés.

La figure 2.2 illustre la méthodologie de conception classique en « V » qui correspond bien à l'algorithme 1.1 où les tests de vérification et de validation sont présents dans toutes les étapes. Par ailleurs, le modèle de SED doit être simplifié et le développeur doit pouvoir extraire uniquement les faits, les événements, les variables et les états qui sont pertinents. Cependant cette simplification ne doit en aucun cas ignorer les objectifs que le modèle doit, en fin de compte, atteindre. C'est un compromis que le développeur doit impérativement étudier pour bien se situer.

A noter que les étapes (5) et (6) peuvent être fusionnées lors de la conception de la simulation des systèmes complexes à évènements discrets. Dans ce cas, la décomposition du modèle en sous modèles est inévitable. En outre, ce genre de systèmes ne peut pas être modélisé par un simple algorithme séquentiel.

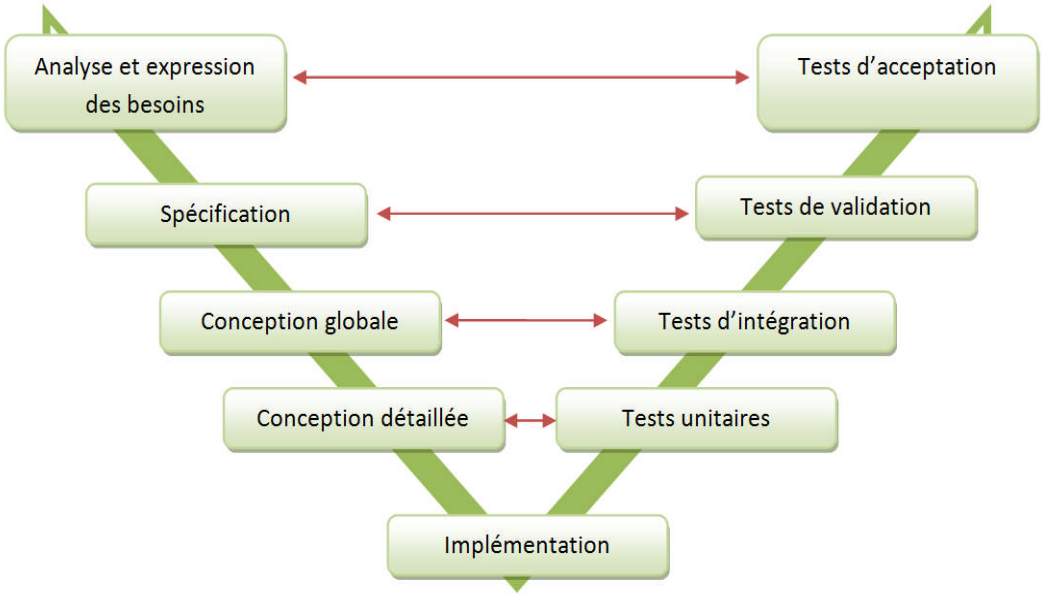

Fig 2.2 Cycle classique de conception en « V »

2.2.1.2. Simulation du modèle d'un SED

Si les étapes de l'algorithme 2.1 sont bien respectées et que l'on a un modèle correctement converti en un programme, alors on peut passer à l'étape de la simulation ou de la mise en évolution dans le temps du modèle. L'algorithme 2.2 (inspiré de celui de Shannon[Shannon, 1976, 1998]) résume cette démarche en quelques lignes.

- Algorithme 2.2
  Simulation d'un modèle de SED
  (1) Etablir le plan d'expérimentations (Cette tâche est difficile si on aura un nombre important des paramètres à étudier. On pourra donc, tomber dans une infinité de combinaisons).

(2) Procéder à l'exécution et aux rapports des résultats.
(3) Analyser les résultats.
(4) Décider à propos des résultats, s'ils sont satisfaisants alors passer à l'étape (5) sinon voir quelles étapes doit être mises en jeu et réitérée entre (1) et (2).
(5) Documenter les résultats afin de rendre leur exploitation facile.
(6) Exploiter les résultats (Pour les futurs systèmes, il s'agit de les implémenter).

La figure 2.3 résume graphiquement les deux précédents.

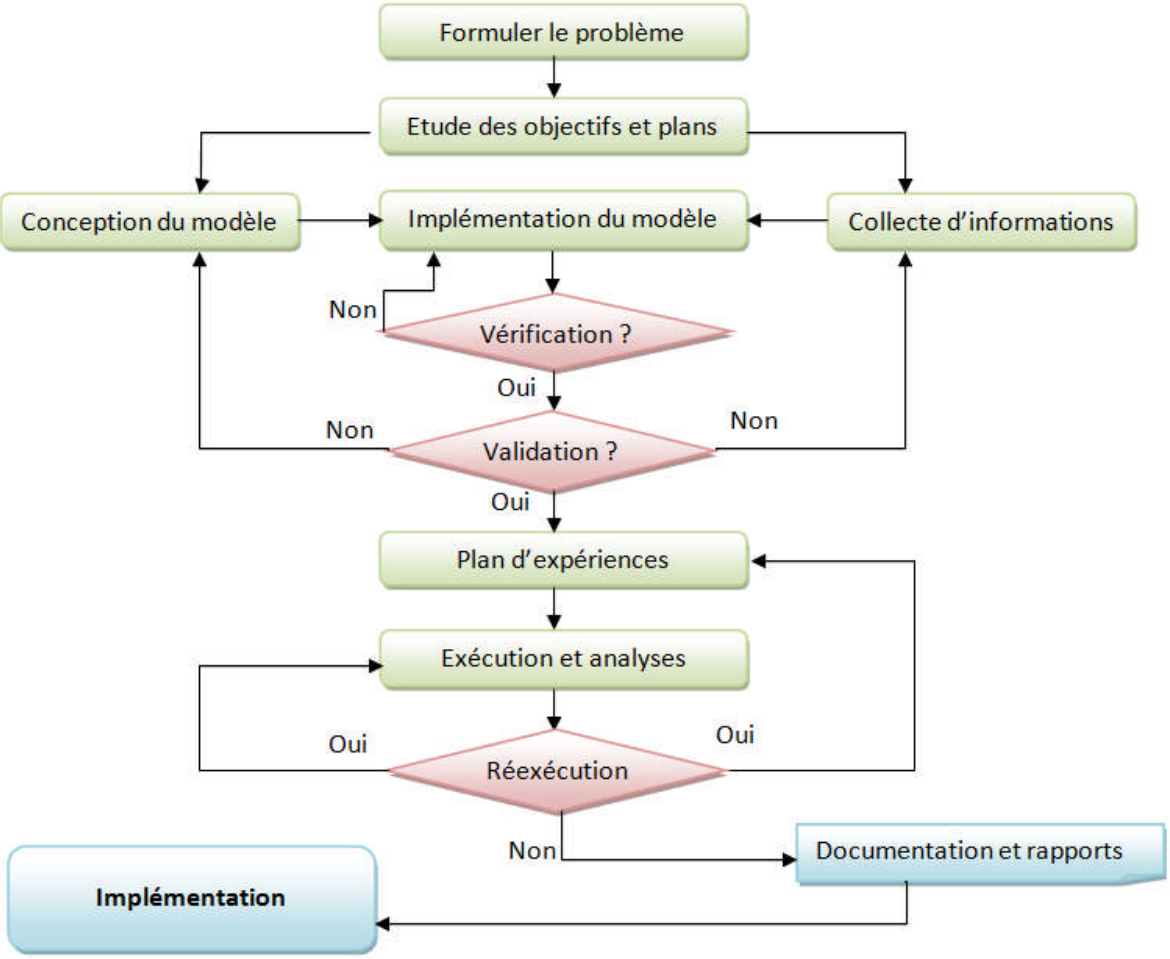

Fig 2.3. Etapes de conception et de simulation d'un modèle de SED

2.2.1.3. Les générateurs des nombres aléatoires

Nous avons vu dans les sections précédentes comment modéliser et simuler les modèles des SED. Cependant un point très important sur lequel se basent les SED doit être suffisamment discuté : C'est l'aspect stochastique d'arrivée des évènements. Si cette caractéristique n'est pas respectée alors tout le processus de la M&S sera mis en jeu et les résultats de la simulation ne reflèteront nullement la réalité. A cet effet on aura recours aux générateurs des nombres aléatoires.

Les générateurs des nombres aléatoires sont des programmes capables de produire des nombres aléatoirement, généralement compris entre 0 et 1. Avec des outils mathématiques on peut transformer ces nombres en un indice d'occurrences des événements. Il existe plusieurs techniques sur lesquelles sont basés les générateurs des nombres aléatoires. Nous citons ici, quelques-unes :

- **Récurrences linéaires simple :** Il s'agit des générateurs de la forme :

$$X_i = aX_{i-1} + c \pmod{m}$$

Avec $U_i = X_i/m$. La période d'un tel générateur est bien entendu inférieure à m. Si c=0 sa période est même inférieure ou égale à m−1 car 0 est le point fixe. On ne considérera que deux cas, qui sont les plus utilisés dans la pratique[Lüscher, 1994].La table 2.1 montre quelques exemples [James, 1990] :

Méthode	Paramètres	Observation
Lehmer	a = 23, m =10^8 + 1	assez moyen
RANDU	a = 65539, m = 2^{31}	mauvais
Marsaglia	a = 69069, m = 2^{32}	raisonnable
SURAND	a = 16807, m = $2^{31} - 1$	Raisonnable
INMOS	a = 1664525, m = 2^{32}	le meilleur a connu pour ce
L'Ecuyer	a = 41358, m = $2^{31} - 1$	bon
CDC	a = 5^{15}, m = 2^{47}	longue période

Table 2.1 Quelques valeurs utilisées pour m et a, (c = 0).

- **Récurrences linéaires multiples:** Elles sont définies par la relation :

$$X_i = \sum_{j=1}^{k} a_j X_{i-j} \pmod{m}$$

La période d'un tel générateur est $m^k - 1$ si m est premier et $a_1, \ldots .. a_k$ bien choisis. Noter que la période est bien plus grande que le nombre de valeurs prises. Un exemple est le générateur de Marsaglia et Zaman : $X_n = X_{n-r} - X_{n-s}$ *(mod b)*. Les valeurs proposées par Lüscher[Lüscher, 1994] sont :

$$b = 2^{24},\ r = 24,\ s = 10.$$

- **Décalage de registre:** Dans cette technique,on opère directement sur la représentation binaire des réels dans l'intervalle [0, 1]:

$$u = 0 + \frac{b_1}{2} + \frac{b_2}{4} + ... + \frac{b_l}{2^l} = \langle 0, b_1, ...b_l \rangle$$

On réalise une suite de nombres binaires bi ∈ {0, 1} par la récurrence :

$$b_i = a_1 b_{i-1} + ... + a_q b_{i-q} \pmod 2, \quad a_i \in \{0, 1\}$$

Puis on calcule :

$$u_i = \langle 0, b_{il+1}, b_{il+2}, ...b_{il+l}, \rangle$$

Sachant que :

$$\underbrace{b_1 \ldots . b_l}_{u_0} \underbrace{b_{l+1} \ldots . b_{2l}}_{u_1} \underbrace{b_{2l+1} \ldots . b_{3l}}_{u_2} ...$$

On constate qu'il y'a 2^q suites (b_{i-1}, ...b_{i-q}) différentes possibles, et comme zéro est point fixe, la période des b_i est au plus égale à $2^q - 1$; on choisira toujours un générateur ayant cette période. Pour que la période de la suite u_i soit = $2^q - 1$, il faudra utiliser effectivement toutes les suites (b_{i+1}, ...b_{i+l}) apparaissant (et non pas une sur l), pour cela on prendra l tel que :

$$Pgcd(l, 2^q - 1) = 1.$$

- **Loi discrète:** Soit une variable à générer qui prend les valeurs 1, 2, ...r avec les probabilités respectivesp_1, p_2, ...p_r : P(X = i) = p_i. On commence par générer une suite uniforme indépendante U_n puis on pose :

$$X_n = i \quad \text{si} \quad p_1 + \cdots + p_{i-1} < U_n \leq p_1 + \cdots + p_i.$$

Si les U_n sont i.i.d U([0, 1]), les X_n sont clairement indépendantes avec la loi voulue.En pratique on testera successivement $p_1 > U_n$ puis, $p_1 + p_2 > U_n$, etc... Pour que la simulation soit laplus rapide possible, il faudra donc présenter les p_i par ordre décroissant.

Nous avons cité dans cette section quelques bases mathématiques sur lesquelles sont fondés la plupart des générateurs de nombres aléatoires. Néanmoins, d'autres théorèmes et techniques mathématiques assez intéressants peuvent être trouvés dans [Matsumoto et Nishimura, 1998 ; Ecuyer, 1998 ; Ecuyer et al, 1998 ; James, 1990 ; Lüscher, 1994].

2.3. Paradigmes et formalismes

2.3.1. Le paradigme

Le paradigme est un cadre général de raisonnement. Il peut être défini comme étant un point de vue qui va diriger la conception du modèle. C'est en quelque sorte une vision scientifique et un référentiel de pensée qui peut être emprunté par le modélisateur afin de concevoir son modèle. Par exemple : les paradigmes orientés objet, agents, acteurs…etc. L'historien des sciences Thomas S. Kuhn [Kuhn, 1972] a défini le terme paradigme comme étant :

"*certains des exemples acceptés de pratiques scientifiques - ce qui inclut lois, théories, applications et instrumentations- qui fournissent des modèles desquels proviennent des traditions particulières et cohérentes de recherche scientifiques*".

2.3.2. Le formalisme

La construction d'un modèle suivant un paradigme nécessite sa description d'une manière formelle. Ainsi le langage formel particulier permettant cette description est appelé formalisme. En outre, pour un paradigme donné, il peut exister plusieurs formalismes. Généralement, ceux-ci sont classifiés selon plusieurs critères : la représentation et la gestion du temps, la représentation de l'espace et des objets modélisés et le rapport au hasard [Milleret al, 2004; Quesnel, 2006]. La figure 2.4 illustre une classification des formalismes selon E. Ramat[Ramat, 2006].

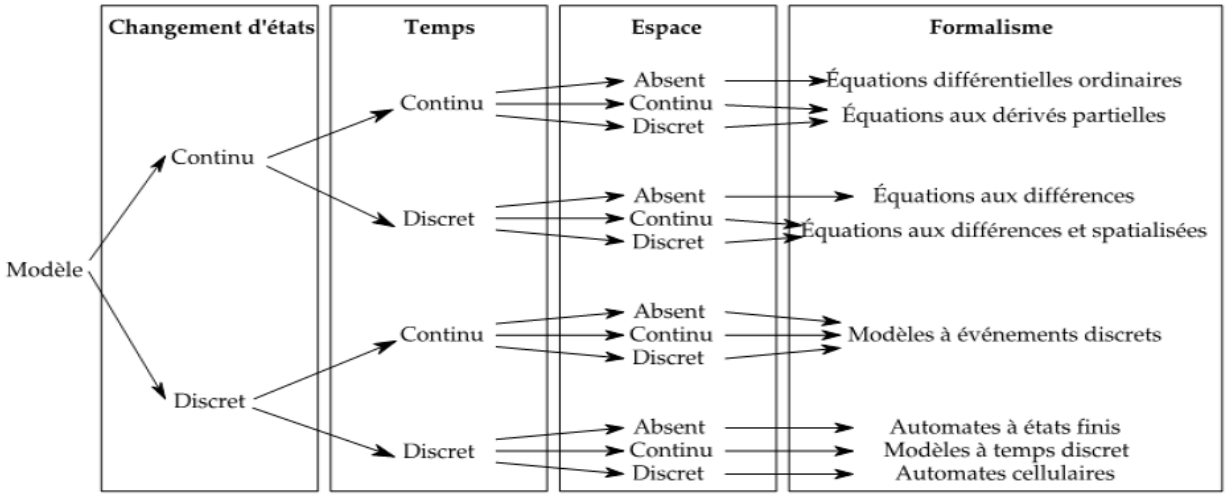

Fig.2.4. Classification des formalismes selon les aspects continus ou discrets des variables du temps et de l'espace [Ramat, 2006].

Dans ce qui suit nous allons citer les formalismes les plus utilisés dans la modélisation de différents systèmes. Nous discuterons, ainsi, de quels types de systèmes sont les mieux adaptés pour chacun des formalismes.

2.3.2.1. Les automates à états finis

Les automates à états finis permettent de spécifier formellement la dynamique des systèmes à l'aide des notions d'état et de transition. Une écriture simple de ce formalisme peut être faite à l'aide de l'équation A=(X, S, Δ) où X et S sont respectivement les entrées du modèle et les états du système. La fonction Δ décrit la fonction de transition en indiquant l'état suivant en fonction de l'état courant et de l'entrée du système. Les entrées sont des symboles, mais il est facile de passer de la notion de symbole à celle d'événement et ainsi de disposer d'un outil de spécification à événements discrets. On peut représenter un automate à états finis à l'aide d'un graphe où les sommets représentent les états et les arcs les transitions. Les automates à événements finis constituent un point de vue complémentaire où les sommets représentent les événements et les arcs la succession possible des événements. Les arcs portent, en général, une durée. Cette durée représente le temps qui peut s'écouler entre deux événements. La figure 2.5 illustre un exemple d'automate à états finisreprésentant l'état civil d'un citoyen.

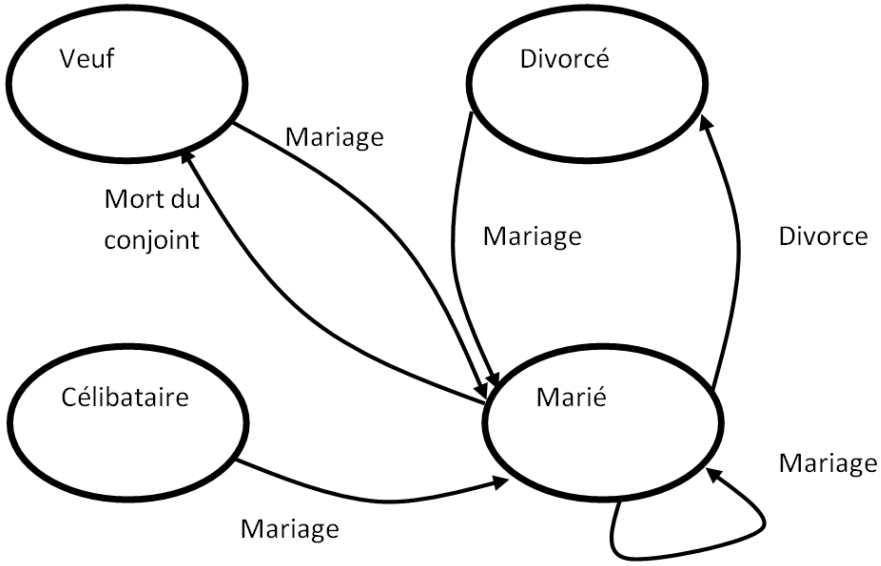

Fig. 2.5. Automate à états finis de l'état civil d'un citoyen

2.3.2.2. Les systèmes d'équations différentielles

Les équations différentielles se caractérisent par leur force d'expressions des mathématiques et des outils formels qui les accompagnent. A partir d'un système d'équations différentielles, on peut déduire des propriétés de convergence ou de stabilité à l'infini sous certaines conditions. On peut également manipuler le système avec tous ces paramètres et en en déduire des propriétés. Il est, en effet, facile si les équations ont de bonnes propriétés de déduire l'évolution temporelle des variables composant le système.

L'inconvénient des équations différentielles est qu'il n'existe pas toujours le bon théorème ou le bon outil pour le système que l'on construit. Dans ce cas, on doit passer par

des outils de simulation et d'analyse numérique. Le deuxième désavantage concerne la représentation d'entités discrètes. Il est très difficile, voire impossible, avec les outils mathématiques courants, de représenter des entités discrètes et de représenter les processus les mettant en jeu. Par ailleurs, les équations différentielles sont, dans la grande majorité des cas, utilisées pour représenter des lois ou des contraintes entre variables continues du système. Ce qui les rend plus appropriées pour les systèmes continus.

2.3.2.3. Les automates cellulaires

Les automates cellulaires décrivent des systèmes où l'espace doit être considéré. Ce formalisme hybride décrit la structure spatiale du système à l'aide d'une grille multidimensionnelle de cellules interconnectées et décrit le comportement local de chaque cellule de l'automate, en fonction de son voisinage. Les automates cellulaires constituent l'un des formalismes les plus utilisés en modélisation des systèmes spatiaux. Néanmoins, ce formalisme montre une faiblesse de représentation de l'espace. Il le considère comme étant discret et non continu. A cet effet, ce formalisme se trouve mieux approprié pour les systèmes discrets où l'aspect de l'espace doit être considéré.

2.3.2.4. Les stat-charts

La modélisation par UML offre une extension objet des automates à états finis : les stat-charts. Où l'expression des transitions peut se construire autour des attributs de la classe des entités. Le formalisme des stat-charts peut se définir comme étant une combinaison des formalismes à base d'états et de transitions tels que les automates à états finis. Parmi ces transitions, on trouve: les transitions déclenchées sur des événements, les transitions temporisées et les transitions conditionnelles. L'ensemble de ce type de transitions est présent dans DEVS, ce qui argumente l'idée que les stat-chartes sont des cas exceptionnels du formalisme DEVS. En effet, il est possible de définir une spécification DEVS d'une partie des stat-charts [Borland et al, 2003]. En outre, les stat-charts se caractérisent par la possibilité de faire décomposer un état en sous-états (voir [Fishwick, 1995]). Ainsi ils proposent une syntaxe relativement riche pour la modélisation de la dynamique d'un système à événements discrets.

2.3.2.5. Les réseaux de Petri

Les réseaux de Petri « RdP » ou PN pour « Petri net » constituent un formalisme de modélisation, initialement développé par [Peterson, 1977]. Ils sont très appropriés pour la modélisation des systèmes dynamiques. Les PN sont des graphes orientés avec deux types de sommets: les places et les transitions, reliés entre elles par des arcs. Un arc ne doit jamais relier deux sommets du même type. Les places représentent -dans la plupart des cas- l'aspect statique du système, tandis que les transitions sont responsables de sa dynamique. Chaque place peut contenir n jetons (n Є N avec N ensemble des entiers positifs). Les arcs peuvent être marqués par un poids (entier strictement positif), on parle alors de « PN généralisés ». Si le poids de tous les arcs est égal à 1, alors le PN est dit « ordinaire ».

Une transition est dite valide si toutes les placesen amont contiennent chacune un nombre de jetons ≥ au poids de l'arc qui la relie à cette transition. Une transition valide peut être franchie. Le franchissement ou le tir d'une transition se traduit par deux actions simultanées: la première consiste en la destruction des jetons des places se situant en amont de l'arc reliant la place à la transition franchie. La deuxième génère autant de jetons dans les places se situant en aval de la transition que le poids de l'arc reliant cette transition à ces places.

L'évolution d'un RdP correspond à l'évolution de son marquage au cours du temps (évolution de l'´état du système): il se traduit par un déplacement des marques ce qui s'interprète comme la consommation/production de ressources déclenchée par des évènements ou des actions. Déterminer l'évolution d'un RdP correspond en fait à le simuler.

2.3.2.5.1. Spécification formelle d'un réseau de Petri

Il existe plusieurs définitions formelles pour spécifier les PN. Nous donnons ci-après celle qui définit un PN comme étant un quintuple:

PN=(P,T,PRE,POST,M0)

P : est l'ensemble des places.

T : est l'ensemble des transitions.

PRE : est la matrice générée par l'application PxT→N.

PRE[i,j]=n / n=0 si la place pi n'est pas en amont de la transition tj sinon n=τ / τ est le poids de l'arc reliant pi à tj.

POST : est la matrice générée par l'application TxP→N.

POST[i,j]=n / n=0 si la place pi n'est pas en aval de la transition tj sinon n=τ / τ est le poids de l'arc reliant tj à pi.

M0 : est le vecteur de marquage initial.

M[i]=k / k est le nombre de jetons dans la place pi.

La figure 2.6 montre un exemple de PN correspondant au système Producteur-Consommateur avec un tampon de sept cases. Le producteur est modélisé par deux places P1 et P2 et deux transitions T1 et T2. Si P1 contient un jeton ça veut dire que le producteur est prêt à produire. T1 et T2 représentent respectivement le début et la fin de production. Quant à P1 est la place qui signifie que le producteur est à l'état de production. La fin de production d'un produit se traduit par le dépôt d'un jeton dans la place P3 (le tampon) ainsi et si le consommateur est prêt à consommer (P4 contient un jeton) T3 peut être franchie (le début de consommation). La place P6 représente le nombre de cases vides dans le tampon. Si cette dernière ne contient aucun jeton, la transition T1 ne peut pas être franchie ce qui signifie que

le tampon doit avoir au moins une case vide pour que le producteur puisse recommencer la production.

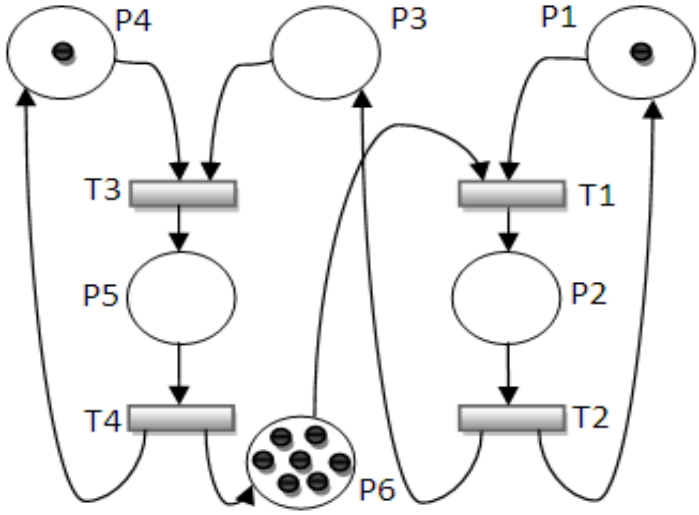

Fig. 2.6. Modèle RdP du système Producteur-Consommateur

2.3.2.5.2. Structure et caractéristiques des RdP

Dans cette partie nous allons présenter des caractéristiques particulières des RdP :

- **RdP borné** : Un RdP est dit k-borné pour un marquage initial M_0 si le nombre de jetons dans chacune de l'ensemble de toutes les places ne dépasse pas k quel que soit la suite de franchissement à partie de M_0. Notons que cette caractéristique dépend toujours du marquage initial.
- **RdP sauf ou binaire** : Si Un RdP, pour un marquage initial M_0, est 1-borné, alors il est dit sauf ou binaire.
- **RdP vivant** : Une transition T est dite vivante, pour un marquage initial M_0, s'il existe toujours une suite de franchissements qui amène au franchissement de T à tout instant de l'évolution du RdP pour M_0. Par conséquence, si toutes les transitions d'un RdP sont vivantes pour un marquage initial M_0, alors ce RdP sera dit « vivant ».
- **RdP conforme** : Un RdP est conforme pour un marquage initial M_0, s'il est, à la fois, binaire et vivant.
- **Le blocage** : Le blocage est un marquage pour lequel aucune transition ne peut être validée. Cet état est souvent appelé « état puits ».
- **Le Conflit** : Un RdP est dit « avec conflit » s'il comprend au moins une place qui est on amont d'au moins de deux transition. La figure 2.7 illustre un RdP sans conflit à gauche (a) et un RdP avec conflit (b).

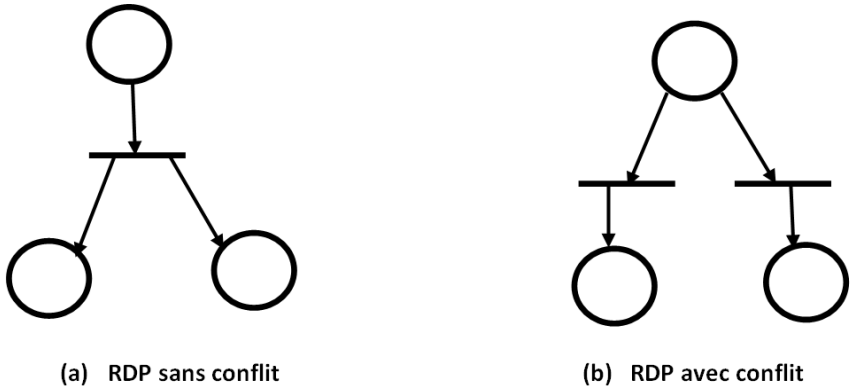

Fig. 2.7. RdP avec et sans conflit

- **RdP pur** : Un RdP est dit « pur » s'il ne comprend pas une place qui est en amont et en aval d'une même transition. Par opposition, tout RdP contenant au moins une place liée à une même transition par deux arcs : un en entrée et l'autre en sortie, est un RdP impur. Notons que tous les RdP impurs peuvent être transformés en des RdP purs équivalents. La figure 2.8 montre un RdP impur (a) et le RdP pur équivalent (b).

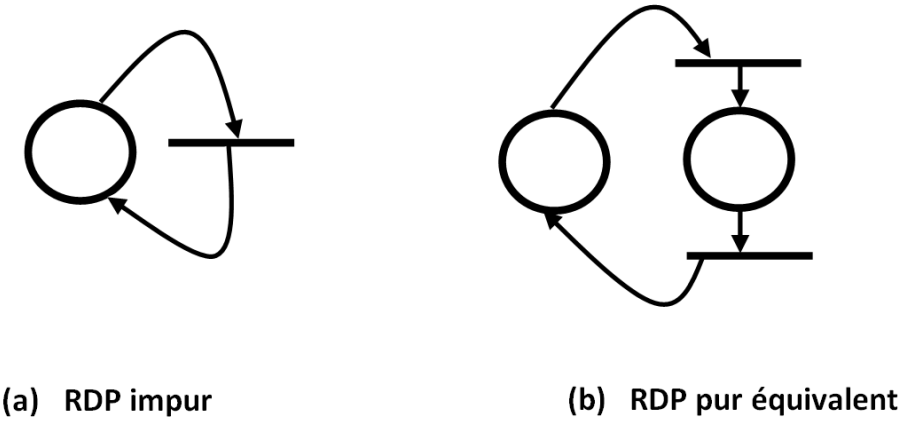

Fig. 2.8RdP impur et son RdP pur équivalent

- **Le parallélisme :** Cette caractéristique détermine la possibilité de lancer plusieurs processus simultanément. Dansla figure 2.9, le franchissement de T_début met un jeton dans chacune des P1, p2 et éventuellement d'autres places (Pn), représentant, ainsi, le déclanchement de plusieurs processus (Pss1, Pss2,…,Pssn) en même temps. Si les processus finissent par une même transition (T_fin par exemple), alors là on parle de la synchronisation de l'achèvement de l'ensemble des processus.

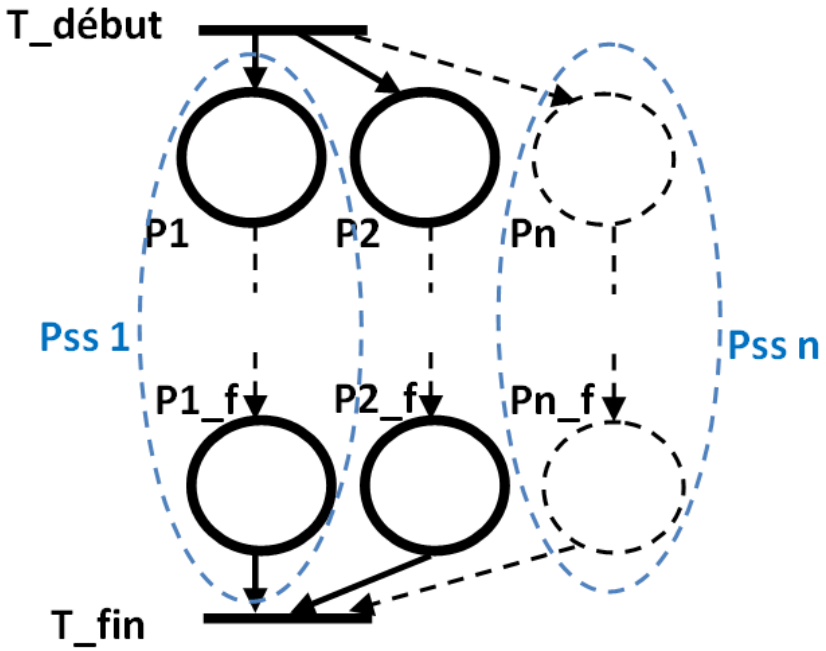

Fig. 2.9. Le parallélisme dans les RdP

- **La synchronisation mutuelle:** Elle est souvent appelée **le point de rendez-vous :** C'est le fait que plusieurs processus doivent s'attendre les uns les autres en une étape dans leur évolution. Par exemple : le test de mise en marche d'un moteur doit attendre le montage de toutes les composantes. Dans la figure 2.9, la transition T_fin (le point de rendez-vous) ne peut être franchie que s'il y a au moins un jeton dans chacune des places qui sont en amont soient (p1_f, p2_f,…,pn_f). Ce qui signifie que les processus qui ont mis des jetons dans leurs places finales Pi_f doivent attendre les autres qui ne sont pas encore arrivés à cette étape.
- **Le sémaphore :** Cette caractéristique assure qu'un processus Pss1 doit attendre un autre processus Pss2 qu'il atteigne une certaine étape de son évolution. Cela rend l'évolution de Pss1 dépendante de Pss2, alors que l'évolution de Pss2 ne dépend nullement du Pss1 (voir la figure 2.10).

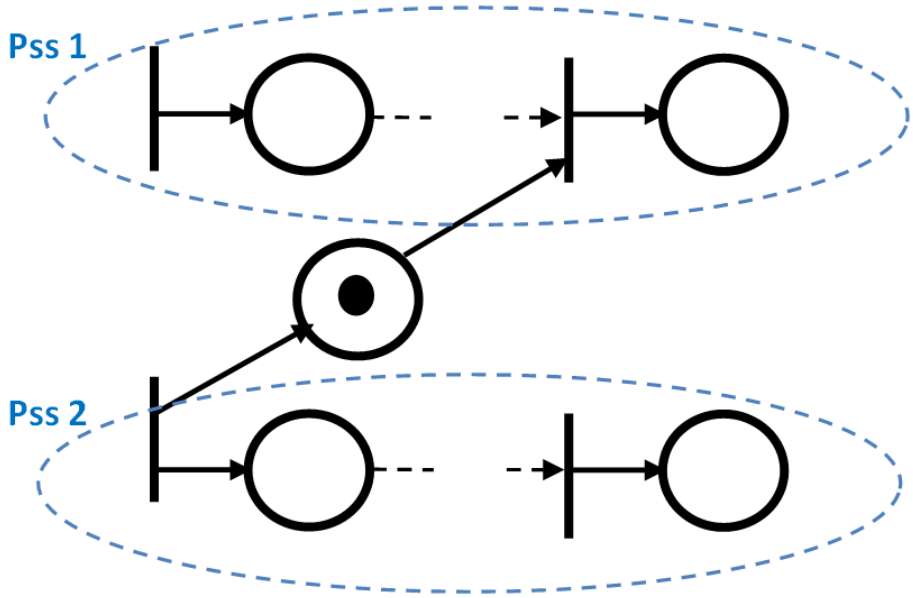

Fig. 2.10. Le sémaphore dans les RdP

Les formalismes que nous avons présentés dans cette section sont à titre indicatif plutôt qu'exhaustif. Il y a d'autre formalismes et langages qui n'ont pas moins de valeur. Cependant, nous avons détaillé les RdP d'avantage puisque ce formalisme est fondamental dans notre contribution de transformation entre formalismes. En fait, cette section peut être vue comme étant une introduction afin d'aborder le formalisme DEVS sur lequel nous focalisons nos travaux (voir chapitre 4).

2.3.2.6. Le formalisme DEVS

2.3.2.6.1. Introduction

DEVS est un formalisme modulaire, abstrait et indépendant de toute implémentation. Il permet la modélisation des systèmes causaux et déterministes [Zeigler, 2000; Ramat,2003]. Un modèle atomique DEVS est basé sur un temps continu, des entrées, des sorties, des états et des fonctions (sortie, transition, durée de vie des états). Des modèles plus complexes sont construits en connectant plusieurs modèles atomiques de façon hiérarchique. Les interactions sont assurées à travers les ports d'entrée et de sortie des modèles, ce qui favorise la modularité.

2.3.2.6.2. Les modèles atomiques DEVS

AtomicDEVS = $(X,Y,S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, l_t)$

– La base de temps est continue et n'est pas explicite : $T = \Re$;

– X est l'ensemble des entrées externes. Elles peuvent interrompre le comportement autonome du modèle, tel que prévu par la fonction de transition externe δ_{ext}. La réaction du système par rapport à un événement externe dépend de l'état courant du système, l'entrée et le temps écoulé dans cet état ;

– Y est l'ensemble des sorties du modèle ;

– S représente l'ensemble des états séquentiels : la dynamique consiste en une séquence ordonnée d'états de S ;

– δ_{int} est la fonction de transition interne qui fait passer le système d'un état à un autre de manière autonome. Elle dépend du temps passé dans l'état et des conditions;

– δ_{ext} est la fonction de transition externe;

– δ_{con} est la fonction de transition de conflit;

– λ est la fonction de sortie du modèle.

– $l_t(s)$ représente la durée de vie d'un état « s » du système ;

Un modèle DEVS peut être représenté graphiquement. La figure 2.11 représente un modèle atomique par un rectangle avec un ensemble d'entrées appelées ports d'entrées et de sorties appelées ports de sortie, représentés par des triangles remplis. La valeur vi est la valeur prise par un port d'entrée ou de sortie. Cette valeur appartient à l'ensemble des valeurs possibles du port pi. Un port d'entrée prend une valeur lors de l'émission d'un événement attaché à ce port. Un port de sortie prend une valeur lorsque la fonction de sortie prend une valeur pour ce port.

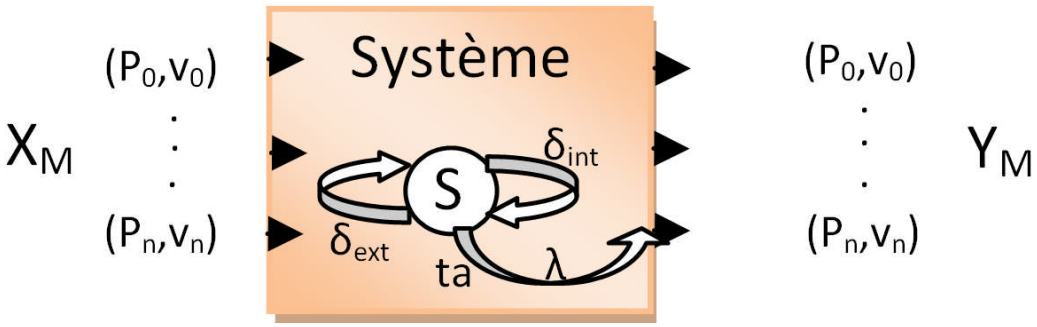

Fig.2.11 Représentation graphique d'un modèle DEVS atomique

S est l'ensemble des états du système. L'ensemble des valeurs prises par le vecteur d'état à un instant donnée est appelé état du système. L'ensemble S_T des états totaux du système est :

$S_T = \{(s, e)|s \in S, 0 < e < l_t(s)\}$

Où e représente le temps écoulé dans l'état s. Ce concept d'état total (s, e) est fondamental car il permet de spécifier un état futur en fonction du temps écoulé dans l'état présent. $l_t(s_k)$ est le temps pendant lequel le modèle restera dans l'état s_k, si aucun événement externe ne survient.

Un état s de S est dit passif si et seulement si sa durée de vie est infinie. Dans ce cas, la fonction de transition interne n'est pas définie. La fonction classique de transition est décomposée en deux fonctions, représentant respectivement les évolutions autonomes et celles dues à des événements externes.

La fonction de transition interne est définie par : δ_{int}: S → S.

Elle spécifie les états futurs des états actifs. Ces états correspondent à des régimes transitoires ou des états instables du système.

La fonction de transition externe est : δ_{ext}: S x X → S.

Elle représente la réponse du système aux événements d'entrée. Le système est dans un état (s, e) à un instant t. Lorsqu'un événement externe survient, cette fonction retourne le nouvel état du système en fonction de s et de l'événement aperçu.

Cette décomposition de la fonction de transition en deux fonctions constitue l'un des points forts du formalisme DEVS en matière de spécification des évolutions autonomes du modèle. La spécification des changements d'états est complétée par la fonction de conflit, notée :

δ_{con}: S x X → S.

Cette fonction permet de spécifier l'état futur dans le cas où un événement est survenu en même temps que l'état du système vient de terminer sa durée de vie. C'est le cas où les deux fonctions δ_{ext} et δ_{int} seront en conflit pour déterminer le nouvel état. Cette fonction est présente seulement dans une variante de DEVS :Parallel DEVS [Zeigler, 2000].

Notons que dans DEVS classique avec ports, un port p reçoit une seule valeur v d'un événement externe. Avec le formalisme DEVS parallèle [Chow et Barros, 1994; Chow &Zeigler, 1994], un modèle DEVS peut recevoir sur un port d'entrée donné plusieurs valeurs simultanées. La deuxième tâche de la fonction δ_{con} est de gérer les événements simultanés en appliquant un ordre de priorité entre ces événements.

La fonction de sortie est une application de l'ensemble des états S dans l'ensemble des sorties Y. Elle est définie par :

λ: S → Y

Cette fonction sera activée lorsque le temps écoulé dans un état donné sera égal à sa durée de vie.

Un système peut être autonome et donc ne recevoir aucun événement extérieur. La dynamique du système est alors le seul fait de la fonction de transition interne. Cette fonction de transition est définie pour spécifier les changements d'états dus exclusivement à l'état interne du système et au temps.

La figure2.12illustre l'évolution d'un modèle DEVS sur un exemple.

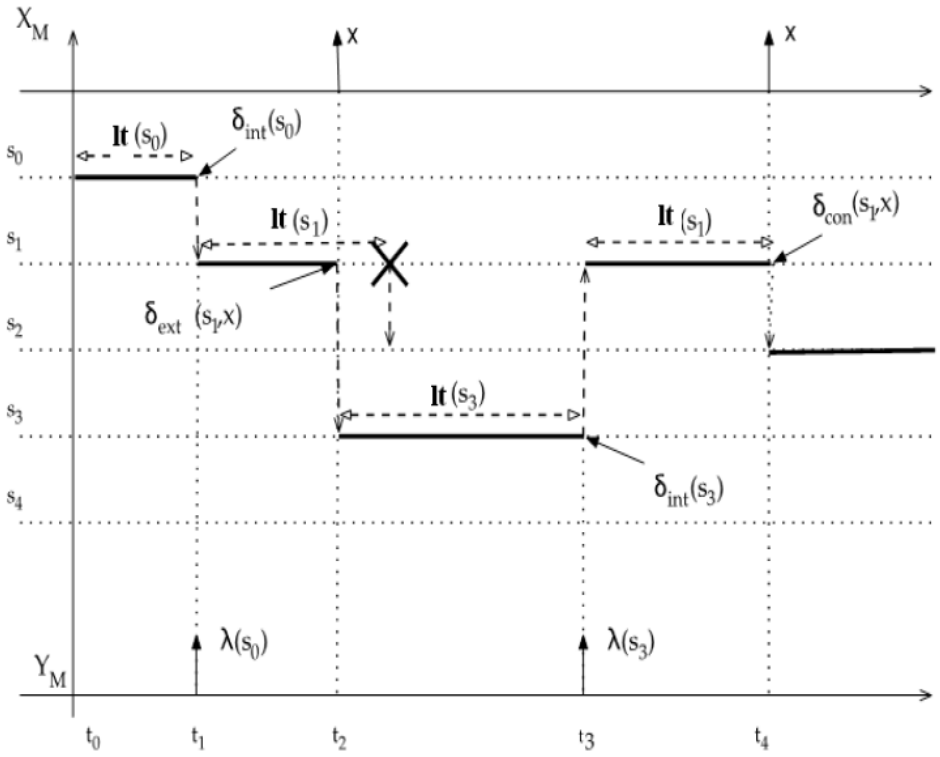

Fig.2.12 Exemple de graphe de transition d'un modèle DEVS.

A l'état initial t_0, le système est dans l'état s_0. La fonction « l_t » nous indique que pour l'état s_0, le système changera d'état à $t + l_t(s_0)$ si aucun événement extérieur ne survient. A $t_1 = t + l_t(s_0)$, aucune entrée n'a eu lieu. La fonction de sortie est donc activée et Y prend pour valeur la valeur produite par la fonction λ pour l'état s_0. Après avoir affecté les ports de sortie, la fonction de la transition interne δ_{int} est appliquée. Le système passe dans l'état $s_1 = \delta_{int}(s_0)$ et changera d'état à $t_1 + l_t(s_1)$. A l'instant t_2, qui est inférieur à $t_1 + l_t(s_1)$, un événement extérieur est placé en entrée. Donc c'est la fonction de transition externe qui détermine le nouvel état. Dans ce cas, la fonction de sortie n'est pas appliquée. Elle est appliquée exclusivement lors de transition interne. A l'instant t_2, le système passe dans l'état $s_3 = \delta_{ext}(s_1)$. Si aucun événement externe n'avait eu lieu, le système serait à $t_1 + l_t(s_1)$ dans l'état $s_2 = \delta_{int}(s_1)$. Le système évolue ensuite jusqu'à l'instant t_4 par transition interne. A l'instant t_4, la fonction de transition interne doit être activée mais au même moment un événement extérieur survient. La fonction δ_{con} règle le conflit en indiquant le nouvel état. Ce nouvel état est

fonction de l'état courant et de l'événement d'entrée. La fonction de sortie ne prend pas de valeur.

2.3.2.6.3. Les modèles couplés DEVS

Le formalisme du DEVS couplé décrit un système à événements discrets dans les termes d'un réseau de composants couplés.

CoupledDevs = (X_{self}, Y_{self} ,D, {M_d/dЄD}, EIC, EOC, IC).

S_{elf}: représente le modèle lui-même.

– X_{self} est l'ensemble des entrées possibles du modèle couplé ;

– Y_{self} est l'ensemble des sorties possibles du modèle couplé ;

– D est l'ensemble des noms associés aux composants du modèle, self n'étant pas dans D;

– {M_d/dЄD} est l'ensemble des composants du modèle couplé, tel que d appartient à D;

– EIC « Extenal Intput Coupling », EOC « Extenal Output Coupling » et IC « Intenal Coupling » définissent la structure de couplage dans le modèle couplé;

– EIC est l'ensemble des couplages externes en entrée. Ils connectent les entrées du modèle couplé à ceux de ses propres composants;

– EOC est l'ensemble des couplages externes en sortie. Ils connectent les sorties des composants à ceux du modèle couplé;

– IC définit le couplage interne. Il connecte les sorties des composants avec les entrées d'autres composants dans le même modèle couplé.

Pour illustrer le modèle DEVS couplé, nous proposons un exemple simple dont la représentation graphique est fournie sur la figure 2.13.

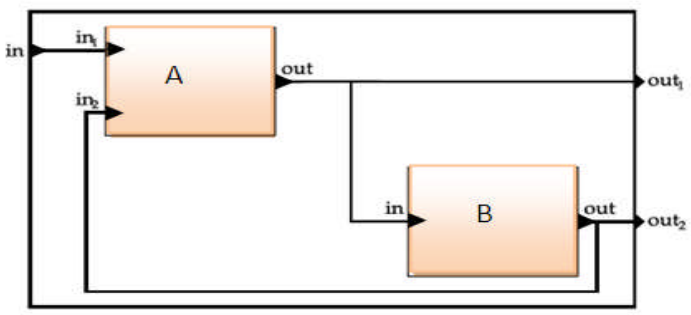

Fig.2.13 Représentation graphique d'un modèle DEVS couplé.

2.4. La M&S basée DEVS

2.4.1. Introduction

DEVS, comme nous avons vu, est formalisme hiérarchique. Il est bien adéquat aux systèmes complexes, du fait, qu'il propose une représentation de ces derniers sous forme de sous-modèles dont le couplage est typiquement systématique. D'autre part, DEVS distingue explicitement les changements de l'état du système issus de l'intérieur de ceux provoqués par des événements de l'extérieur. Cela donne une souplesse quant à la modélisation des SED. En outre, le formalisme DEVS ne se restreint pas à des formalités mathématiques abstraites. En effet, des correspondances ont été faites entre la structure hiérarchique de DEVS et des simulateurs abstraits pour l'implémentation des fonctionnalités des modèles DEVS.

2.4.2. Simulateur de Modèles couplés DEVS

Parallèlement à l'élaboration des différents modèles DEVS présentés précédemment, B.P. Zeigler a développé le concept de simulateur abstrait [ZEIGLER, 2000]. L'architecture de simulation est inspirée de la structure hiérarchique des modèles couplés DEVS. Un simulateur abstrait représente une description algorithmique permettant d'implémenter les fonctions du modèle (voir algorithme 2.3), afin de le faire évoluer dans le temps. L'idée fondamentale est faire correspondre à chaque élément du modèle un composant du simulateur. La construction d'un simulateur indépendant du modèle permet une séparation, au niveau réalisation, des parties modélisation et simulation.

Pour effectuer une simulation, une hiérarchie de processeurs, équivalente à la hiérarchie des modèles, est construite. A chaque composant du modèle (Figure 2.14) est associé un processeur de la structure hiérarchique du simulateur. Chaque processeur participe à la simulation en exécutant les fonctions qui expriment la dynamique du modèle. Les processeurs sont :

- le Simulateur qui assure la simulation des modèles atomiques en utilisant les fonctions définies par DEVS (Algo 2.3),
- le Coordinateur qui assure le routage des messages entre les modèles couplés en fonction des définitions de couplage (Algo 2.4),
- le Coordinateur Racine « Root » correspondant au modèle couplé global. Il assure la gestion globale de la simulation. Il ordonne le début et la fin de la simulation et gère l'horloge globale (Algo 2.5). Au début de la simulation, il envoie à son fils un message d'initialisation. Tant que la simulation n'est pas terminée, le « Root » envoie à son fils direct un *-message (*, t)

Le déroulement de la simulation s'effectue grâce à l'échange de messages spécifiques [Zeigler, 2000] entre les différents processeurs comme décrit dans la Figure 2.14. Les messages peuvent être de l'un des types suivants :

- Xmessage : utilisé lorsque le modèle aperçoit un événement externe,

- *message : utilisé pour simuler un événement interne,
- Ymessage : utilisé pour simuler un événement de sortie,
- Imessage : utilisé pour initialiser le modèle.

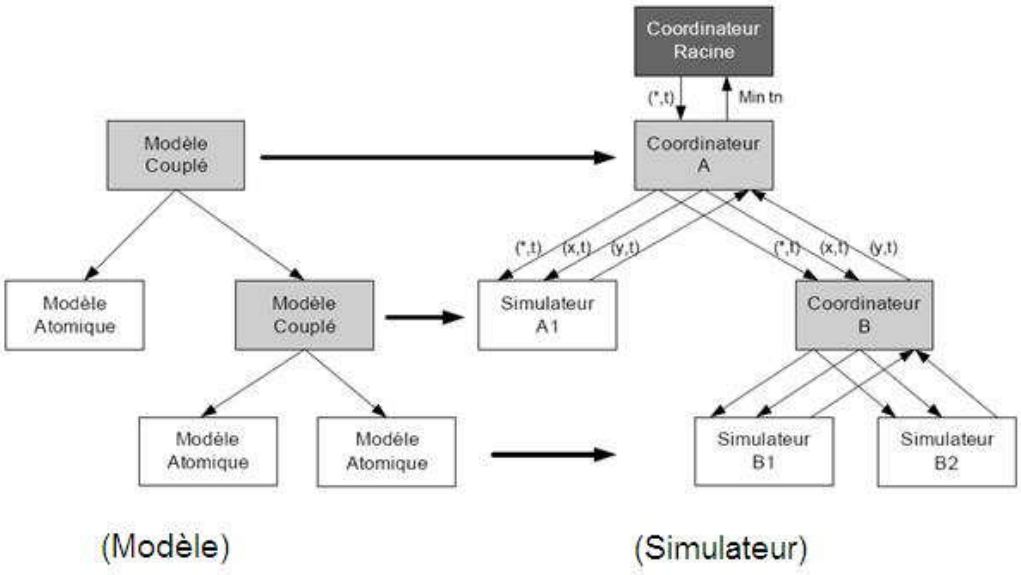

Fig. 2.14 Correspondance entre modèle DEVS et son simulateur.

Algorithme 3 : Algorithme du simulateur DEVS.

Variables :

parent // coordinateur parent

tl// temps du dernier événement

tn// temps du prochain événement interne

DEVS =< X,Y,S,δint , δext ,l, ta >// modèle associé

y // sortie courante du modèle

Réception i-message (i,t) au temps t :

tl = t −e

tn = tl +ta(s)

Réception *-message (*,t) au temps t :

si (t ≠ tn) alors

Erreur : mauvaise synchronisation

y= l(s) //stockage de la sortie avant changement d'état

envoie y-message (y,t) au parent coordinateur

s = δint(s)

tl= t

tn= tl+ta(s)

Réception x-message (x,t) au temps t avec x en entrée :

si !(tl ≤ t ≤ tn) alors

Erreur : mauvaise synchronisation

e = t −tl

s = δext(s,e,x)

tl= t

tn= tl+ta(s)

Algorithme 4 : Algorithme du coordinateur DEVS.

Variables :

parent // *coordinateur parent*

tl// temps du dernier événement

tn//temps du prochain événement interne

MC =< X,Y,D,{Mi},{Ii},{Zi, j} >

listeevent //liste des événements (*d, tnd*) *(triée par tn croissant)*

d //fils imminent sélectionné*

Réception i-message (*i, t*) **au temps t :**

pour chaque modèle d dans D faire :

envoie un i-message (*i, t*) au fils d

tl= max{tld|d 2 D}

tn= min{tnd|d 2 D}

Réception *-message (**, t*) **au temps t :**

si (*t ≠ tn*) alors :

Erreur : mauvaise synchronisation

d = premier*(*listeevent*)

envoie *-message (**, t*) à *d**

tl= t

tn= min{tnd|d 2 D}

Réception x-message (*x, t*) **au temps t avec x en entrée :**

si !(*tl ≤ t ≤ tn*) alors :

Erreur : mauvaise synchronisation

//*consultation du couplage externe pour obtenir les fils*

influencés

receveur = {r |r 2 D, MC 2 Ir, ZMC,r(x) ≠ 0}

pour chaque r dans *receveur* :

envoie x-message (*xr, t*) avec *xr= ZMC,r*(*x*) à r

tl= t

tn= min{tnd|d 2 D}

Réception y-message (*yd*, t*) **au temps t de la part de** *d** **:**

si *d* ϵ IMC* et *Zd*,MC*(*yd**) ≠ 0 alors :

envoie y-message (*yMC, t*) avec *yMC= Zd*,MC*(*yd**) au parent

receveur = {r |r ϵ D, d ϵ Ir, Zd*,r*(*yd**)≠ 0*}*

pour chaque r dans *receveur* :

envoie x-message (*xr, t*) avec *xr= Zd*,r*(*yd**) à r

Algorithme 5Algorithme du coordonateur*"Root"*.

Variables :

t//*temps de simulation*

fils //*subordonné directe*

tn// temps du prochain événement du fils

envoie un i-message (i,t) au fils

boucle jusqu'à la fin de la simulation :

envoie un *-message(*,t) au fils

t= tn

2.4.3. Motivations du choix du formalisme DEVS

Les raisons essentielles pour lesquelles nous nous sommes intéressés au formalisme DEVS, comme étant un formalisme pivot dans nos travaux de transformation entre formalismes (voir chapitre 5), sont :

- DEVS permet la modélisation d'un système sur plusieurs niveaux de description [Zeigler, 2000].
- Il permet la séparation entre la modélisation et la simulation d'un système.
- Il permet la simulation des systèmes automatiquement à partir de modèles prédéfinis [Zeigler 2000]. Ceci rend possible la notion de la réutilisation (Reuse) de modèles préalablement stockés dans des bibliothèques [Redjimi et Boukelkoul, 2013], ces modèles étant considérés comme des boites noires.
- Il permet de simuler des modèles continus [Bolduc et Vangheluwe, 2003].
- Il rend facile la modification du comportement des modèles du moment que les fonctions de transition d'états sont clairement définies et séparées.

- Il permet la modélisation des systèmes à structure dynamique (DSDEVS) [Barros, 1996,1997 ; Baati, 2007].
- Il rend facile le couplage des modèles hétérogènes grâce à la notion de transformation entre modèles[Boukelkoul et Redjimi, 2013].
- DEVS est un formalisme fermé sous couplage (Closed under Coupling). Les modèles couplés sont, en leur fonctionnement, identiques aux modèles atomiques, comme illustré dans la figure 2.15. Ainsi les fonctions de décomposition des systèmes complexes et leur reconstitution à partir des composants sera plus au moins facile et systématique.

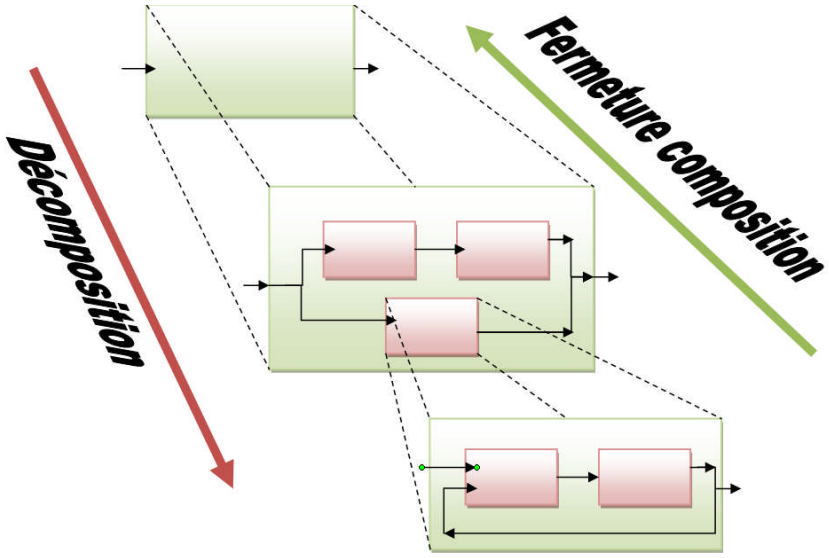

Fig .2.15. La composition et décompositiondans la hiérarchie des modèles DEVS.

2.4.4. La simulation multi-facette

La modélisation multi-facettes est une méthodologie de modélisation qui a été introduite par B.P. Zeigler [Zeigler, 1984b]. Elle est basée sur lesconcepts de la théorie des systèmes.Elle se concentre sur l'organisation de modèlesde base pour un domaine. Par domaine, on entend l'ensemble dessystèmes qui partagent des caractéristiques ou des attributs, des comportements et des modes de représentations. Dans cette variante demodélisation, il y a lieu de présenter les notions définies par Zeigler et qui sont illustrées figure 2.16.

- Entité du monde réel : où l'objet, peut montrer un comportement changeant fortementselon le contexte dans lequel il est étudié ou selon les aspects de son comportement qui sontétudiés.
- Modèles de base : représentation hypothétique, abstraite des propriétés de l'objet. En particulier,son comportement qui est valide quel que soit le contexte et qui décrit toutes lesfacettes de l'objet.
- Système : objet bien défini du monde réel, observé sous certaines conditions en considérantseulement certains aspects de son comportement.

- Cadre expérimental : décrit un ensemble limité de circonstances sous lesquelles un système(réel ou modèle) doit être observé ou être soumis à expérimentation. Le cadre expérimental décrit donc les objectifs de celui qui réalise l'expérience sur le système réel, ou, par la simulation, sur un modèle.
- Modèle décomposable : représentation abstraite d'un système dans le contexte d'un cadre expérimental. Ces modèles reflètent généralement certaines propriétés de la structure et/ou du comportement du système (avec un certain degré de précision).
- Expérimentation : action de réaliser une expérience. Une expérience peut interférer ou non avec les opérations du système (influencer ses entrées et ses paramètres). En tant que tel, l'environnement d'expérimentation peut être vu comme un système en soi (qui peut lui même être modélisé en modèle décomposable). L'expérimentation induit aussi l'observation et les méthodes de mesure des résultats.
- Simulation : simuler un modèle décomposable dans un certain formalisme (tels que les réseaux de Petri[Diaz, 2001], les équations différentielles [Hubbard, 1999] ou les Bond Graphs [Dauphin-Tanguy, 2003]) est le fait de calculer ses dynamiques d'entrées/sorties. La simulation qui permet de "mimer" une expérience réelle qui peut être vue comme une expérimentation virtuelle. Lorsque le but de la modélisation est de faire la description d'un système de manière compréhensible et réutilisable, le but de la simulation est d'être rapide et précise. Un point crucial dans la relation système-expérience/modèle-expérimentation virtuelle, est qu'il existe une relation d'homomorphisme entre le système et le modèle : fabriquer un modèle à partir d'un système réel et simuler son comportement devrait alors mener aux mêmes résultats que de faire une expérimentation réelle puis d'observer et codifier les résultats expérimentaux.
- Vérification : Processus d'inspection de l'intégrité de résultats d'un programme de simulation en respect du modèle décomposable dont il est dérivé.
  Validation : Processus de comparaison entre les mesures et les résultats de simulation dans le contexte d'un cadre d'expérimentation précis. Quand la comparaison montre des différences, le modèle formel qui a été décrit peut ne pas correspondre au système étudié. Néanmoins, un grand nombre de comparaisons avec des résultats de simulation ne valide pas le modèle. Il est important de noter que la correspondance entre le comportement du modèle et celui du système n'est valable que dans le contexte du cadre d'expérimentation. Par voie de conséquence, lorsque des modèles sont utilisés pour échanger des informations, un modèle doit toujours être regardé dans son contexte. De même, un modèle ne doit pas être développésans que l'on s'intéresse simultanément à son cadre expérimental.

Le point central de la méthodologie de modélisation multi-facettes est le mode de représentation.

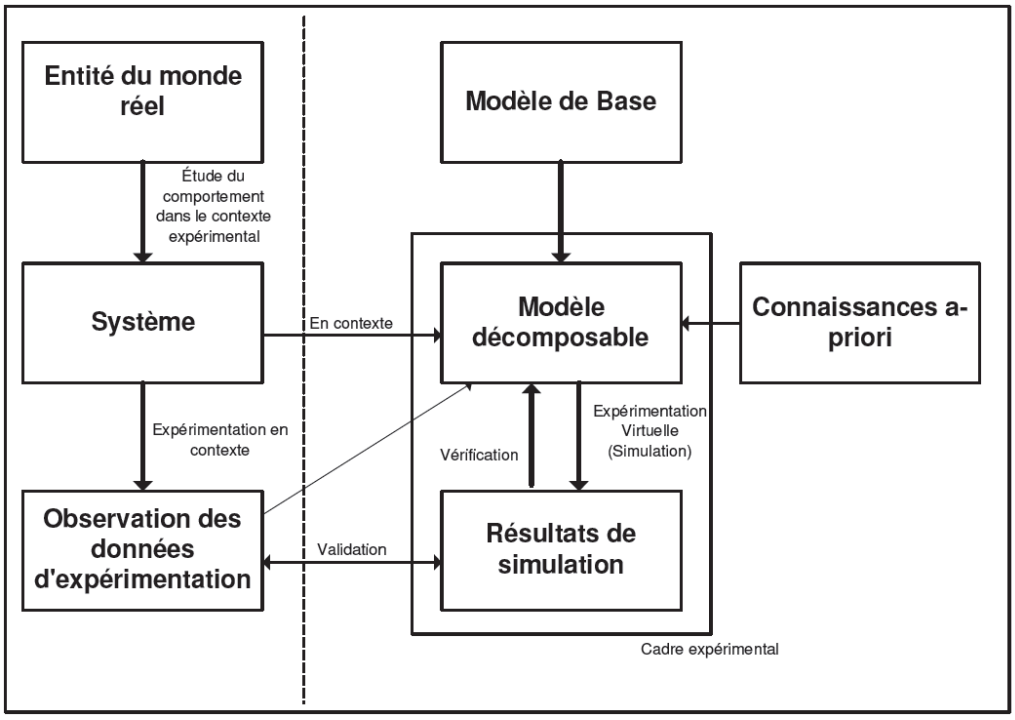

Fig.2.16 Concepts de modélisation et simulation [Vangheluwe, 2000]

2.4.5. Modélisation modulaire et hiérarchique basée DEVS

La structure Entité Système [Zeigler, 1984],reconnaît le besoin de représenter :

- **La décomposition** : comment un système peut être décomposé en composants du système.
- **Le couplage** : comment ces composants peuvent recomposer le système original.
- **La taxonomie** : les variantes admissibles d'un composant et ses spécialisations.
- Ziegler a aussi proposé la modélisation modulaire et hiérarchique comme un moyen d'exprimerla décomposition, le couplage et la taxonomie. La modélisation modulaire et hiérarchique estune approche de la dynamique des systèmes complexes où des blocs de constructions modulaires(des composants systèmes aux interfaces bien définies) sont couplés pour reformer un systèmecomplexe. Ces blocs de construction modulaires sont définis en spécifiant leurs interfaces d'entrées et de sorties sous la forme de ports au travers desquels s'effectuent toutes les interactions.Les modèles atomiques et couplés seront distingués en modélisation des systèmes. Alors que la structure interne d'un modèle atomique est spécifiée en termes d'états et de transitions d'états, la structure interne d'un modèle couplé est spécifiée par les composants qu'il contient et par leurs interconnexions. Cette modularité forme les bases pour des composants réutilisables car des modèles partageant les mêmes interfaces sont interchangeables. Les concepts de modélisation modulaire et hiérarchique ont été appliqués dans de nombreux domaines, notamment en modélisation de circuits électroniques [Kofman et al., 2000], [Capocchi et al., 2003] et en modélisation physique [Filippi et al., 2001].

2.4.6. DEVS en tant que formalisme unificateur

Bien que des efforts ont pris le défi dans l'objectif de normalisation des outils de modélisation. La plupart des chercheurs affirment que DEVS est le formalisme le plus approprié pour cette mission. La force de DEVS se résume dans sa capacité d'expression grâce au concept d'abstraction appliquée à chaque niveau, en allant des modèles atomique vers une collaboration de toute une société de modèles où les uns interagissent avec les autres. En plus, DEVS est indépendant de l'implémentation. Il offre une vision modulaire et hiérarchique des modèles dynamiques. Les événements générés par un modèle peuvent prendre des valeurs dans divers domaines et peuvent être des stimuli pour d'autres modèles. Ainsi, d'après Zeigler [Zeigler,2000], on peut prouver qu'il existe un modèle DEVS pour tout système à évènements discrets. Mais on peut aller plus loin; en effet, DEVS peut être utilisé comme formalisme « universel » [Touraille et al, 2010], permettant le couplage de modèles décrits dans des formalismes et selon des paradigmes hétérogènes. La principale idée est que les modèles sont considérés comme des boites noires qui n'ont aucun lien avec l'environnement extérieur que via des ports d'entrée et des ports de sorties qui s'échangent des évènements et des valeurs. Avec cette caractéristique d'abstraction, plusieurs modèles peuvent être couplés tout en bénéficiant de la réutilisation des modèles déjà existants.

Pour les modèles basés sur DEVS, on a montré que le couplage est une tâche typique. Cependant les modèles non-DEVS nécessitent un effort supplémentaire pour qu'ils soient couplés. Deux méthodes existent pour intégrer un modèle non-DEVS dans le cadre DEVS : la co-simulation et la transformation [Schmidt, 2006]. La transformation des modèles non-DEVS vers le formalisme DEVS vient dans le cadre d'avoir une spécification des modèles dans un langage uniforme. Nous nous intéressons dans nos travaux, à cette technique. Et nous allons la détailler d'avantage dans le chapitre qui suit.

Quant à la co-simulation, ce sont les communications entre les simulateurs qui sont standardisées et non pas les spécifications des modèles. HLA (High Level Architecture) [IEEE, 2000], développé par le département de défense américain, s'introduit comme solution au problème d'interopérabilité en s'appuyant sur le formalisme DEVS. HLA fait coopérer plusieurs simulateurs, géographiquement distants, ce qui nécessite des outils performants pour pallier au problème de la lenteur. Une approche plus souple, utilisant une architecture orientée service et plus précisément des services web, peut être trouvée dans [Mittal et al, 2007].

Le projet VLE (Virtual Laboratory Environment) s'inscrit lui aussi dans cette optique de multi-modélisation [Quesnel, 2010], en permettant à l'utilisateur de spécifier ses modèles selon différentes variantes de DEVS.

2.5. Conclusion

Dans ce chapitre nous avons défini les SED et les contraintes liées à leur modélisation et simulation en matière du caractère stochastique et de synchronisation des événements. Ainsi que les formalismes souvent utilisés dans la spécification de cette catégorie de systèmes. Nous avons détaillé d'avantage le formalisme DEVS et les RdP étant donné que notre contribution est basée sur ces deux formalismes (chapitre 4).

Dans la modélisation des SED, seules les variables les plus pertinentes dans le système doivent être retenues, et cela, suivant les objectifs de l'étude. La séparation entre les comportements internes et externes est aussi, une tâche qui ne doit pas être omise lors de l'identification des limites et frontières du système étudié. Par ailleurs, DEVS implémente différentes fonctions de transition qui assurent le bon déroulement de cette étape. Une fonction externe qui s'occupe du changement de l'état du modèle suite à la perception des événements de l'extérieur, une fonction interne qui change l'état d'une façon autonome et une fonction de sortie qui sert à influencer l'environnement externe par l'envoi d'événements. Par le concept d'abstraction qu'offre DEVS, les systèmes seront décomposés en sous-systèmes, facilitant ainsi leur spécification. Ces sous-systèmes seront recomposés, grâce au concept de couplage, pour reconstituer le modèle général. Ainsi, et à chaque fois que l'on remonte à un niveau d'abstraction plus haut, on aura des modèles couplés qui sont spécifiés de la même façon que les modèles atomiques : Cette caractéristique de fermeture sous couplage constitue un des points plus forts du formalisme DEVS. Ainsi, DEVS peut être utilisé comme formalisme « universel » [Touraille et al, 2010], permettant le couplage de modèles décrits dans des formalismes et selon des paradigmes hétérogènes.

Chapitre 3

Environnements pour la modélisation à évènements discrets basés DEVS

3.1. Introduction :

Dans cette section, nous présentons un panorama des environnements basés-DEVS les plus connus. Dans l'objectif de faire une comparaison entre les différents outils dans une synthèse ci-après On trouvera dans [Wainer, 2003] une recherche plus approfondie sur le sujet.

3.2. Environnements de M&S basés DEVS

3.2.1. ADEVS

ADEVS pour "A Discrete EVent system Simulator" est un simulateur de modèles DEVS programmé en C++ et JAVA sous forme de classes à l'université d'Arizona par Jim Nutaro [Nutaro, 2012]. La force de ADEVS réside dans la conservation du comportement de la dynamique par la simulation basée événements. Il comprend des classes dites « Par Simulateurs » pour « parallel simulators » qui sont responsables de la simulation sous des environnements multi-cœurs. Ainsi ce simulateur bénéficie considérablement du parallélisme des processeurs multi-cœurs et du partage de mémoires. Cependant ADEVS montre une faiblesse remarquable quant à la simulation des modèles à structure variable.

3.2.2. DEVS/JAVA

DEVSJAVA est un environnement développé en Java à l'université de l'Arizona [Sarjoughian et Zeigler, 1998]. Il supporte la norme "High Level Architechture" (HLA) [Zeigler et al, 1999]. DEVSJAVA inclut l'extension de modifications de la structure des modèles. Bien qu'il soit basé sur le modèle orienté objet, JAVADEVS supporte aussi le paradigme « Agent ». Cet environnement est couplé à l'interface CDM (Collaborative DEVS

Modeler) qui offre aux modélisateurs distants de collaborer entre eux afin de composer des modèles modulaires et hiérarchiques par le biais de diagrammes structurés et partagés [Sarjoughian et Zeigler, 1999] et ce grâce au support de la nome HLA.

3.2.3. CD++

Il s'agit d'un ensemble de bibliothèques qui ont été développées en 2002 par G.Wainer [Wainer, 2002] à l'université de Carleton, au Canada et Buenos Aires, en Argentine. Au niveau de CD++, on peut générer des modèles DEVS et Cell-DEVS, sous forme de graphes d'états. Les versions récentes de CD++ intègrent des extensions de DEVS parallèle. Cette plateforme offre aussi la possibilité de modéliser les RDP, les automates cellulaires, ou les graphes d'états DEVS. Sur le volet graphique de CD++, une application dite DEVSView a été développée et intégrée afin représenter les états et les événements sous forme d'animations.

3.2.4. MOOSE

MOOSE (Multimodel Object Oriented Simulation Environment) [Fishwick, 1998] est un environnement de modélisation et de simulation développée à l'université de Floride. MOOSE est composé d'une interface graphique, d'une bibliothèque de modèles et d'un moteur de modélisation et de simulation. L'environnement MOOSE supporte l'intégration de plusieurs types de modèles tels que les modèles en diagrammes de blocs, les modèles à équation sous contraintes et les modèles à base de règles. Ces modèles peuvent être connectés de manière récursive et hiérarchique.

3.2.5. ATOM^3

AToM3 (A Tool for Multi-formalism and Meta-Modeling) [Vangheluwe et al, 2002] est un environnement multi-paradigmes sous Python, développé à l'université McGill Montréal, Canada, en collaboration avec l'université d'Autónoma de Madrid (UAM), Espagne. Atom3 se caractérise par la modélisation de différents types de formalismes (méta-modélisation). Avec cet environnement, les modèles peuvent être transformés ou traduits automatiquement en d'autres modèles qui sont décrits soit dans le même formalisme soit dans un autre (voir figure 3.1). La partie la plus importante au niveau de cet environnement est le « Kernel » qui est responsable de toutes les tâches liées au chargement, création, stockage et manipulation des modèles et ce via le processus de réécriture graphique des modèles

Cependant la traduction de modèle d'un formalisme vers un autre sans pertes d'informations n'est pas toujours possible. Pour pallier ce problème, on adopte l'approche de Vangheluwe dans [Vangheluwe et al, 2000] qui consiste en une transformation de tous les modèles sous forme DEVS tout en bénéficiant des avantages du formalisme DEVS en matière de couplage et d'unification. Ainsi les modèles générés peuvent être retransformés. Le simulateur DEVS attaché est PyDEVS (Python DEVS) [Vangheluwe et Bolduc, 2002].

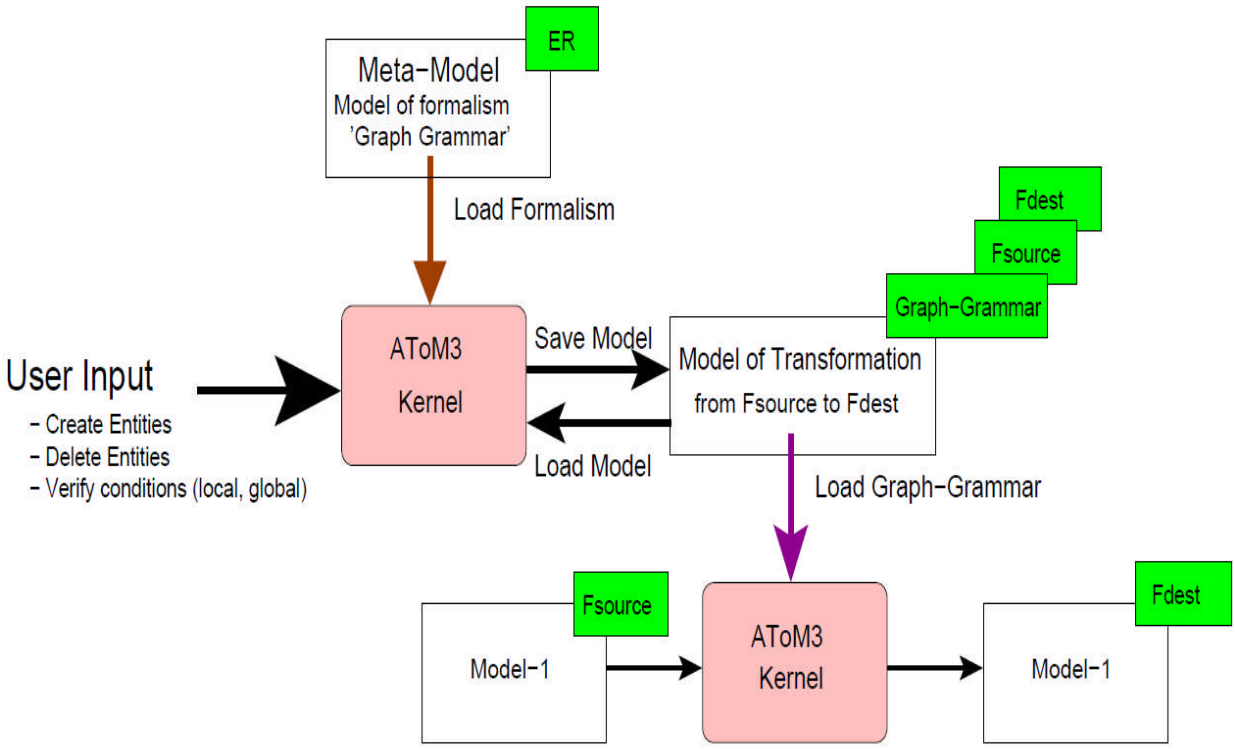

Fig 3.1 Transformation des modèles en AToM^3 [Vangheluwe et Bolduc, 2002]

3.2.6. CELL-DEVS

L'environnement Cell-DEVS pour (Cellular DEVS) se base sur le formalisme des automates cellulaires [Ilachinski, 2001]. Il est très utile pour la modélisation des systèmes où il y a lieu de représenter de l'espace. Le principe des automates cellulaires se résume dans une fonction de transition globale qui met à jour l'état de toutes les cellules en exécutant une fonction localement dans chacune des cellules, de manière synchrone et en tenant comptes des états des cellules voisines. La différence entre cet environnement et le paradigme des automates cellulaires est la définition explicite du comportement des cellules en une durée de vie de l'état des cellules.

3.2.7. SWARM

Swarm (Software for Agent-based Modeling Ressource) [SWARM, 2002] est un environnement constitué de multiples composants organisés sous forme de bibliothèques. Son architecture est très convenable pour l'implémentation de modèles basés sur les agents. Sa principale caractéristique est la réutilisation de modèles. En fait, les modèles réalisés dans SWARM vont être enregistrés pour une éventuelle réutilisation. Dans SWARM la simulation se fait à travers un ensemble d'interfaces graphiques prédéfinies ou réadaptées par les modélisateurs en langage C.TARSIER

3.2.8. ECLPSS

Eclpss (Ecological Component Library for Parallel Spatial Simulation) [Woodbury et al, 2002] est un environnement basé sur le langage Java. Il permet d'élaborer facilement des simulations d'écosystèmes dans de nombreuses échelles de temps et d'espace. Les modèles Eclpss sont composés de trois entités : un ensemble de variables, un ensemble de composants avec état modifiable et un ensemble de simulateurs pour ces composants. Les composants Eclpss sont développés en code java et stockés dans une bibliothèque pour réutilisation. L'intérêt principal de cet environnement est la possibilité d'évoluer des modèles sur des machines parallèles.

3.2.9. CORMAS

CORMAS pour (Common-pool Resources and Multi-Agent Systems) [Bousquet et al, 1998] est un environnement destiné à la programmation de modèles multi-agents. CORMAS est structuré en trois modules: un module pour la description des entités (agents), un module pour la gestion de la dynamique de l'environnement et un module qui observe la simulation. Ces modules sont composés des classes en SmallTalk [Clavel]. L'interface de développement de CORMAS est l'environnement de développement intégré de VisualWorks [Cincom, 2003].

3.2.10. Small DEVS

SmallDEVS est un environnement expérimental de simulation basé sur le formalisme DEVS. Il tient compte de l'aspect classe, aussi bien que la construction modèle orienté objet basé sur les prototypes. Ses composantes offrent la manipulation interactive avec des modèles et des simulations possibles. Cet outil supporte bien la modélisation et la simulation interactives. SmallDevs a été développé par Vladimir Janousek et ElodKironsky (université de Brno de Technlogie, République de Tchèque).

3.2.11. GALATEA

GALATEA (*GLIDER with Autonomus, Logic-based Agents, Temporal reasoning and Abducation*) est un environnement de modélisation et de simulation qui a été développé par Mayerlin Uzcategui, Jacinto Davila et Kay Tucci (université de Los Andes, Venezuela) [Davila et al, 2005]. Ilse présente comme une famille de langages destinés à modéliser les systèmes multi-agents pour les simuler dans des plateformes multi-agent basées DEVS [Bisgambiglia et al, 2012]. GALATEA est le produit de deux lignes de recherches: langages de simulation basés sur la théorie de Zeigler de simulation et d'agents logiques. Il y a dans GALATEA une proposition à intégrer, dans la même plateforme de simulation, les outils conceptuels et concrets pour plusieurs variantes de simulation telles que la simulation des SMA, la simulation distribuée, interactive, continue et discrète d'événement. Par ailleurs, GALATEA est le descendant direct de GLIDER, un langage de simulation basé DEVS qui a aussi intégré des outils pour la modélisation continue.

3.2.12. VLE

VLE (Virtual Laboratory Environment) [Duboz et al, 2003 ; Quesnel, 2006] est une plate-forme multi-agents basée sur le formalisme DEVS destinée à la modélisation des systèmes complexes développé au LIL (Laboratoire d'Informatique du Littoral). VLE est constitué d'un ensemble de modules regroupés sous forme de bibliothèques (figure 3.2). Chaque module est appelé pour une étape de cycle de vie du modèle. Les principaux avantages de ces bibliothèques sont la portabilité et l'efficacité tant au niveau rapidité de développement qu'en terme de temps d'exécution et ce grâce au choix du langage de programmation C++ où le modèles atomiques sont implémentés par des objets C++ stockés dans les bibliothèques dynamiques (.so, .dll, .dylib). Le graphe de connexion, les initialisations et les observations des modèles seront dans des fichiers XML d'extension VPZ.

VLE adopte le concept de paquet, avec lequel les modèles, les plans d'expériences et la documentation d'un projet de modélisation sont stockés dans des archives pour faciliter la diffusion et la réutilisation. En outre, certaines de ces bibliothèques sont extensibles via l'utilisation des plug-ins. Ainsi, les développeurs peuvent étendre la plate-forme sans avoir à développer directement dans les interfaces de VLE.

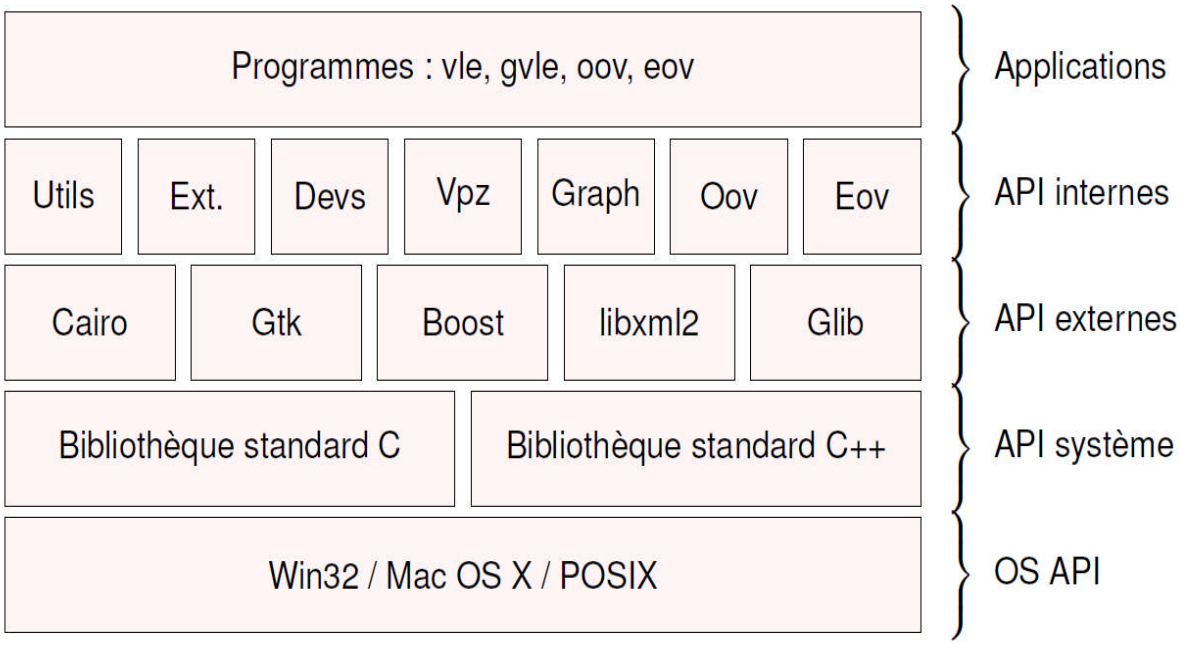

Fig 3.2 Architecture de VLE

Nous allons citer, dans ce qui suit, les bibliothèques les plus importantes se trouvant au niveau de VLE sont :

- Utils : une bibliothèque pour augmenter la portabilité de la plate-forme. Elle fournit des fonctions de chargements dynamiques de bibliothèques, de conversion de données, etc.

- Graph : cette bibliothèque permet de représenter la structure des modèles DEVS avec les modèles couplés, modèles atomiques, les ports et les connexions. Elle fournit également des fonctions utiles au noyau de simulation pour calculer le chemin des événements de manière efficace.

- Vpz : une représentation objet des données, structure de modèles, conditions expérimentales et observations, de VLE. Son utilisation principale est l'enregistrement et la lecture des plans d'expériences.

- devs : le noyau de simulation qui implémente le noyau de simulationPDEVS. Elle peut être étendue par des greffons pour développer des comportements de modèles atomiques.

- extension : l'ensemble des extensions que nous fournissons aux modélisateurs, équations différentielles, équations aux différences, systèmes à événements discrets, etc.

- oov, eov : les outils pour l'enregistrement de résultats en cours ou enfin de simulation dans des fichiers de sorties, dans des objets mémoire ou dans des interfaces graphiques.

3.2.13. JAMES2

JAMES2est un environnement de simulation des modèles de différents formalismes basé sur le concept des modules. Ces derniers sont séparés les uns des autres où chacun d'entre eux assure une fonctionnalité particulière dans le cadre de la modélisation et la simulation [Himmelspach et al, 2010] (voir figure 3.3). JAMES2 permet une séparation explicite du modèle et de son simulateur. Il montre aussi une souplesse d'extensibilité grâce à l'installation des plugins [Himmelspach and Uhrmacher, 2007]. Par ailleurs, la plus part de ses composants peuvent être changés, offrant ainsi, une architecture d'expérimentation pour la définition et l'exécution des algorithmes d'une manière relativement souple [Himmelspach et al, 2008].Au niveau de JAMES2, on peut choisir autant de simulateurs pour le même algorithme. Ce qui le rend un environnement dédié à la comparaison des algorithmes.

La figure 3.3 illustre l'ensemble des modules sur lesquels est basé JAMES2 :

- Le module « User Interface » : assure les fonctionnalités liées à l'utilisateur. Par ce module on peut créer, exécuter et contrôler la simulation des modèles.

- Le module « Instrumentation » : assure l'intégration des composants observateurs au modèle soumis à la simulation. Ainsi on pourra choisir quelle donnée à extraire pour l'analyse. Cette séparation des observateurs du modèle, assurée par JAMES2, offre une fonctionnalité importante quant à réutilisation du même modèle pour un autre cadre de simulation suite à des nouveaux objectifs.

- Le module « Experiment » : constitue l'ensemble des plans et des simulations faites dans le cadre des mêmes objectifs. Cet ensemble est enregistré pour des éventuelles analyses et validation des modèles.
- Le module « Model » : encapsule les formalismes supportés par JAMES 2. Au moment de la lecture du modèle, des classes Java vont être créées où les méthodes publique du modèle vont être réécrites de telle sorte que les simulateurs puissent interagir avec le modèle, puisque JAMES 2assure une séparation explicite du modèle de son simulateur.
- Le module « Datasink » : assure l'interfaçage de la simulation et la base de données. Avec des captures de valeurs de variables moment de la simulation, on peut faire des éventuelles analyses de données.
- Le module « Exp.sink » : joue presque le même rôle que le module « Data sink », mais là, les informations capturées concerne le plan d'expérience et l'ensemble des simulations faites dans le même cadre. Les données peuvent être stockées dans des différents formats (fichier XML, TXT, base relationnelle…etc.).
- Le module « Simulator » : comporte l'ensemble des fonctionnalités liées à la simulation telles que la gestion du temps, la surveillance de la simulation, le choix des simulateurs…etc. Ce module peut avoir des extensions afin d'assurer la distribution de la simulation et le parallélisme. Ce module assure aussi, la coexistence de simulations de plusieurs modèles basés sur différents formalismes

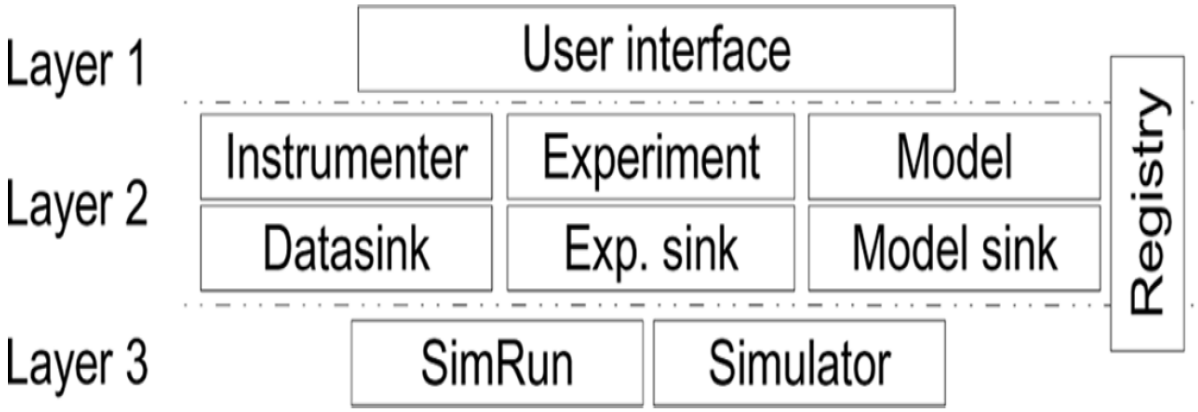

Fig 3.3 Modules de JAMES2 [Himmelspach et al, 2010]

3.2.14. DEVSimPy

DEVSimPy est un environnement de modélisation et de simulation sous DEVS [Capocchi et al, 2011]. Il est basé sur le moteur de simulation « Python DEVS » [Vangheluwe et Bolduc, 2002], ce qui le rend un environnement dédié à la M&S des systèmes complexes. DEVSimPy est basé sur des packages DEVS.py et Simulation.py. Le premier assure la spécification facile de l'architecture hiérarchique des modèles DEVS (voir la figure 3.4). Alors que Simulation.py constitue le moteur de simulation. DEVSimPy offre une interface graphique facile permettant ainsi, la création et la simulation des modèles DEVS en se basant sur la bibliothèque GUI et l'approche drag and drop.

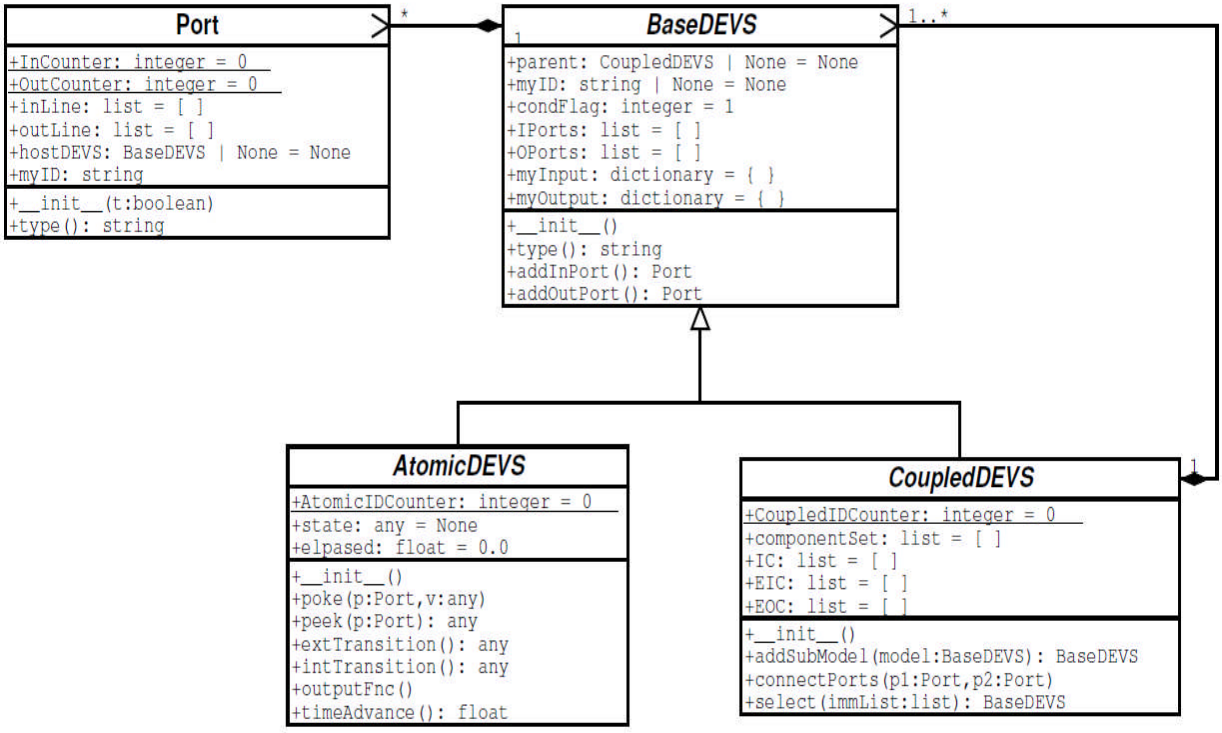

Fig 3.4 Architecture de modélisation sous Python DEVS [Vangheluwe et Bolduc, 2002]

3.3. Synthèse des différents outils

Dans cette section nous présentons une comparaison des différents environnements cités ci-dessus. Chaque environnement a un coté positif selon certains aspects néanmoins, il peut montrer un manque pour d'autres. La plateforme ADEVS, par exemple, montre une faiblesse quant à la modélisation de la structure dynamique, cependant des bibliothèques ont été rajoutées pour combler cette faille [Uhrmacher, 2001]. Pour la modélisation multi-agents, on aura recours aux environnements CORMAS, VLE, SWARM et JAMES [Schattenberg, et Uhrmacher, 2001] qui sont aussi basés sur le formalisme DEVS. En général, ce type de modèles n'est pas utilisé pour obtenir des résultats quantitatifs mais plutôt pour vérifier des hypothèses d'évolution ou de dynamique d'un système. Néanmoins il est difficile d'évaluer l'impact des simplifications faites lors de la modélisation, ce qui constitue un inconvénient par

rapport à des approches plus classiques sous forme d'automates cellulaires comme Cell-DEVS ou Eclpss. A noter que l'utilisation de normes et standards tels que les langages de programmation (Java pour DEVSJAVA, Eclpss ou C++ pour VLE) et les formalismes utilisés notamment DEVS ou encore des protocoles de simulation comme HLA, ne garantit pas l'interopérabilité des modèles au niveau sémantique. D'autre part, ces outils exigent un certain niveau de connaissance dans les langages de programmation pour les utilisateurs puissent les exploiter, ce qui limite la liberté des modélisateurs, à l'exception de VLE qui offre, relativement, un niveau plus haut d'abstraction pour la conception des modèles. JAMES 2, offre un cadre plus souple de modélisation grâce au concept de plugin. Il est très souhaitable pour la M&S des systèmes complexes, cependant il montre une faiblesse dans ses interfaces graphiques. Par contre DEVSimPy propose une interface graphique facile basée sur le concept drag and drop. Mais il est peu approprié aux variantes DEVS autre que DEVS classique. Pour l'environnement Atom3, on note la problématique de la définition de la grammaire qui constitue une contrainte majeure contre la standardisation. Même les auteurs de cet outil affirment que DEVS est le formalisme unificateur avec succès. Des travaux sont en cours de généralisation de DEVS soit DEVS Standardisation [DEVS-Group, 2003]. Il y a aussi le souci d'intégration des outils de traitements des données avec les outils de simulation. Tarsier propose une intégration parfaite avec la possibilité de stocker et d'analyser différentes couches de données et cela par rapport au niveau d'abstraction des modèles et des simulations. La plupart des approches logicielles utilisent l'architecture en modules. Ainsi DEVSJAVA/CDM, MOOSE, VLE, JAMES 2 ou encore Eclpss, CORMAS et Tarsier proposent une séparation explicite du moteur de simulation. Eclpss, Tarsier, CORMAS et dans une certaine mesure SWARM et VLE proposent aussi des modules de contrôle de simulation, permettant même d'interagir avec des modèles en cours de simulation. Ce qui rend facile la vérification des modèles. Plusieurs environnements proposent des bibliothèques de modèles. Ainsi MOOSE, SWARM, Eclpss, Atom3 et dans une certaine mesure DEVSJAVA, Cell-DEVS, CORMAS et VLE offrent la possibilité de stocker et récupérer des modèles déjà développés pour des raisons de réutilisabilité de modèles. Cependant seul Atom3 propose de manière claire une syntaxe pour le stockage.

3.4. Conclusion

Ce chapitre est une extension du chapitre précédent où on a parlé des formalismes permettant la modélisation des SED. En contre partie, nous avons été orientés, dans ce chapitre, vers les implémentations et les environnements. Ainsi, nous avons présenté un panorama de différents outils et plateforme permettant la modélisation et la simulation des systèmes à événements discrets. En suite nous nous sommes focalisé sur les environnements basés sur le formalisme DEVS. La conclusion que nous pourrons tiré à propos de ces environnements est que l'utilisateur est impérativement lié aux langages de programmation ; il doit montrer un certain niveau de connaissances pour pouvoir concevoir ses modèles. En plus, la plus part des ces environnement manque d'une licence qui permet formellement

l'utilisation, la modification et la redistribution de la source afin de rendre l'exploitation souple et évolutive. Cependant, d'autres environnements qui montrent une certaine abstraction (tels que VLE) et offrent une certaine liberté à l'utilisateur ne proposent pas malheureusement des packages qui intègrent tout le nécessaire de l'installation.

Ces dans ces circonstances que nous avons opté pour le développement d'un environnement basé DEVS (qui sera détaillé dans le chapitre 5). Cet environnement va être spécialement caractérisé par la facilité d'utilisation (la population visée n'est pas des informaticiens). Il sera modulaire de façon à faciliter l'intégration des formalismes au fur et à mesure. En plus, il sera open source, ce qui favorise la collaboration et l'amélioration du produit.

Chapitre 4

Les transformations entre modèles

4.1. Introduction :

Dans ce chapitre nous allons aborder la notion de transformation entre modèles. La question qui se pose est : « Pourquoi transformer un modèle en un autre ? Autrement dit, pour quoi ne pas modéliser les systèmes en utilisant un seul formalisme qui soit universel et se libérer ainsi de toute transformation ? ». La réponse elle-même pose une problématique : le formalisme universel, s'il existe, combien d'aspects va-t-il toucher ?

Il apparait, actuellement qu'aucun formalisme n'est capable, à lui tout seul, de gérer tous les aspects des systèmes. Surtout, quand il s'agit des systèmes complexes. La diversité des formalismes n'est en aucun cas un obstacle. On pourrait même considérer que c'est plutôt une richesse pour la spécification. Chaque formalisme est très adéquat pour la modélisation d'un domaine donné, soit une catégorie de représentation de systèmes. Il s'agit, donc, de déterminer avec soin les formalismes à appliquer dans tels ou tels cas.

Les plateformes de simulation sont souvent basées sur plusieurs formalismes déterminés. Et c'est dans ce contexte, que s'installe la notion de transformation entre modèles. Elle vient dans l'objectif de faciliter la tâche des modélisateurs en leur offrant la possibilité de réutiliser des bibliothèques de modèles hétérogènes en un seul modèle, sans avoir une connaissance approfondie des autres formalismes.

D'ailleurs, l'approche IDM (Ingénierie Dirigée par les Modèles) ou MDE (Model-Driven Engineering) utilise un ensemble de modèles comme des éléments d'entrée pour la modélisation de concepts spécifiques d'un système. Ces modèles sont formalisés avec une grande précision pour les rendre exploitables par des processus de transformation.

4.2. Définition de la notion de transformation entre modèles

Kleppe [Kleppe, 2003] donne la définition suivante pour la notion de transformation entre modèles :

"A Transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language"…

Cette définition est largement consensuelle. Elle décrit la transformation comme étant un processus de génération automatique d'un modèle cible à partir d'un modèle source selon un ensemble de règles et de définitions. Ces règles doivent être en mesure de décrire comment un modèle source peut être transformé en un modèle cible. Une règle de transformation est une description de la façon dont des constructions dans le langage source peuvent être transformées dans des constructions dans le langage cible. Pour réaliser la transformation, les modèles (source et cible) doivent être représentés dans un certain langage de modélisation (par exemple : UML pour des modèles de conception). Ce langage de modélisation étant lui-même défini par un méta-modèle.

Les règles de transformations sont basées, elles mêmes, sur les langages de transformations, tels que les standard QVT et ATL. En exécutant les règles de transformation, le processus produit en sortie le modèle cible du modèle source [Touil, 2007].

4.3. L'approche IDM

L'IDM est un regroupement detechniques de génie logiciel dans l'objectif de gérer la modélisation des systèmes complexes. Elle peut être considérée comme un domaine qui a émergé avec les technologies liées à l'instrumentation des modèles. Il existe différentes approches pour manipuler les modèles dans leur processus de développement des systèmes. L'approche la plus connue et peut-être la plus développée est l'approche MDA « Model Driven Architecture » ou «Architecture Dirigée par les Modèles ».

4.3.1. L'Architecture Dirigée par les Modèles (MDA)

MDA est une approche de développement proposée par l'OMG (*Object Management Group*) [OMG, 2004]. L'idée de base du MDA est de séparer les spécifications fonctionnelles d'un système des spécifications techniques de son implémentation sur une plateforme donnée. Autrement dit, cette approche permet de réaliser le même modèle sur plusieurs plates-formes grâce à des projections. L'approche MDA est entièrement basée sur les modèles et leurs transformations.

Le cycle de développement de l'approche MDA est vu sous la forme d'un Y [Roques, 2000](voir la figure 4.1). Les feuilles représentent les spécifications fonctionnelles du système

et les spécifications techniques de la plate-forme cible. L'implémentation est le résultat de l'intégration des deux spécifications.

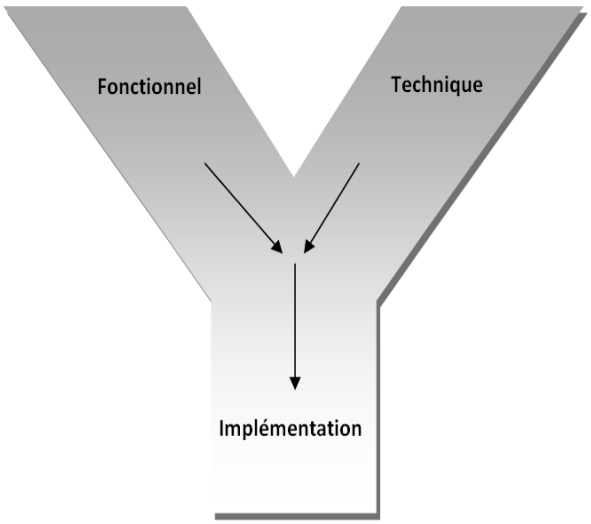

Fig.4.1 Cycle de développement en Y

4.3.2. Transformation des modèles dans MDA

Dans l'approche MDA, l'OMG a défini plusieurs modèles qui vont servir dans un premier temps de modèles de démarrage de la tâche de modélisation puis, par transformations successives, on va générer le code de l'application. Les quatre principaux types de modèles définis dans l'approche MDA sont [Bézivin, 2002]:

- CIM (Computation Independant Model): Appelé aussi modèle de domaine ou modèle métier. Avec CIM, les besoins sont formulés pour décrire la situation dans laquelle le système sera utilisé.
- PIM (Platform Independant Model): Le PIM décrit le système indépendamment de la plate-forme cible sur laquelle il s'exécutera. Il offre une description abstraite loin du détail technique.
- PDM (Platform Description Model): Le PDM est le modèle qui décrit une plate-formed'exécution.
- PSM (Platform Specific Model): Le PSM est le résultat de la combinaison du PIM et du PDM. Ilreprésente une vue technique détaillée du système. Il peut exister avec différents niveaux dedétails.

La figure 4.2 donne une vue générale des transformations possibles entre ces différentstypes de modèles.

Les transformations de raffinement de type PIM vers PIM ou PSM vers PSM visent à enrichir, filtrer ou spécialiser le modèle. Il s'agit de transformations de modèle à modèle [Czarnecki, 2006].

La transformation de PIM vers PSM permet de spécialiser le PIM en fonction de la plate-forme cible. Elle n'est effectuée qu'une fois le PIM suffisamment raffiné. Cette transformation de modèle à modèle est réalisée en s'appuyant sur les informations fournies par le PDM [Czarnecki, 2006].

La transformation de PSM vers le code (l'implémentation) est une transformation de type modèle à texte. Le code est parfois assimilé à un PSM exécutable. On aura souvent à rajouter des lignes de code manuellement [Czarnecki, 2006].

Les transformations inverses code vers PSM et PSM vers PIM jouent le rôle de rétro-ingénierie. Ce type de transformation est souvent utile pour la réutilisation des modules stockés dans le cadre de l'approche MDA.

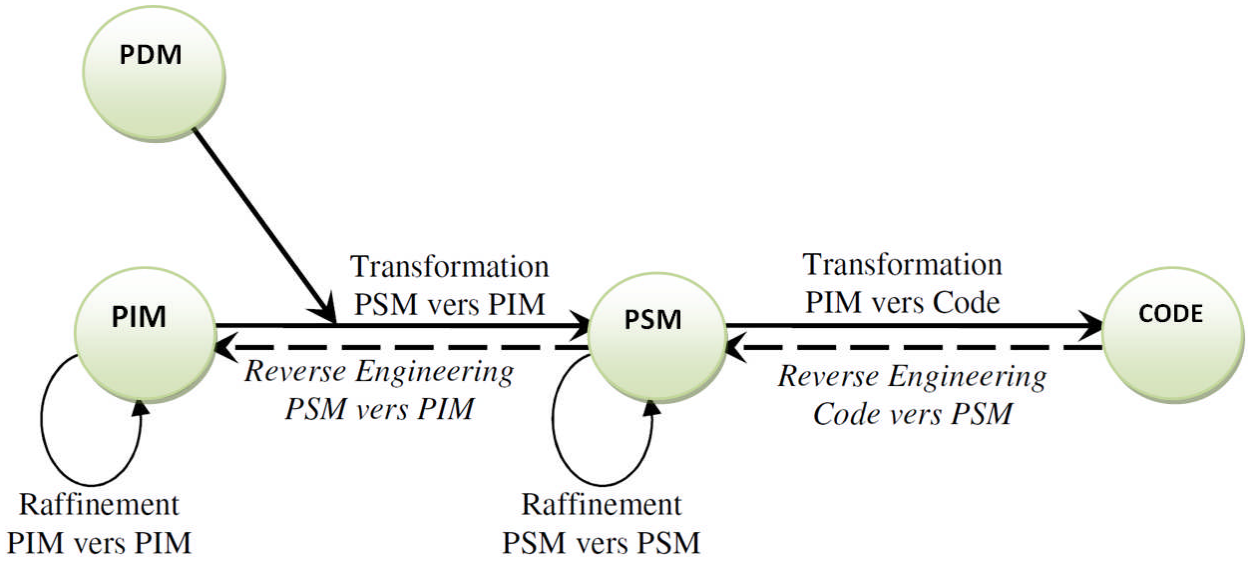

Fig.4.2. Les modèles et les transformations dans l'approche MDA

4.4. Transformations SMA vers DEVS

4.4.1. Les systèmes Multi-Agents (SMA)

La complexité des systèmes, ainsi que le besoin de la distribution du calcul (surtout dans la modélisation des systèmes naturels) ont motivé la communauté scientifique à adopter le paradigme « Agent » [Ferber, 1997]. Par ailleurs, l'approche multi-agent est à l'origine de la connexion de plusieurs domaines spécifiques de l'intelligence artificielle, des systèmes informatiques distribués et des technologies du génie logiciel. C'est une discipline qui se concentre sur les comportements collectifs produits par les interactions de plusieurs entités autonomes et flexibles appelées agents. Ces interactions sont axées autour de la coopération, la concurrence ou la cohabitation entre ces agents. Un système multi-agent est un système distribué composé d'un ensemble d'agents. Dans un SMA, un agent est une entité atomique

[Chaib-Draa, 1999]. L'agent peut être réactif, en ce sens qu'il dispose de capacités d'actions limitées (de type réponse à un stimulus), cognitifs ; c'est-à-dire qu'il peut élaborer des plans d'actions, prendre des décisions et autres taches 'intelligentes' ou hybride dans le cas où l'agent est à la fois réactif et cognitif.

Un système multi-agent présente, généralement, les caractéristiques suivantes :

- Il n'y a aucun contrôle global du système multi-agents.
- Les données sont décentralisées.
- Le calcul est asynchrone.

D'autre part, J.Ferber [Ferber, 1997] représente un système multi-agent par le couple <A, W> où A est un agent et W un environnement. La figure 4.32 représente ce couple.

$A = (Pa, Percept_a, F_a, Infl_a, S_a)$

Avec :

- P_a : représente la fonction de perception de l'agent.
- $Percept_a$: l'ensemble des stimuli et sensations qu'un agent peut recevoir.
- F_a : la fonction de comportement de l'agent.
- $Infl_a$: la fonction d'action de l'agent.
- S_a : l'ensemble des états internes de l'agent.

$W = (E, \Gamma, \Sigma, R)$

Avec :

- E l'espace dans lequel l'agent évolue.
- Γ l'espace des influences produites par l'agent.
- Σ état de l'environnement.
- R la loi d'évolution.

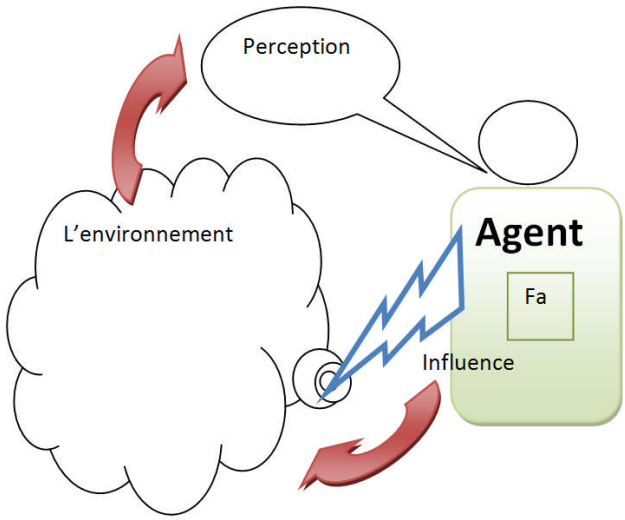

Fig.4.3 Principe d'action-réaction d'agent.

4.4.2. Transformation SMA vers DEVS

La ressemblance entre l'agent et le formalisme DEVS est assez importante. Chose qui rend la transformation relativement facile surtout quand il s'agit des agents réactifs. Donc la transformation sera à un niveau d'abstraction plus au moins bas. En effet la transformation model-to-model est plus adéquate dans ce cas. Par ailleurs, DEVS a été utilisé les premières fois pour la spécificationdes SMA durant les années 1990 [Duboz, 2004]. Les travaux d'Uhrmacher et Arnold en 1994 furent la première expérience [Uhrmacher et Arnold, 1994] où on décrivait le comportement autonome d'un agent comme étant des transitions externes formalisant, ainsi, la perception. De même que pour l'influence sur l'environnement qui a été formalisée par la fonction de sortie dans DEVS. Ensuite, et en s'inspirant de ce travail, d'autres travaux ont été menés dans ce sens, tels quel'intégration des SMA réactifs en DEVSproposée dans la thèse de Duboz [Duboz, 2004]. Akplogan[Akplogan et al, 2009]propose un modèles DEVS de la décision pour simuler le comportement d'agents intelligents dans le domaine de la culture. Dans ce travail, on s'intéresse beaucoup plus aux comportements des agents dans un environnement à événements discrets (voir la figure 4.4).

D'autres travaux portent sur le mapping des agents en DEVS, dans l'objectif de simuler des modèles SMA dans des environnements basés DEVS, tels que dans [Joannon, 1988] où l'on parle d'une Application à la conduite de systèmes de culture basée sur la modélisation et la simulation d'agents en DEVS.

Au niveau de l'agent ; la fonction de perception peut être transformée en fonction de transition externe de DEVS (δ_{ext},). Quant à la fonction d'influence, elle correspondra à la fonction de génération des événements en sortie des modèles DEVS (λ). Pour les messages, ça ne va pas poser de problèmes du moment que les modèles DEVS s'échangent, eux aussi, des messages sous formes d'événements.

Cependant la liberté de l'agent de communiquer autant d'éléments ne sera pas transformée avec perfection puisque les modèles DEVS communiquent à travers des interconnexions via des ports d'entrée et de sortie [Zeigler, 1984, 2000].

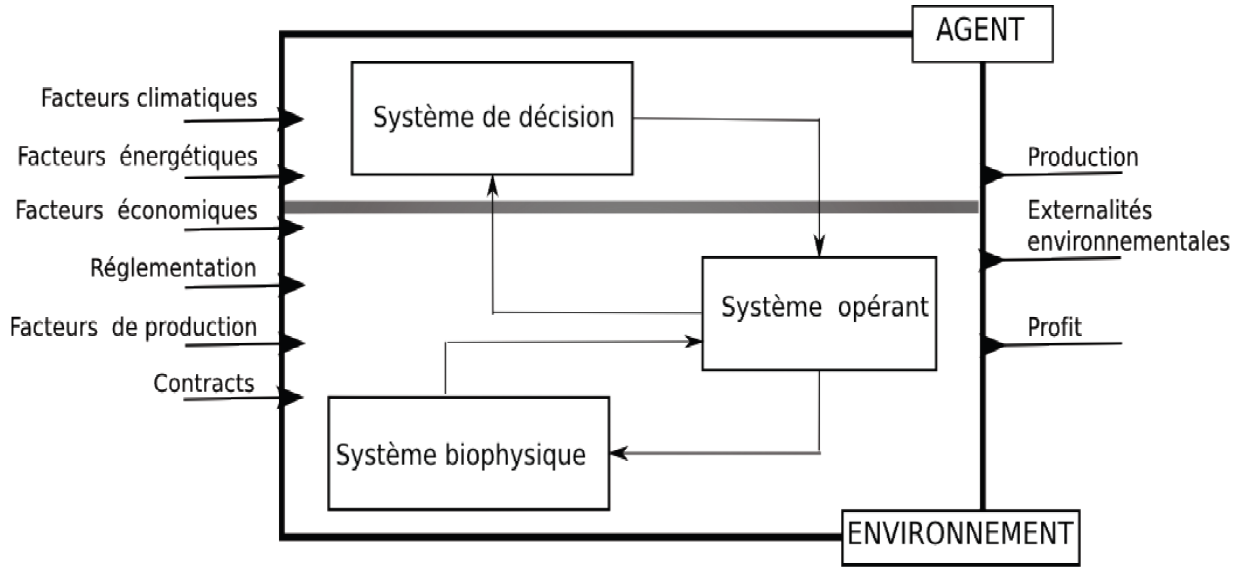

Fig.4.4 Modèle DEVS d'agent intelligent [Akplogan et al, 2009]

4.5. Transformation des RdP en DEVS

Dans ce volet on peut citer les travaux de Jacques et Wainer qui ont utilisé l'outil CD++ pour développer les RdP [Jacques et Wainer, 2002] et la modélisation à base de multi-paradigmes pour générer les RdP et les State-Chartes [Lara et Vangheluwe, 2002]. La figure 4.5 représente le modèle conceptuel correspondant à une place dans un RdP [Jacques et Wainer, 2002].

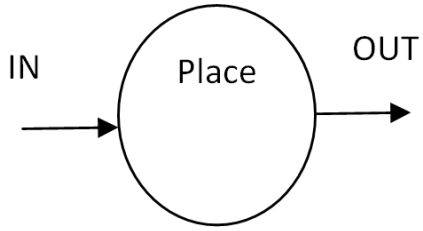

Fig 4.5. Modèle conceptuel d'une place de PN

En fait, le mapping entre les RdP et DEVS est naturel du moment que les deux formalismes modélisent les systèmes à événement discrets. D'après Zeigler[Zeigler, 2000] tout modèle à événements discrets dans n'importe quel formalisme a impérativement une image dans le formalisme DEVS. Sachent que les RdP comptent parmi les formalismes qui modélisent les systèmes dynamiques en toute perfection.

L'inconvénient de cette transformation est qu'elle n'offre pas un automatisme de génération des modèles DEVS qui corresponde à une variété de transitions. Le modélisateur, se trouve, ainsi, limité dans ses choix.La figure 4.6 illustre comment dans [Jacques et Wainer, 2002] la transition d'un RDP a été modélisée en un modèle DEVS atomique.Nous montrons aussi comment les fonctions de transitionsont été implémentées dans l'aperçu du pseudo-code illustré figure 4.7.

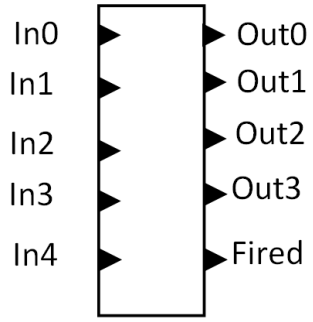

Fig 4.6. Modèle DEVS correspondant à une transition de PN.

```
δext (s,e,x) {
retrieve id and number of tokens from message
case id = 0    /* generic message */
increment tokens
    hold in active 0  /* to advertise the number of
tokens */

case id != 0   /* specific message */
    id matches id of this place?
         no:   disregard the message
         yes:  decrement tokens by the number of
tokens specified if there are enough. Otherwise
throw an exception.
       hold in active 0 /* to advertise the number
of tokens */
}end of external transition function


δint (s)  {
passivate      /* wait for the next external event
*/
}


λ(s) {
combine id and tokens state variables in one mes-
sage and send on the out port.
}
```

Fig .4.7 Implémentation des fonctions des fonctions de transition du modèle DEVS correspondant à une transition d'un RdP [Jacques et Wainer, 2002]

4.6. Contribution à la transformation des RdP en DEVS

4.6.1. Introduction :

Après avoir présenté, les différents cas de transformation entre formalismes. Nous allons, maintenant, détailler notre approche de transformation des RdP vers le formalisme DEVS [Redjimi et Boukelkoul, 2013]. Notre contribution rentre dans le cadre de la multi-modélisation à base de formalismes hétérogènes. Par ailleurs, la modélisation et la simulation des systèmes complexes nécessitent leurs représentations par des sous-modèles plus simples. La recomposition de ces derniers avec la prise en considération de leurs interconnexions et interactions, nécessite la mise en œuvre des outils et des mécanismes de couplage. C'est dans cette optique que notre contribution se focalise. Dans ce qui suit, nous allons illustrer les étapes de transformation proposée.

4.6.2. Description de la méthode

Il s'agit de transformer les RdP existants sous format de fichiers XML en modèle DEVS. Chaque RdP correspondra à un modèle DEVS couplé contenant des modèles DEVS atomiques autant de places et de transitions du RdP. Le modèle couplé est composé, non seulement, des modèles atomiques, mais aussi des modèles couplés considérés comme des boites noires, qui sont préalablement stockés pour des fins de réutilisation. Ainsi ces modèles peuvent être simulés dans les environnements DEVS tels que VLE, JAVADEVS, ADEVS…etc. Plusieurs travaux se sont intéressés à cette question de transformation. Citons l'utilisation de CD++ pour développer les RdP [Jacques et Wainer, 2002] et la modélisation à base de multi-paradigmes pour générer les RdP et les State-Chartes [Lara et Vangheluwe, 2002]. Ces approches se rapprochent le plus de notre travail. Cependant elles proposent des outils limités de génération des PN. Le développeur choisit parmi les modèles stockés ceux qui conviennent aux places et transitions de son RdP. Ainsi, il reconstitue le graphe manuellement (pour les systèmes complexe, c'est une tâche extrêmement fastidieuse). On peut se trouvé même incapable de représenter des places ou des transitions dont le nombre de jetons mis en jeu est important. En revanche, notre approche [Boukelkoul et Redjimi, 2013] se distingue par la mise en place d'algorithmes capables de transformer de façon automatique les RdP existants que soient-ils complexes ou non. Nous proposons dans le chapitre suivant une implémentation d'un environnement de modélisation et de simulation (programmé en java) qui offre la possibilité de construire ses modèles DEVS manuellement, les importer depuis des fichiers XML ou les générer automatiquement à partir des RdP, après -bien sûr- une transformation adéquate. La figure 4.8 illustre le cycle de transformation d'un RdP vers un modèle DEVS.

4.6.3. La transformation des modèles

L'idée de notre approche est d'avoir comme résultat un modèle DEVS couplé (CDEVS dans la suite) fidèle au RdP en entrée (soit déjà existant). Pour cela, nous proposons un algorithme qui génère automatiquement CDEVS en spécifiant ses composants et leurs

interconnexions. Par défaut le modèle CDEVS contient deux ports d'entrée : un pour l'initialisation des modèles correspondant aux places (On l'appelle InitP), et l'autre pour contrôler la simulation. Il est connecté à tous les modèles correspondant aux transitions (On l'appelle InitT). Ainsi, en envoyant des événements via ce port, toutes les transitions vont l'apercevoir. Et de cette façon que la simulation sera contrôlée, du moment que les transitions représentent l'aspect dynamique des RdP. Par contre les places sont les éléments qui représentent l'aspect statique.

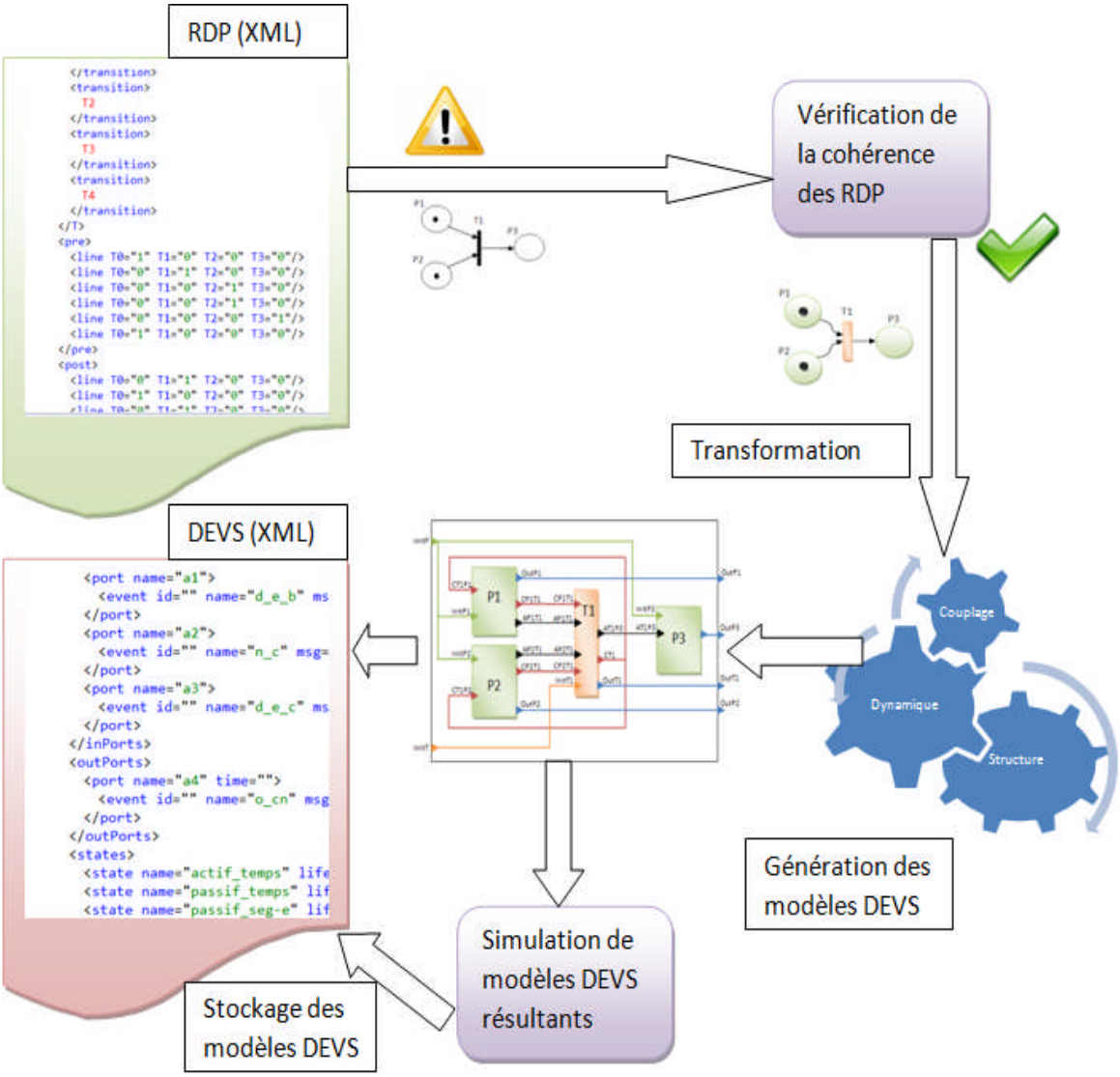

Fig .4.8 Cycle de transformation d'un RdP en un modèle DEVS

Formellement la transformation est présentée comme suit :

PN = (P, T, PRE, POST, Mo) → CDEVS=(X, Y, D, EIC, EOC, IC)

Où:

D = { P □ T }

X = { InitP, InitT }

Y = {OutDi / Di est le modèle atomique qui représente Pi ou Ti}

EIC = {

(CDEVS.InitP, PDEVS.IntPi) □

(CDEVS.initT ,TDEVS.IntTj)

Tel que i□ N+ & i< nombre de places, j Є N+ & j < nombre de transitions

}

EOC={

(Pi.OutPi, CM.OutPi) ,

(Tj.OutTj, CM.OutTj)

Tel que i Є N+ & I < nombre de places, et j Є N+ & j < nombre de transitions

}

IC = {

{(Pi.APiTj, Tj.APiTj) / PRE[i,j]>0 }

□ {(Tj.ATjPi, Pi.ATjPi) / POST[i,j]>0 }

□{ {Tj.CTj} X {Pi.CTjPi} / PRE[i,j]>0 }

□{ (Pi.CPiTj , Tj.CPiTj} / PRE[i,j]>0 }

Tel que i Є N+& I < nombre de places, et j Є N+ & j < nombre de transitions

}

4.6.3.1. Transformation des places d'un RdP

Chaque place P_i du modèle RdP va être transformée en un modèle atomique DEVS que nous appelons dans la suite « $PDEVS_i$ » Pour « Place-DEVS ». Ce modèle sera initialement vide. Il ne contient qu'une variable d'état M pour le marquage et deux ports ; un pour l'initialisation « $InitP_i$ » et un autre pour les sorties des résultats « $OutP_i$ », comme illustré dans la figure 4.9(a). A travers ces deux ports, le modèle PDEVS sera couplé avec le principal modèle CDEVS. En cours de la transformation, le modèle CDEVS pourra avoir d'autres ports qui vont représenter les arcs reliant la place à des transitions. L'algorithme 5.1 assure cette transformation que nous allons détailler d'avantage dans les sections qui suivent.

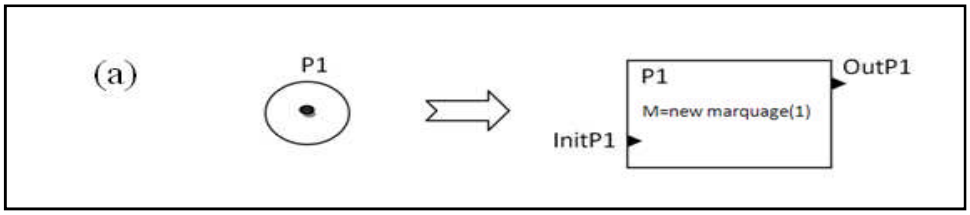

Fig 4.9. Représentation graphique de la transformation élémentaire d'une place en PDEVS

Algorithme 5.1 : Transformation des RdP en DEVS

```
Main_PN_DEVS
Input PN= (P,T,PRE,POST,M0)
Output CDEVS //coupled model
Begin :
   Create CDEVS as coupled DEVS model //void model
   For all transition ido
    createTDEVSi as atomic DEVS model
   end for
   forall places j do
    createPDEVSj as atomic DEVS model
   end for
   forallPDEVSjdo
      add 'InitPj' as intput port and join it to CDEVS.IN.InitP  //starting tokens
      add 'OutPj' as output port and join it to CDEVS.OUT.OutPj  //output stream
   end for
   for all TDEVSido
      add 'InitTi' as input port //initialize, stop, pause, release
      join 'InitTi' port to CDEVS.IN. InitT port //coupling
      add 'OutTi' as output port and join it to
      CDEVS.OUT.OutTi  //output stream
      add 'CTi' as output port // control: check, reserve,
      decrement, cancel
      for all PDEVSjdo
         if (PRE[i,j] > 0)  //upstream place
            add to PDEVSj 'CTiPj' as input port //check, reserve, decrement, cancel
            joinTDEVSi.OUT.CTi to PDEVSj.IN.CTiPj // coupling
            add to PDEVSj 'CPjTi' as output port //ok, busy ,number_of_free_tokens
            add to TDEVSi 'CPjTi' as input port //ok, busy ,number_of_free_tokens
            joinPDEVSj.OUT.CPjTi to TDEVSi.IN. CPjTi //coupling
            add to PDEVSj 'APjTi' as output port  //arc: value = PRE[i,j]
            add to TDEVSi 'APjTi' as intput port  //arc: value  = PRE[i,j]
            joinPDEVSj.OUT. APjTi to TDEVSi.IN.APjTi // coupling
         end if
         if(POST[i,j] > 0)   //downstream places
            add to TDEVSi 'ATiPj' as output port  //arc: value= POST[i,j]
            add to PDEVSj 'ATiPj' as input port  //arc: value = POST[i,j]
            joinTDEVSi.OUT.ATiPj to PDEVSj.IN.ATiPj //coupling
         end if
     end for
   end for
endMain_PN_DEVS
```

4.6.3.2. Transformation des transitions d'un RdP

De même que pour les places, les transitions vont être transformées en modèles DEVS atomiques. Pour toutes transition T_i du modèle RdP, va correspondre un modèle atomique DEVS que nous appelons dans la suite « $TDEVS_i$ » Pour « Transition-DEVS ». Ce modèle contient, par défaut, quatre ports ; un pour l'initialisation « $InitT_i$ » et un autre pour les sorties des résultats « $OutT_i$ ». Les deux autres ports (CPT_i, CT_i) vont servir le modèle en moyens d'action et de perception. Avec ces deux ports, TDEVS pourra envoyer des requêtes pour voir combien de jetons contiennent les CDEVS se situant en son amont. Ainsi, il peut décider de son franchissement. La figure 4.10(b) illustre la transformation de base d'une transition en TDEVS. Notons que le couplage va être réalisé dans une autre étape en appliquant l'algorithme 5.1

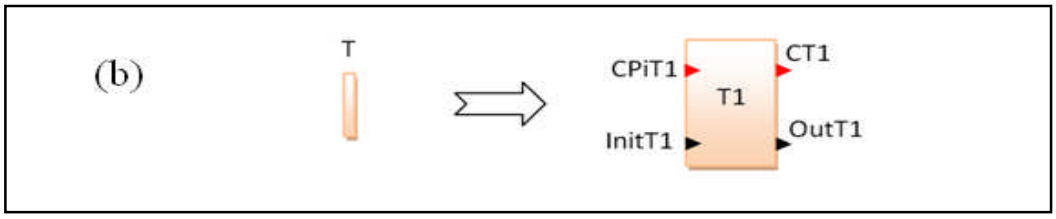

Fig 4.10. Représentation graphique de la transformation élémentaire d'une transition en TDEVS

4.6.3.3. Transformation des arcs d'un RdP

Les arcs, quant à eux, vont être transformés en couplages entre les modèles PDEVS et TDEVS. Nous proposons deux types de couplages : le premier est pour l'incrémentation et décrémentation des jetons (dans la figure 4.11, ce type est représenté par la couleur noire). Le deuxième type, nous l'appelons couplage de contrôle. Il est représenté dans la figure 4.11 par la couleur rouge. Ces couplages vont servir les TDEVS en matière de « check » en envoyant des événements à travers. Ces événements reflètent des requêtes des TDEVS envoyées aux PDEVS, les demandant s'il y en a assez de jetons à réserver. Ainsi que l'annulation de la réservation et la décision de franchissement vont être réalisées à grâce aux événements de contrôle circulants dans ces mêmes interconnexions.

La figure 4.11 représente les transformations élémentaires des arcs en IC entre les PDEVS et les TDEVS. (c) par exemple, représente un arc allant d'une transition vers une place. Cet arc va correspondre à un couplage dont les événements seront des valeurs de nombre de jetons à incrémenter. (d) dans cette même figure (4.11), représente un arc allant d'une place vers une transition. Dans ce cas la place est en amont de la transition. Donc la validation de cette dernière va dépendre du nombre de jetons se trouvant dans la place. C'est

pourquoi des interconnexions de contrôle (en rouge) seront établies entre le TDEVS et le PDEVS correspondants, outre que le couplage IC de décrémentation (en noir). Dans la même figure, (e) représente un conflit franc entre deux transitions. Dans ce cas chaque TDEVS aura ses propres IC indépendamment de l'autre TDEVS. Ainsi, l'aspect de concurrence ne rencontrera aucun problème. La partie (f) de la figure représente le cas où plusieurs places se situent en amont de la même transition. Alors le même port de sortie du TDEVS correspondant à la transition va être couplé à l'ensemble des PDEVS correspondants aux places. Ainsi le même événement envoyé par TDEVS sera aperçu par tous les PDEVS, du moment que tout le monde est concerné par la requête de la transition ; que ce soit une demande de nombre de jetons, réservation, annulation ou franchissement.

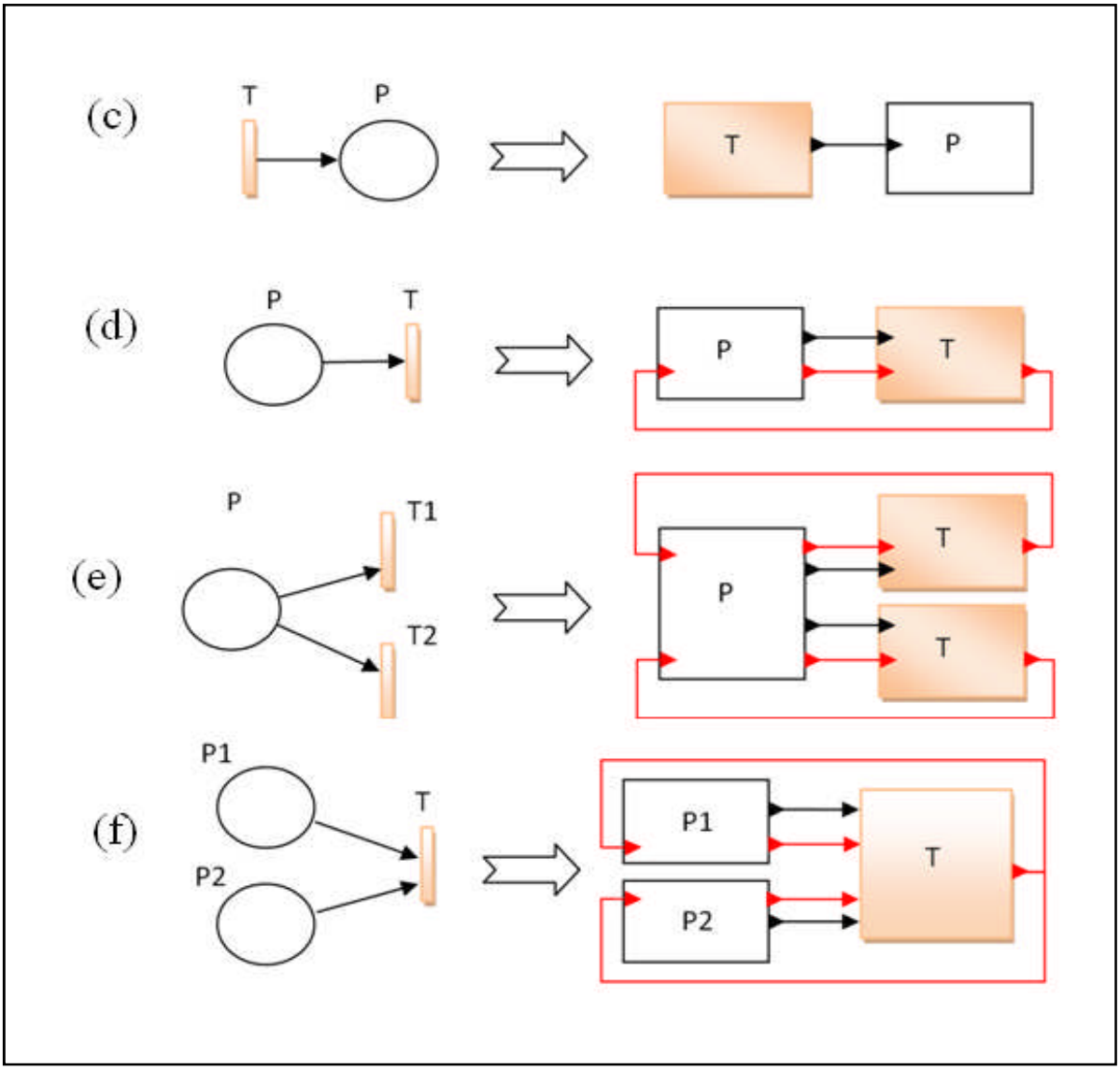

Fig 4.11. Représentation graphique de la transformation des arcs en IC entre les PDEVS et TDEVS générés.

4.6.3.4. Structure du modèle DEVS résultant

Le modèle couplé CDEVS est composé de plusieurs modèles DEVS atomiques comportant autant de places et de transitions que le RdP source. La figure 4.12 illustre le modèle CDEVS correspondant au RdP en entrée. Le modèle DEVS correspondant à « la transition » du RdP (soit TDEVS pour « Transition DEVS ») est caractérisé par un port de sortie « control » (CT1 dans l'exemple) qui a pour rôle soit d'émettre des évènements aux places en amont, vérifier le nombre de jetons ou informer de son franchissement. En revanche, TDEVS reçoit des évènements auprès des modèles correspondant aux places en amont (soient PDEVS pour « Place DEVS » dans ce qui suit) par des ports de contrôle autant de places (soient CPiT1). Les ports de contrôle sont illustrés Figure 4.12 (couleur rouge).

L'évolution du modèle CDEVS commence par un évènement externe « initialize » aperçu par le port d'entrée InitT(couleur orange) et qui sera diffusé sur tous les TDEVS, ainsi que la pause, la reprise et l'arrêt qui sont des événements reçus par le même port. A noter que ce port est couplé uniquement par les TDEVS du moment que les éléments responsables de la dynamique dans les RdP sont les transitions et non plus les places.

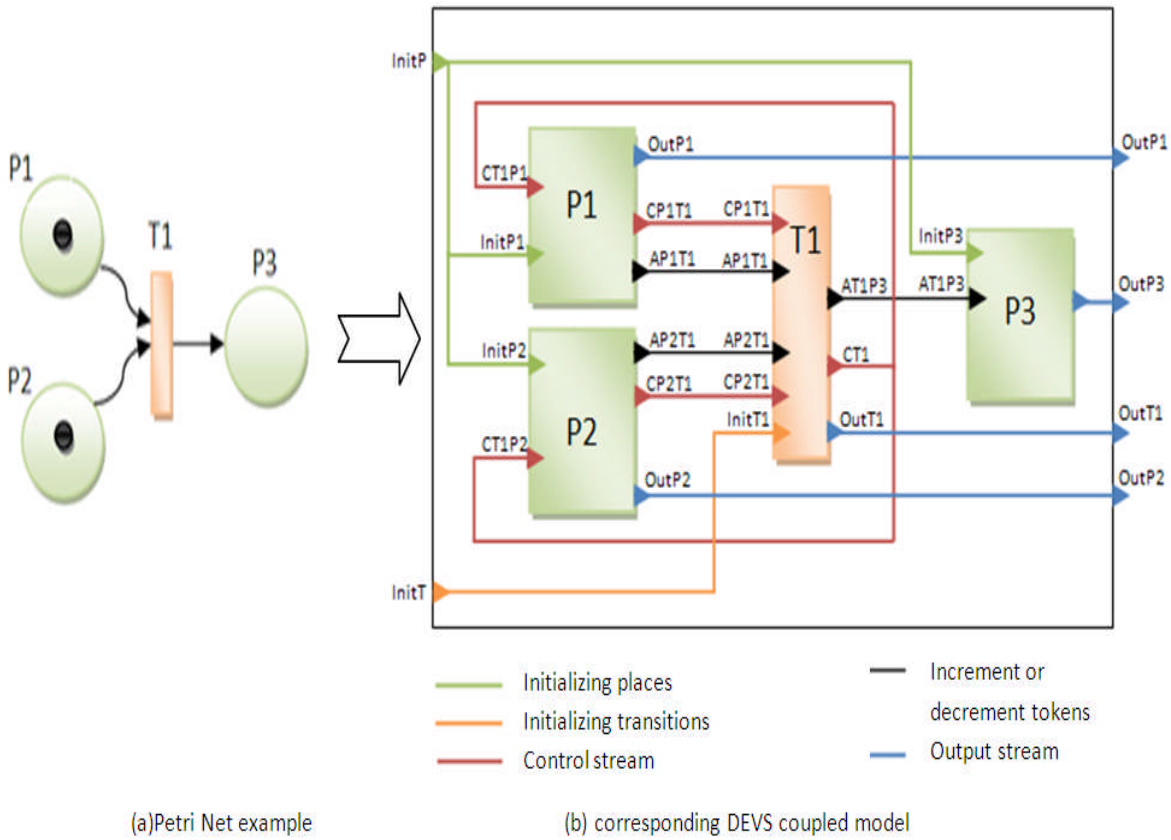

Fig .4.12 Exemple de transformation d'un RdP en CDEVS.

4.6.3.5. La dynamique du modèle DEVS résultant

Dans cette approche de transformation nous modélisons la dynamique DEVS couplé généré par l'algorithme précédent en respectant les fonctions offertes par le formalisme DEVS soient δint, δext, δcon et λ.

Après initialisation des places (PDEVS) par le marquage initial et après avoir lancé l'évolution du modèle par l'évènement « initialize » aperçu par toutes les transitions (TDEVS). Ces dernières se mettent en état «checking » (grace à la fonction δext) pour vérifier si le nombre de jetons des places en amont est suffisant pour réaliser un franchissement. Un événement « check » est alors envoyé (par la fonction λ). Les PDEVS concernés reçoivent cet évènement, ils renvoient à leurs tours le nombre de jetons libres (qui ne sont pas réservés par d'autres transition): grâce toujours à la fonction λ. Si le nombre de jetons est suffisant pour valider la transition (TDEVS), alors cette dernière va changer l'état de « checking » pour qu'il soit « reserving » et envoie l'événement « reserve », grâce à la fonction λ toujours. Le franchissement ne se fait pas directement. On doit passer par une réservation des jetons pour pallier la contrainte des conflits (cas des places qui sont en amont de plusieurs transitions), du moment que les transitions sont en permanente concurrence. De cette façon la propriété des RdP en matière de dynamique et de concurrence est conservée fidèlement dans notre approche de transformation.

Les PDEVS recevant l'événement « reserve » renvoient « ok » s'il reste encore assez de jetons (>= au poids de l'arc), « fail » sinon. Si TDEVS reçoit au moins un « fail » il renvoie immédiatement à tous les PDEVS un signal « cancel » pour libérer les jetons probablement réservés, sinon il remet son état « validated ». A ce moment là, la transition peut passer au franchissement et renvoie par conséquent « decrement » aux PDEVS qui vont détruire les jetons réservés par le TDEVS en question. TDEVS envoie simultanément « increment » aux PDEVS se situant en aval pour incrémenter leur nombre de jetons avec la valeur aperçue par le port d'entrée (poids de l'arc). Après le tir d'un TDEVS, il se remet en état « checking » et ainsi de suite.

Les fonctions δext, δint, δcon et λ, caractérisant les modèles TDEVS sont résumées dans la table.4.1. Les deux premières colonnes représentent les entrées ; soient les événements et l'état courant. Les autres colonnes affichent les sorties de chaque fonction. Les lignes de la table sont regroupées séparément pour chaque état courant. Ainsi pour les modèles PDEVS ; les fonctions sont représentées table 4.2. Par convention, si tous les évènements ont le même impact, on écrit alors « all events ». Les cellules vides signifient l'absence de valeurs: pour λ c'est l'absence d'événements. Pour δext, δint et δcon; cela signifie que la fonction n'a pas produit en sortie un état, donc l'état du modèle reste en principe le même. Le symbole « & » indique que les événements sont simultanés.

Evénement	**Etat courant**	**δ_{ext}**	**δ_{int}**	**δ_{con}**	**λ (état courant)**
Initialize	all states	checking			out
Pause		paused			out
Stop		stopped			out
Release		checking			out
free_tokens	Reserving		reserving	reserving	reserve
Ok		validated, reserving		validated, reserving	
Fail		canceling		Cancel	
all events	Validated		checking		decrement & increment
all events	Canceling		checking		cancel

Tab.4.1 Les sorties des fonctions du modèle TDEVS

Evénement	**Etat courant**	**δ_{ext}**	**δ_{int}**	**δ_{con}**	**λ (état courant)**
initialize	all states	checking		Checking	out
check	Checking	checking	checking	Checking	free_ tokens
reserve		reserving		Reserving	
increment		incrementing		incrementing	
decrement		decrementing		decrementing	
cancel		checking		Checking	

check	Reserving	reserving	checking	Reserving	ok, fail
reserve		reserving		Reserving	
increment		incrementing		incrementing	
decrement		decrementing		decrementing	
cancel		checking		Checking	
check	incrementing	checking	checking	Checking	out
reserve		reserving		Reserving	
increment		Incrementin		incrementing	
decrement		Decrementin		decrementin	
cancel		Incrementin		incrementing	
check	decrementing	checking	checking	Checking	out
reserve		reserving		Reserving	
increment		Incrementin		Incrementing	
decrement		Decrementin		Decrementin	
Cancel		Decrementin		Decrementin	

Tab.4.2 Les sorties des fonctions du modèle PDEVS

4.6.3.6. Le conflit et la compétition

Les RdP se caractérisent par leur représentation formelle des systèmes. Les notions de parallélisme, conflit et compétition sont bien modélisés dans ce formalisme. Ce qui rend la tâche de transformation très délicate. Dans notre approche, ces aspects sont bien respectés du moment que les TDEVS ne procèdent pas directement au franchissement après avoir été validé. Ainsi, le modèle TDEVS doit passer par l'état réservation. S'il n'arrive pas à se valider, il libère automatiquement les jetons réservés pour laisser l'opportunité aux autres TDEVS de réserver les mêmes jetons. Autrement dit, si deux TDEVS visent le même jeton d'un PDEVS quelconque, alors un seul TDEVS pourra le réserver à un instant donné. Ainsi le franchissement se ferra d'une manière sûre dans ce genre de conflit. D'autre part, les RdP se caractérisent par leur spontanéité et leur franchissement aléatoire quant il s'agit de plusieurs transitions valides en même temps. On ne peut en aucun cas, confirmer quelle

transition parmi celles validées sera franchie en premier lieu. Ainsi, un simple programme séquentiel ne pourra pas implémenter cette caractéristique d'une manière assez fidèle. Dans notre cas, cet aspect est fidèlement respecté du point de vu conceptuel. Quant à l'implémentation, nous avons eu recours aux threads de Java [Oracle, 2012] de telle sorte que les modèle TDEVS soient en plein concurrence.

4.6.4. Exemple d'application

Dans cette partie, nous présentons un exemple de transformation de l'un des plus fameux RdP connus dans le domaine pédagogique de l'informatique : Producteur-Consommateur illustré dans la figure 4.13 Ce RdP est constitué de six places et quatre transitions dont la dernière consiste en un tampon de sept cases (le nombre de jetons reflète les cases vides).

Nous allons identifier les modèles DEVS de manière formelle, puis nous illustrons la transformation sous format graphique, du moment que les deux formalismes DEVS et RdP offrent la représentation graphique de leurs modèles. La figure 4.14 représente le modèle DEVS couplé fidèle au RdP du Producteur-Consommateur. Dans cette figure nous conservons la même signification des couleurs utilisée dans la figure 4.12.

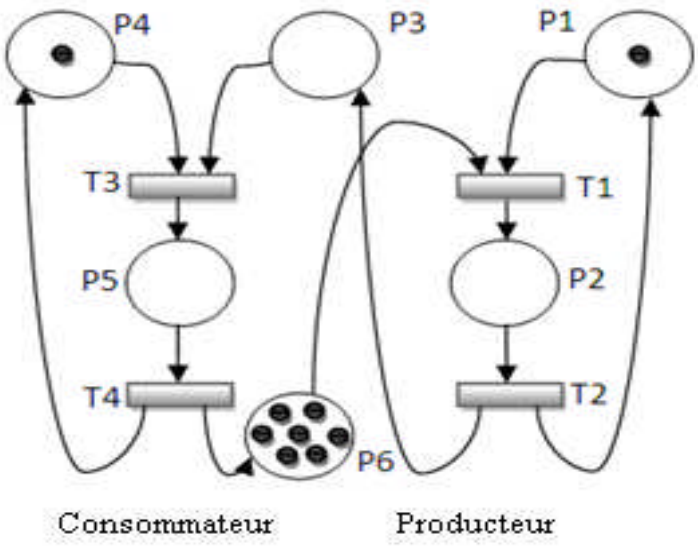

Fig.4.13RdPde producteur-consommateur

La définition formelle du RdP présenté dans la figure 4.13 est :

$PN = (P, T, PRE, POST, M_0)$ tel que:

$P = \{P1, P2, P3, P4, P5, P6\}$

$$T = \{T1, T2, T3, T4\}$$

$$PRE = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} POST = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} M_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 7 \end{pmatrix}$$

Formellement la transformation est présentée comme suit :

PN = (P, T, PRE, POST, Mo) → CDEVS=(X, Y, D, EIC, EOC, IC)

Où:

D = { P∪ T } = {P1, P2, P3, P4, P5, P6,T1, T2, T3, T4}

X = { InitP, InitT }

Y = {OutDi / Di est le modèle atomique qui représente Pi ou Ti}

= {OutP1, OutP2, OutP3, OutP4, OutP5, OutP6, OutT1, OutT2, OutT3, OutT4}

EIC = {

(CDEVS.InitP, PDEVS.IntPi) ∪ (CDEVS.initT , TDEVS.IntTj)

Tel que i Є N+ & i< nombre de places, j Є N+ & j < nombre de transitions

}

= {

(CDEVS.InitP, PDEVS.IntP1), (CDEVS.InitP, PDEVS.IntP2),

(CDEVS.InitP, PDEVS.IntP3), (CDEVS.InitP, PDEVS.IntP4),

(CDEVS.InitP, PDEVS.IntP5), (CDEVS.InitP, PDEVS.IntP6),

(CDEVS.InitT, TDEVS.IntT1), (CDEVS.InitT, TDEVS.IntT2),

(CDEVS.InitT, TDEVS.IntT3), (CDEVS.InitT, TDEVS.IntT4),

}

EOC={ (Pi.OutPi, CM.OutPi) ,

(Tj.OutTj, CM.OutTj)

} Tel que i Є N+ & I < nombre de places, et j Є N+ & j < nombre de transitions

={(P1.OutP1, CM.OutP1) , (P2.OutP2, CM.OutP2) , (P3.OutP3, CM.OutP3) ,

(P4.OutP4, CM.OutP4) , (P5.OutP5, CM.OutP5) , (P6.OutP6, CM.OutP6) ,

(T1.OutT1, CM.OutT1), (T2.OutT2, CM.OutT2),

(T3.OutT3, CM.OutT3), (T4.OutT4, CM.OutT4),

}

IC = { {(Pi.APiTj, Tj.APiTj) / PRE[i,j]>0 }

∪

{(Tj.ATjPi, Pi.ATjPi) / POST[i,j]>0 }

∪

{ {Tj.CTj} X {Pi.CTjPi} / PRE[i,j]>0 }

∪

{ (Pi.CPiTj , Tj.CPiTj} / PRE[i,j]>0 }

}Tel que i Є N+& I < nombre de places, et j Є N+ & j < nombre de transitions

= {

{(P1.AP1T1, T1.AP1T1), (P6.AP6T1, T1.AP6T1), (P2.AP2T2, T2.AP2T2),
(P3.AP3T3, T3.AP3T3), (P4.AP4T3, T3.AP4T3), (P5.AP5T4, T4.AP5T4)}
∪{(T1.AT1P2, P2.AT1P2), (T2.AT2P1, P1.AT2P1), (T2.AT2P3, P3.AT2P3),
(T3.AT3P5, P5.AT3P5), (T4.AT4P4, P4.AT4P4), (T4.AT4P6, P6.AT4P6) }
∪{ (T1.CT1,P1.CT1P1), (T1.CT1,P6.CT1P6), (T2.CT2,P2.CT2P2),
(T3.CT3,P3.CT3P3), (T3.CT3,P4.CT3P4), (T4.CT4,P5.CT4P5) }
∪ { (P1.CP1T1 , T1.CP1T1), (P6.CP6T1 , T1.CP6T1), (P2.CP2T2 , T2.CP2T2),
(P3.CP3T3 , T3.CP3T3), (P4.CP4T3 , T3.CP4T3), (P5.CP5T4 , T4.CP5T4) }
}

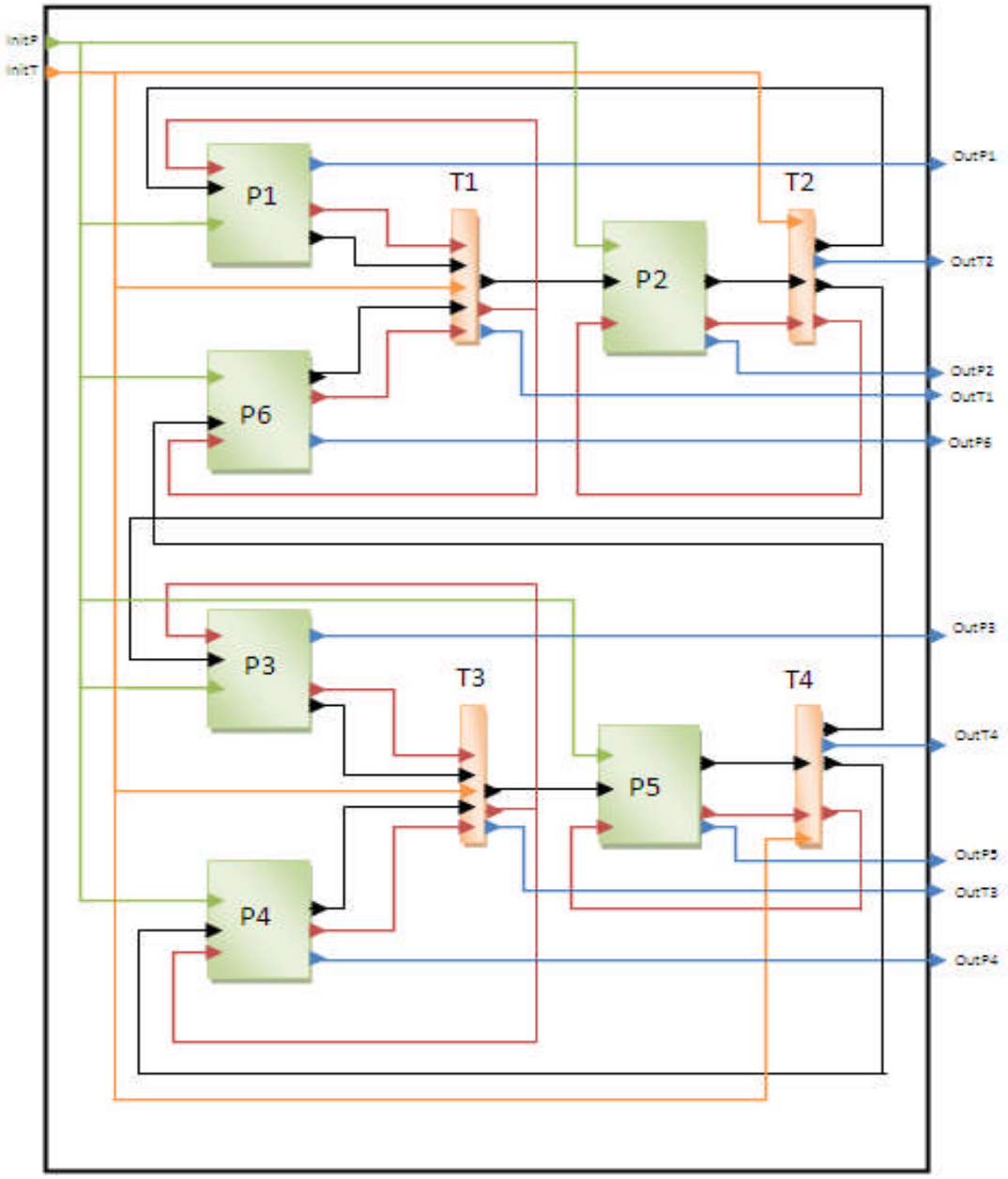

Fig. 4.14 Représentation graphique du DEVS couple correspondant au RdP producteur-consommateur

4.6.5. Discussion

Les réseaux de Petri, étant des outils formels de modélisation des systèmes dynamiques traitant parfaitement l'aspect de concurrence et de parallélisme entre processus, nécessitent un traitement délicat lors de leur transformation afin de ne pas perdre de leurs spécifications. Dans notre approche, la concurrence est conservée par la création d'un état transitoire des transitions. Il s'agit de la réservation des jetons avant de passer au franchissement. Ainsi, un jeton ne pourra pas participer, en même temps, au franchissement de deux transitions qui sont en conflit. En revanche, la transition doit libérer immédiatement les jetons qu'elle a réservés si elle n'arrive pas à se valider. Ceci afin de ne pas paralyser les autres transitions qui sont en conflit avec elle.

Dans cette section nous avons parlé des RdP généralisés pour amener le lecteur à comprendre le mécanisme de transformation. Néanmoins, les autres extensions des RdP telles que les RdP colorés peuvent aussi être transformés. Dans ce cas, les jetons ne vont plus être banalisés. On aura besoin d'étendre le type de représentations de ces derniers afin qu'il comprenne une liste énumérant l'ensemble de couleurs. Ainsi, lors de l'émission de l'événement « check » auprès d'une transition. Les places sises en amont doivent vérifier le port les reliant à cette transition pour n'envoyer que le nombre de jetons libres ayant la même couleur que celle spécifiée au niveau de ce port. Avec le même principe, la destruction de jetons aura lieu, suite à un tir de transition.

En outre, le formalisme DEVS offre une flexibilité quant à la structure interne de ses modèles [Barros, 1996 ; Baati, 2007]. Des modèles peuvent disparaitre, d'autres peuvent prendre le relai. Des interconnexions peuvent avoir naissance, d'autres changent éventuellement les modèles qu'elles couplent. Cet aspect de structure dynamique des modèles DEVS motivera les modélisateurs à faire évoluer leurs modèles des RdP dans les plateformes à base de formalisme DEVS. Ainsi la complexité des RdP liée à la représentation des changements structurels des systèmes va être simplifiée, du moment qu'un modèle DEVS peut avoir plusieurs structures, par conséquent, il représentera plusieurs RdP à la fois.

4.7. Conclusion

Dans cette section nous avons présenté une approche de transformation des réseaux de Petri en modèles DEVS. Par un algorithme qui construit systématiquement des modèles DEVS atomiques pour chacune des places et des transitions, on peut avoir un modèle DEVS couplé en spécifiant les interconnexions entre ces modèles. Et ce, grâce aux matrices PRE et POST qui consistent en une autre définition de l'ensemble « A » des arcs.

Ce travail rentre dans le cadre de transformation des modèles à base de multi-formalismes en un modèle uniforme soit DEVS, dans notre cas. Notre choix de ce formalisme a été fondé sur la force DEVS en matière d'unification et de couplage de modèles. Se caractérisant par son abstraction, son indépendance des implémentations et son aptitude à modéliser des systèmes complexes sous forme d'un modèle hiérarchique, DEVS est un formalisme qui peut être l'unificateur des modèles des systèmes à évènements discrets [Zeigler, 2000].

Par la transformation présentée dans ce papier, les RdP peuvent bénéficier de la simulation offerte par des multiples plateformes basées sur DEVS. Ainsi la question de leur validation aura une réponse via ces outils. En effet, DEVS offre la possibilité de procéder à des vérifications formelles de ses modèles.

Nos perspectives portent sur la mise en application de tels algorithmes sur des modèles complexes tels que les processus industriels où la notion de parallélisme, d'interaction, et de distribution de calculs sont à considérer. Nous envisageons aussi, la mise en œuvre d'une plateforme de test et de validation des RdP tout en profitant des outils open source basés DEVS.

Chapitre 5

Le système proposé

5.1. Introduction

L'approche de transformation proposée dans le chapitre précédent rentre dans le cadre de la multi-modélisation des systèmes complexes et le couplage entre leurs modèles composants où Le problème de l'hétérogénéité est résolu par le concept de transformation.

Dans ce chapitre nous allons présenter une application de la modélisation et de la simulation des modèles DEVS. Cette application se caractérise par sa modularité et sa capacité de réutilisation des modèles stockés sous format XML. Pour le moment notre application intègre l'approche de transformation proposée dans le chapitre précédent. Elle pourra éventuellement intégrer d'autres modules où différents formalismes seront aussi manipulés.

5.2. Architecture générale du système proposé

5.2.1. Introduction :

Notre application est développée en java sous la plate forme NetBeans. Avant de détailler l'architecture du système proposé, nous allons brièvement présenter l'environnement de développement utilisé.

Le concept Java a été développé par l'équipe de James Gosling de Sun Microsystems Inc. Dans le but d'obtenir des programmes téléchargeables et indépendants du support d'exécution, machine et système d'exploitation. Le marché ciblé est celui des machines portables de faible capacité mais pouvant être connectées au réseau. Les promoteurs de Java

en disent que c'est un langage orienté objet, simple, réparti, robuste, sûr, indépendant de l'architecture, portable, efficace, «multithread » et dynamique.

Java comprend un ensemble riche de classes regroupées par domaines dans différents paquetages est fourni d'origine avec Java. Cette standardisation de fait de nombreuses classes devrait permettre d'éviter les problèmes rencontrés avec C++, où chaque éditeur de compilateur a développé ses propres classes, rendant difficile la recompilation d'un programme source C++ dans un environnement de développement différent de celui d'origine. La grande variété des classes proposées dans les paquetages Java permet une programmation facile et rapide d'applications complexes.

Nos motivations à utiliser le langage java est basées sur le fait qu'il est :

• Un langage orienté objet : ce qui facilite la modularité. Ainsi nous aurrons une souplesse quant à l'intégration des formalismes ou interfaces de modélisation déjà existantes totut en bénéficiant des boblothèques Java.

• Un langage qui est indépendant de plateforme d'exécution : Java utilise Java Run Time machine pour l'exécution des ses application comme étant une sous couche d'abstraction qui survole les systèmes d'exploitation.

• Un langage efficace et portable : Java a prouvé dans le temps son efficacité durable.

• Unlangage qui assure le parallelisme : Java comporte des classes dont les objets peuvent s'exécuter en tant que sous processus indépendant (multi-threading). Par ailleurs, les formalismes de modélisation des systèmes dynamiques respectent bien cette caractéristique.

• Un langage puissant, maintenable et simple d'utilisation.

Dans la figure 5.1 une représentation générale l'architecture des composants Java [Oracle, 20012]

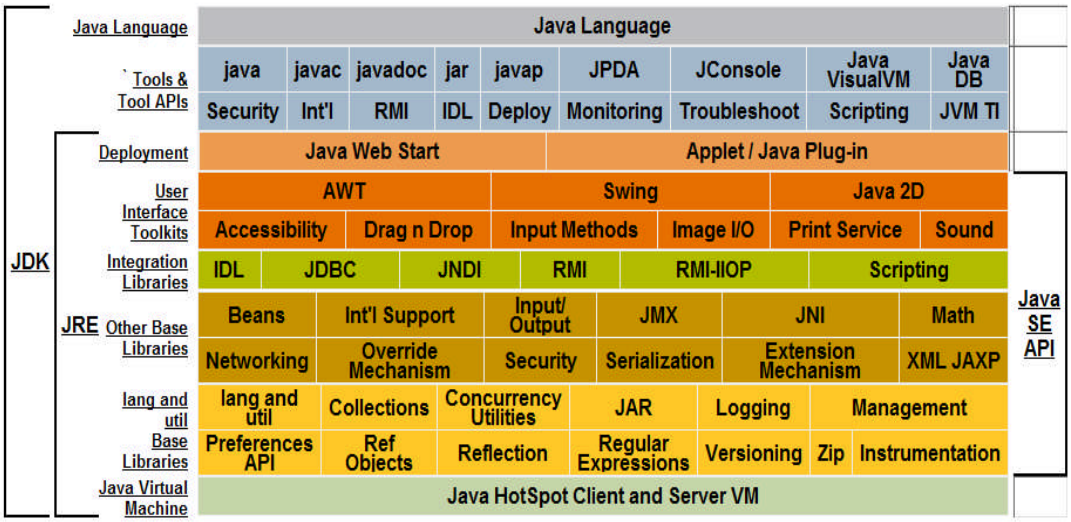

Fig.5.1 Architecture des composants Java [Oracle, 2012]

5.2.2. Architecture de l'application de M&S basée DEVS :

Le système proposé est caractérisé par une structure multicouches dont la modularité représente un principe à respecter dans toutes les phase de développement. Cet aspect est un besoin naturel dans le cas des plateformes multimodélisation. Ainsi l'intégration des éventuels formalisme doit etre une tche systématique. Dans la figure 5.2nous illustrons l'architecture globale du système proposé.

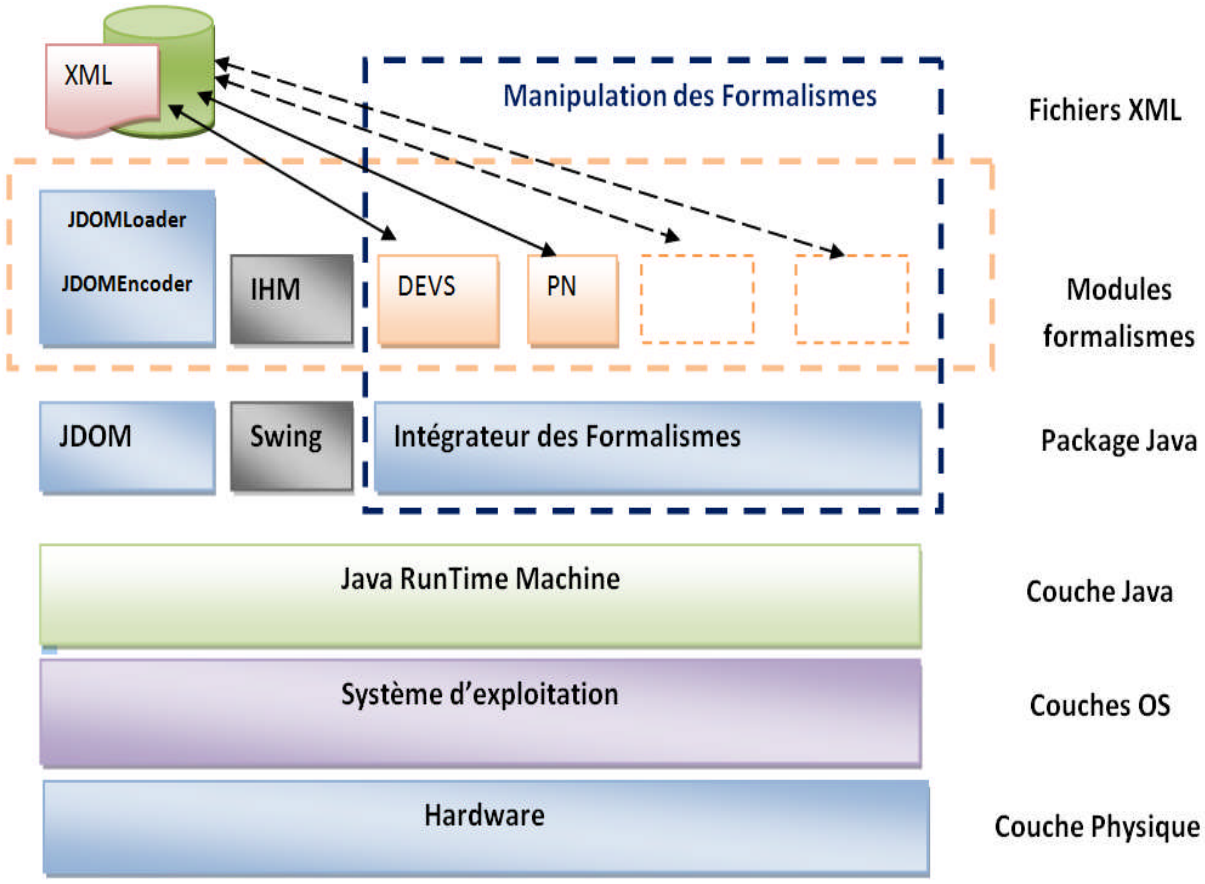

Fig.5.2 Architecture multicouches du système proposé

5.2.2.1. Présentation des différentes couches de l'application

Dans cette section nous allons présenter les différentes couches illustrées figure 5.2.

- La machine virtuelle Java (Java Run Time Machine) : C'est la couche où les programmes java sont exécutés et gérés. Cette couche assure l'indépendence d'implémentation des systèmes d'exploiation. Ainsi tout programme java pourra s'executer sur n'importe quelle machine avec n'importe quel système d'exploitation (Les deux premières couches: système d'exploitation et hardware).

- Packages Java : Notre application intègre un ensemble de bibliothèques. Parmi elles il y a celles qui sont importées telles que JDOM (Java Document Object Model) et Swing et d'autres qui sont propres à l'application telle que l'intégrateur des formalismes qui est un ensemble de classes assurant le fonctionnements des formalismes à rajouter à l'application. Le package Swing assure les fonctionnalités graphiques. Quant à la bibliothèque JDOM ; son role est d'assurer la manipulation des fichier XML. Nous allons la détailler d'avantage par la suite.
- La couche des formalismes : Elle comprend l'ensemble des classes contenant les règles et contraintes des formalismes auxquelles le modélisateur doit avoir recours pour réaliser son modèle à traves une interface conviviale IHM développée par les objets Swing. Les classes des formalismes sont munies des plugins (schématisés par des flèches à double sens dans la figure 5.2) afin qu'elles puissent intéragir ave JDOM, et par conséquent, avec les fichiers XML.
- La couche des fichiers XML : Notre application stocke ses modèles sous forme de fichiers XML pour une éventuelle réutilisation et même pour assurer la portabilité vers d'autres plateformes. La manipulation directe est assurée par des classe de la biblothèque JDOM.

5.2.2.2. L'API JDOM et les fichiers XML

5.2.2.2.1. Origines de JDOM :

Au niveau des API Java, la manipulation basique des fichiers XML est assurée par l'API « SAX » pour « Simple API for XML » comme étant un interpréteur ou parseur. Ce type de parseur utilise des événements pour piloter le traitement d'un fichier XML. Un objet (nommé handler en anglais) doit implémenter des méthodes particulières définies dans une interface de l'API pour fournir les traitements à réaliser : selon les événements, le parseur appelle ces méthodes.

JDOM pour « Java Document Object Model » est une API Java pour la manipulation avancée des fichier XML[Hunter et McLaughlin, 2012]. Elle utilise des collections SAX pour parser les fichiers XML. Avec JDOM, on peut modéliser, parcourir et manipuler un document XML. Le principal rôle de DOM est de fournir une représentation mémoire d'un document XML sous la forme d'un arbre d'objets et d'en permettre la manipulation (parcours, recherche et mise à jour). A partir de cette représentation (le modèle), JDOM propose, non seulement, de parcourir le document mais aussi de pouvoir le modifier. Ce dernier aspect est l'un des aspects les plus intéressants de JDOM.

A noter que JDOM comprend des classes qui assurent l'interfacage avec DOM (Doccument Object Model). Ce dernier est défini pour être indépendant du langage dans lequel il sera implémenté. DOM n'est donc qu'une spécification qui, pour être utilisée, doit être implémentée par un éditeur tiers. DOM n'est donc pas spécifique à Java.

Bien que JDOM se caractérise pa sa simplicité, Il est très pratique pour développer des applications complexes autour de XML avec DOM.

5.2.2.2.2. Créer des fichier XML par JDOM :

Pour créer un fichier XML, il suffit de construire chaque élément (noeuds) puis de les ajouter les uns aux autres de façon logique. Un noeud est une instance de org.jdom.Element.La figure 5.3et 5.4 montre un nœud simple et détaillé respectivement. L'eregistrement d'un fichier XML est présenté dans la figure 5.5.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <models>
  - <AtomicDEVS>
      <id>T1</id>
      <name>T1</name>
    </AtomicDEVS>
  </models>
```

Fig.5.3 Nœud XML par JDOM

```
<?xml version="1.0" encoding="UTF-8" ?>
- <models>
  - <AtomicDEVS>
      <id>T1</id>
      <name>T1</name>
    </AtomicDEVS>
  - <AtomicDEVS>
      <id>T2</id>
      <name>T2</name>
    </AtomicDEVS>
  - <AtomicDEVS>
      <id>P1</id>
      <name>P1</name>
    </AtomicDEVS>
  - <AtomicDEVS>
      <id>P2</id>
      <name>P2</name>
    </AtomicDEVS>
  </models>
```

Fig.5.4 arborescence de nœuds XML par JDOM

```
static void enregistreFichier(String fichier) throws Exception
{
      XMLOutputter sortie = new XMLOutputter(Format.getPrettyFormat
      sortie.output(document, new FileOutputStream(fichier));
}
```

Fig.5.5 Enregistrer fichier XML par JDOM

5.2.2.2.3. Passer de JDOM à DOM et l'inverse:

Comme nous l'avons vu auparavent ; DOM n'est pas spécifique à java. Il nous arrive parfois d'aller de JDOM vers DOM et inversement. La figure 5.6 illusre cette transformation de document.

```
import org.jdom.input.*;

org.jdom.Document DOMtoJDOM(org.w3c.dom.Document documentDOM) throws
{
    //On utilise la classe DOMBuilder pour cette transformation
    DOMBuilder builder = new DOMBuilder();
    org.jdom.Document documentJDOM = builder.build(documentDOM);
    return documentJDOM;
}
```

Fig.5.6 Passage de DOM à JDOM.

```
import org.jdom.output.*;

org.w3c.dom.Document DOMtoJDOM(org.jdom.Document documentJDOM) throws
{
    //On utilise la classe DOMOutputter pour cette transformation
    DOMOutputter domOutputter = new DOMOutputter();
    org.w3c.dom.Document documentDOM = domOutputter.output(documentJD
    return documentDOM;
}
```

Fig.5.7 Passage de JDOM à DOM.

5.3. Implémentation de l'application

Dans cette partie nous allons présenter les principales interfaces pour l'exploitation de l'application que nous appelons DEVSModSim pour « DEVS Modélisation et Simulation ».

La figure 5.8 montre la console principale de l'application. Elle est divisée en deux parties : celle qui est à gauche est pour le contrôle et l'autre qui est à droite sert d'écran d'affichage des modèles sous format graphique. Dans la partie contrôle, on distingue deux listes (pour visualiser les modèles DEVS atomiques et couplés). Dans cette même partie, il y a un panneau pour contrôler la simulation (initialisation : pause et arrêt). En dessous, il y a un écran d'affichage textuel des étapes d'exécution et éventuels logs.

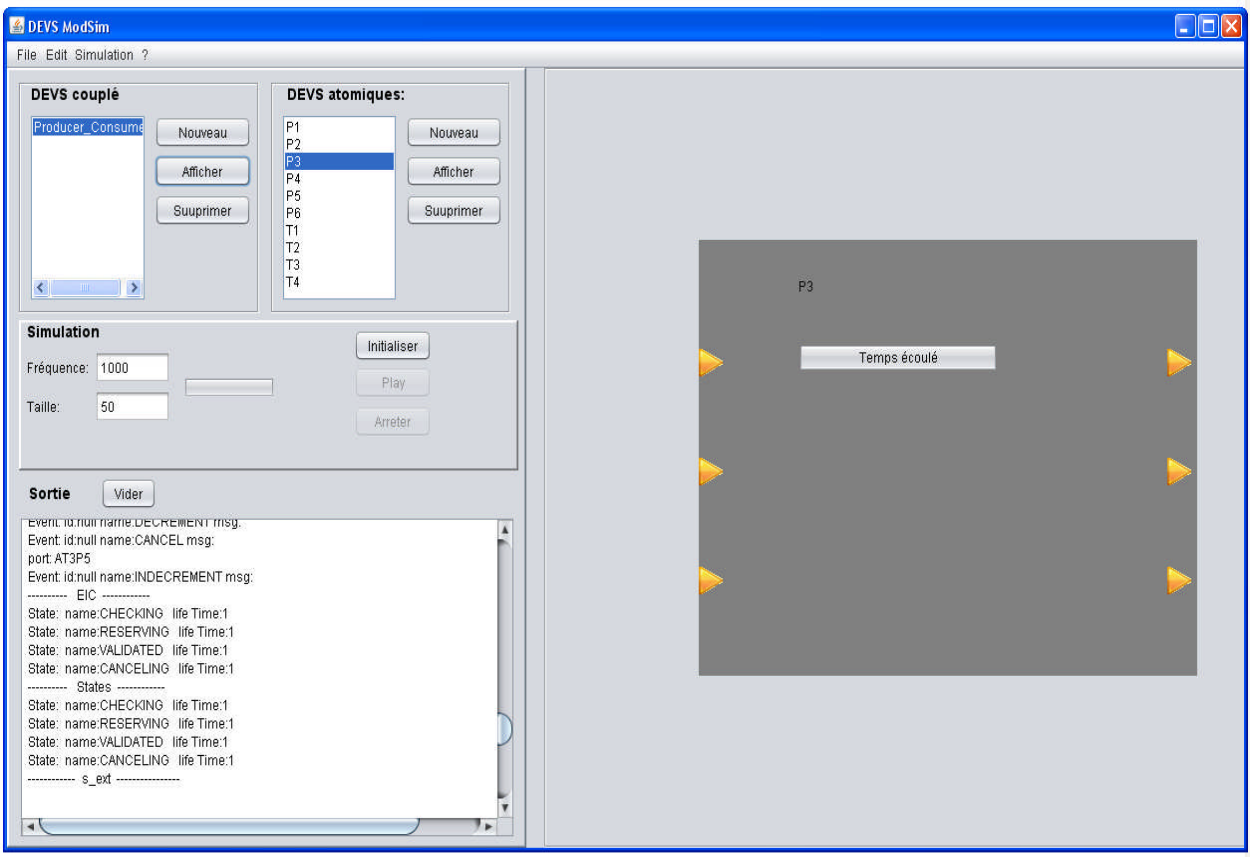

Fig.5.8 Interface d'entrée de l'application.

La figure 5.9 illustre l'interface d'édition d'un modèle atomique. Au niveau de cette interface, on peut créer ou modifier un modèle DEVS atomique. Changer ses états, ses ports d'entrée ou de sortie, ou même ses fonctions de transition.

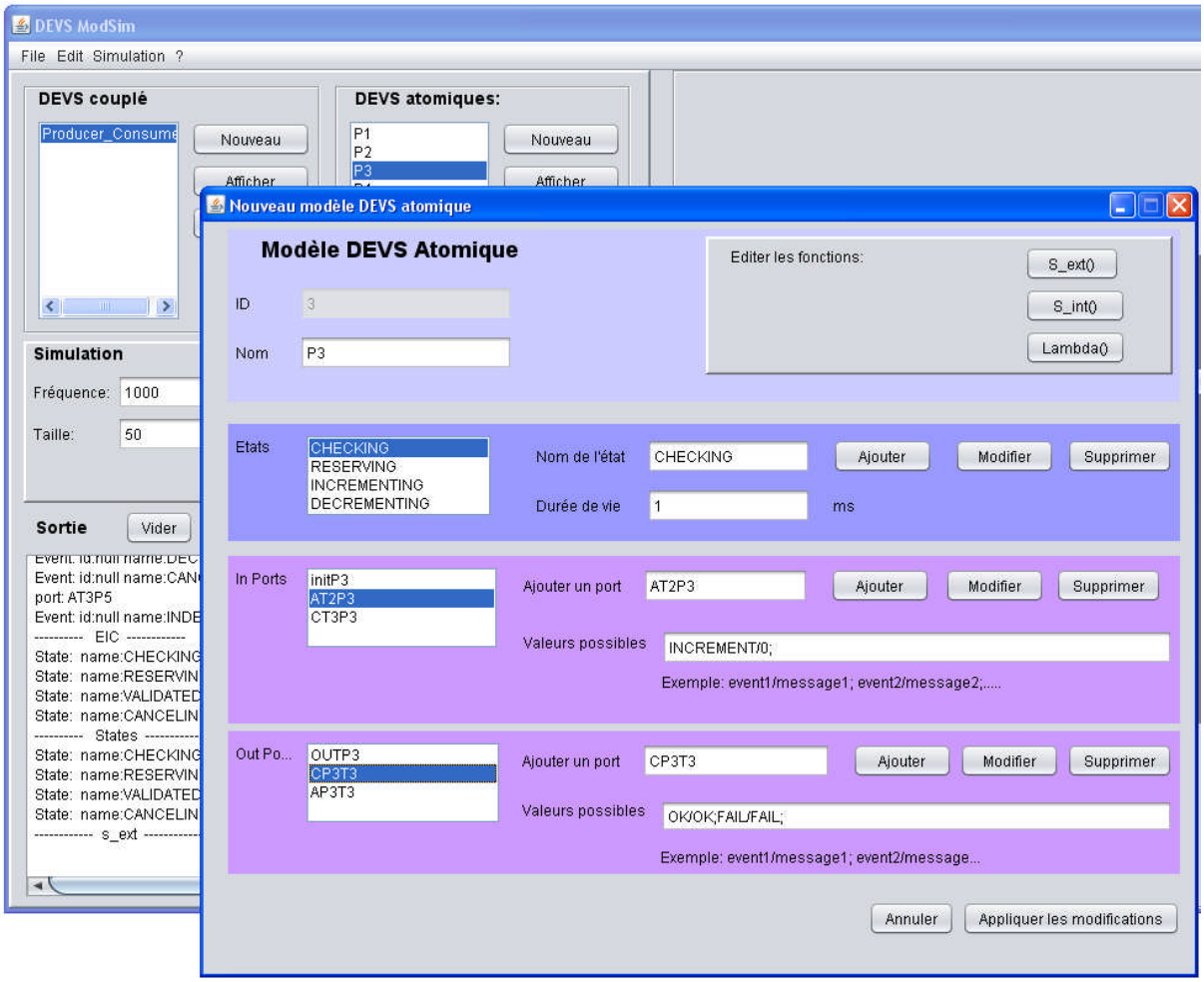

Fig.5.9 Interface d'édition d'un modèle atomique.

Les figures5.10, 5.11, 5.12 illustrent les interfaces d'édition des fonctionsde transition δ_{ext},δ_{int} et λ respectivement.

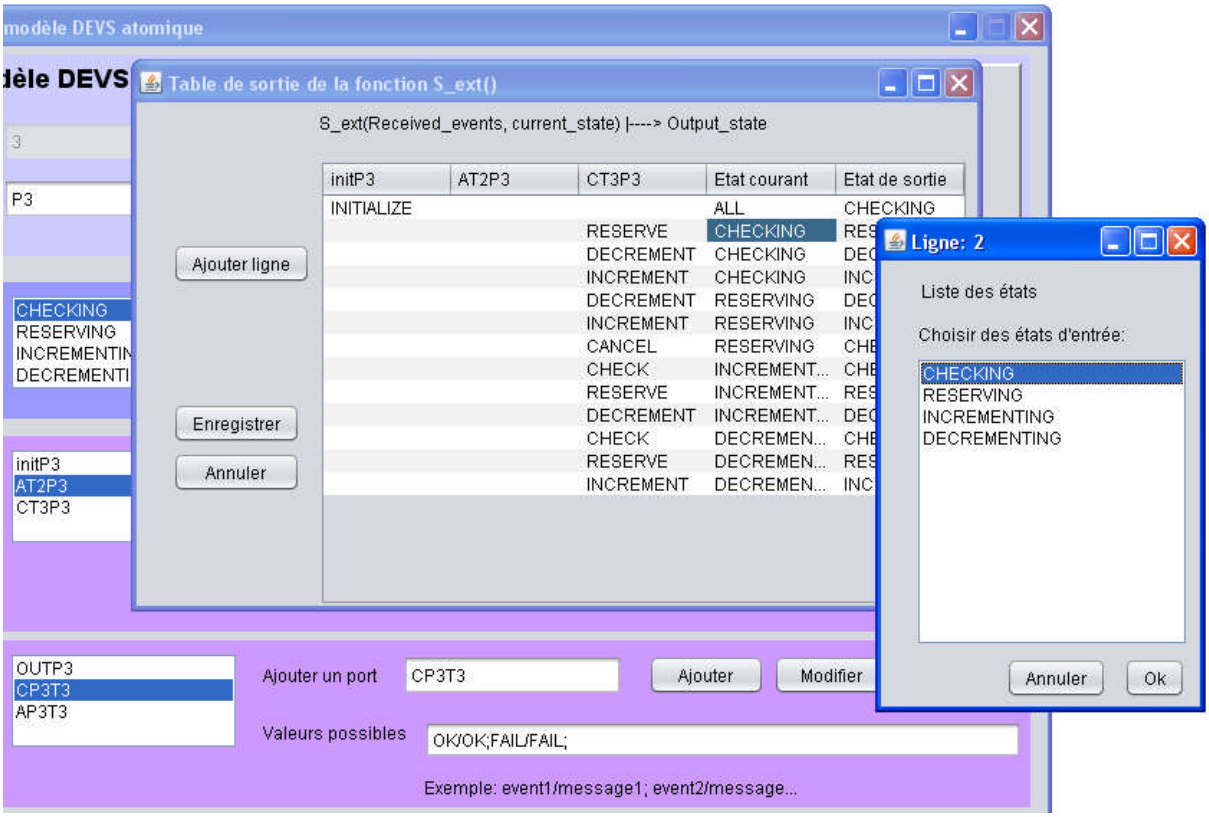

Fig.5.10 Interface d'édition de la fonction δ_{ext}

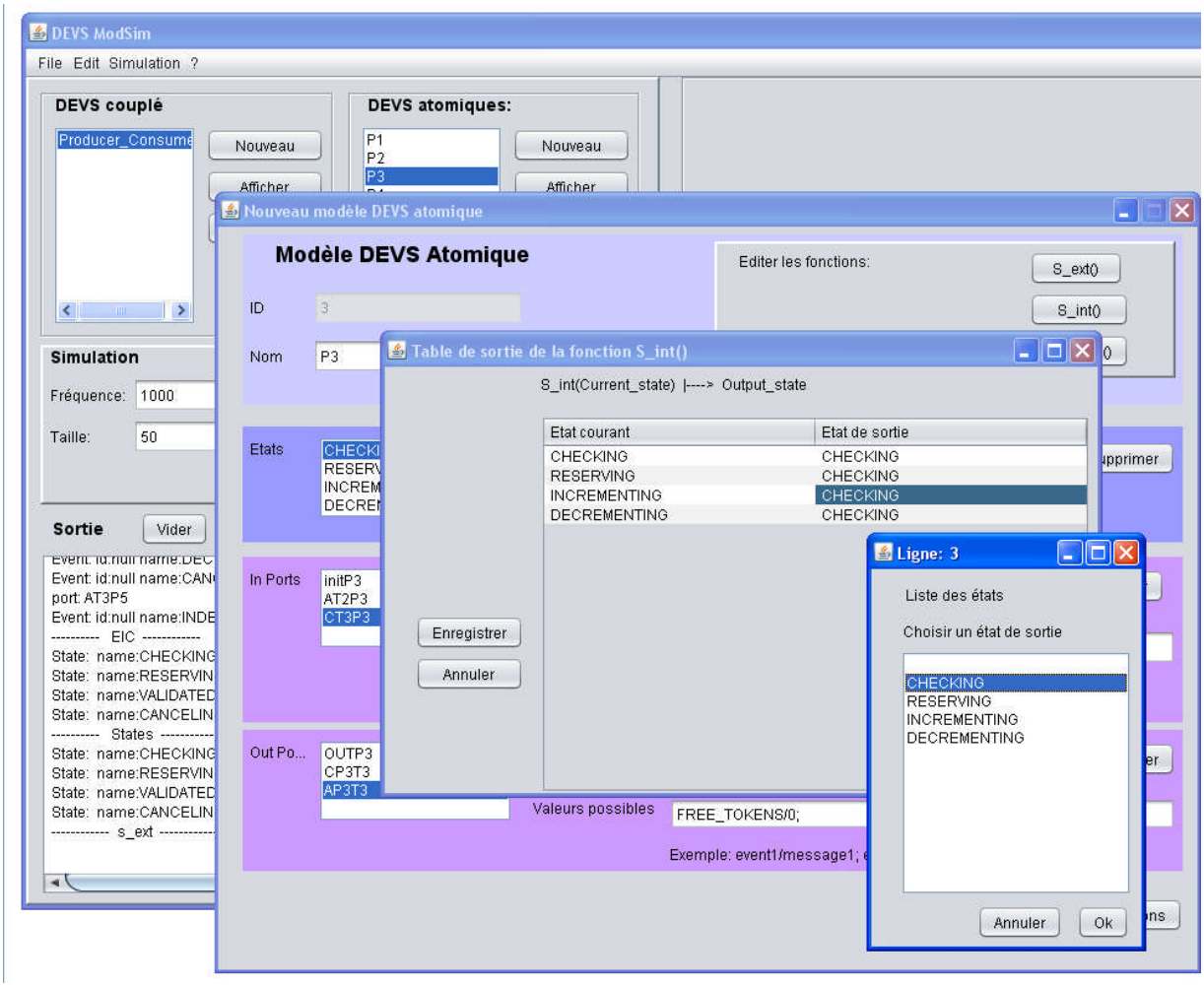

Fig.5.11 Interface d'édition de la fonction δ_{int}

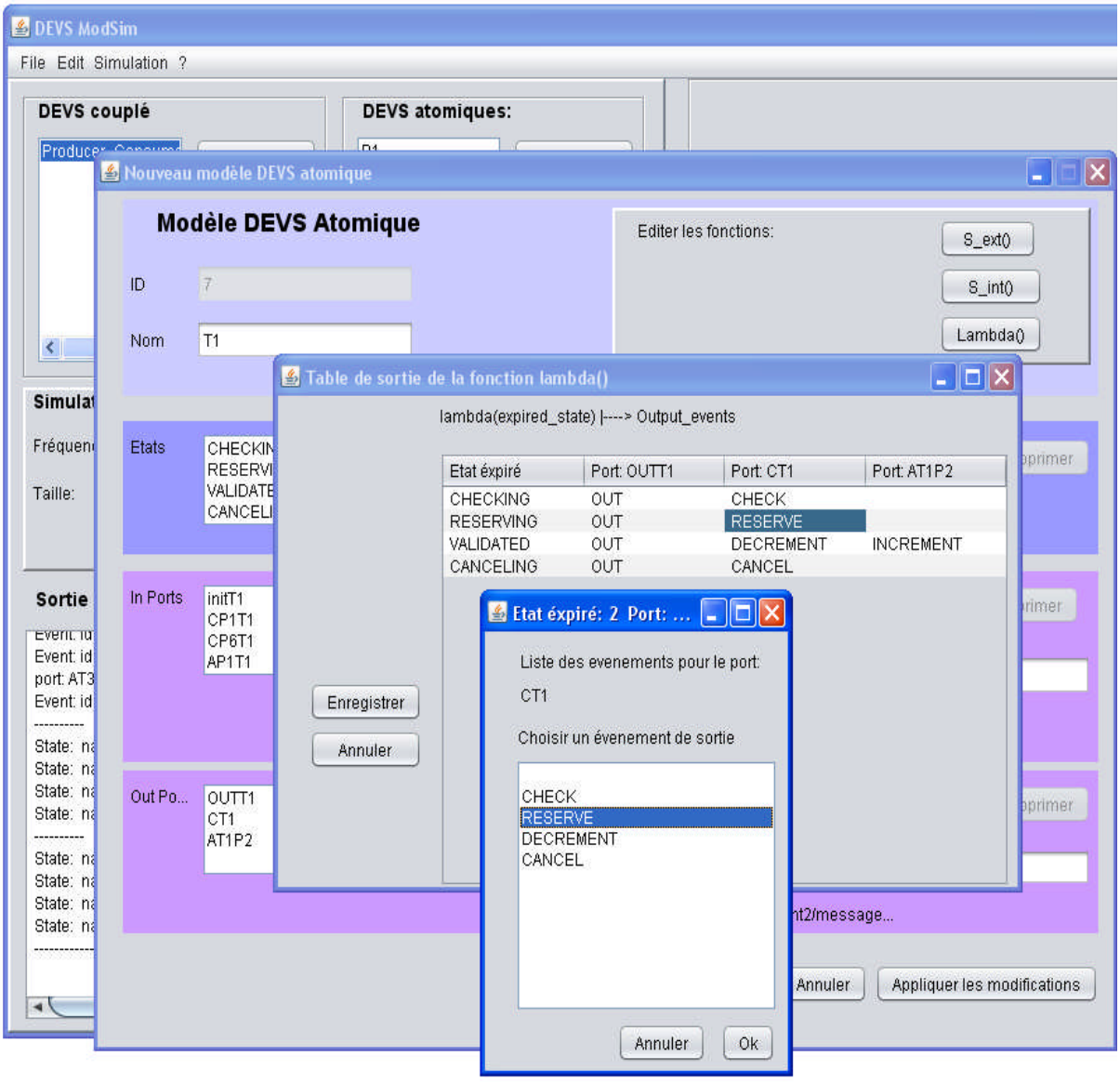

Fig.5.12 Interface d'édition de la fonction λ

Maintenant que nous avons passé en revue les principales interfaces avec lesquelles on peut créer, modifier ou supprimer un modèle DEVS atomique, nous allons illustrer celles de mise à jour d'un modèle DEVS couplé. Nous allons voir, ainsi, comment manipuler les différents types de couplage. La figure 5.13 illustre l'interface principale d'édition d'un modèle DEVS couplé.

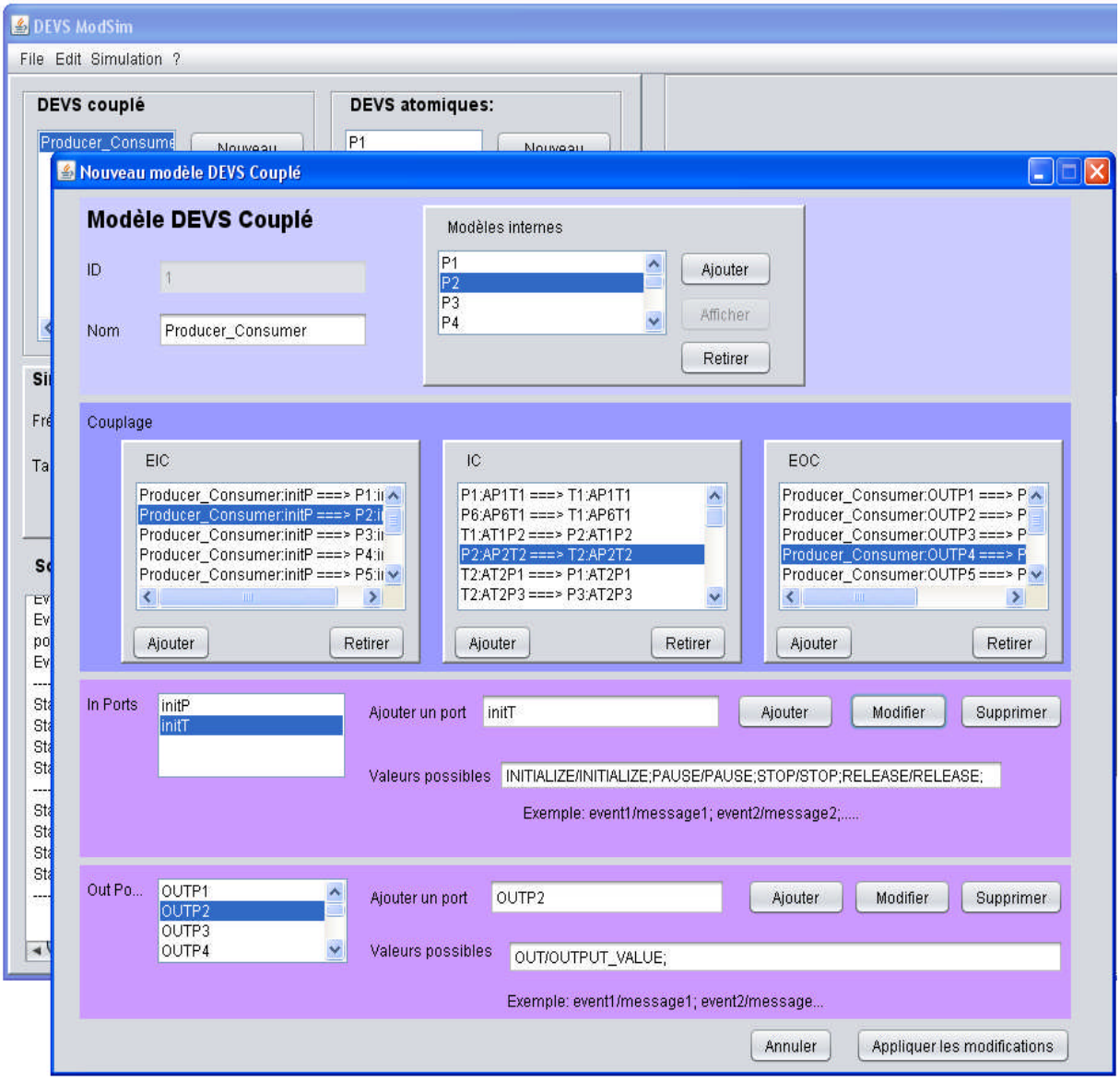

Fig.5.13 Interface d'affichage et d'édition d'un modèle DEVS couplé

Lafigure qui suit (5.14) illustre l'interface d'édition des couplages : EIC, IC et EOC. Alors que la figure 5.15 donne un appercu comment un fichier XML peut être chargé.

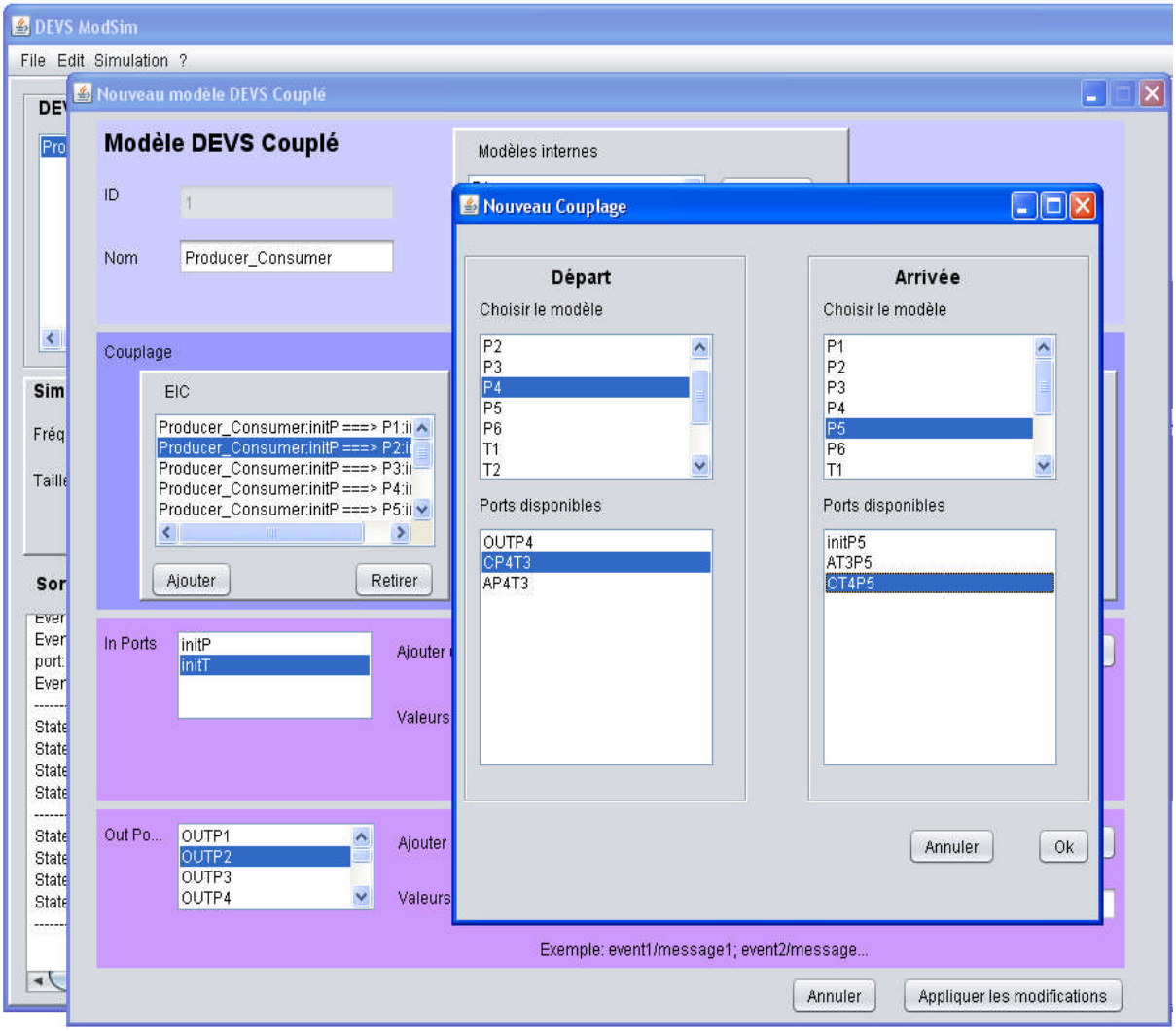

Fig.5.14 Interface d'affichage et d'édition d'un couplage DEVS

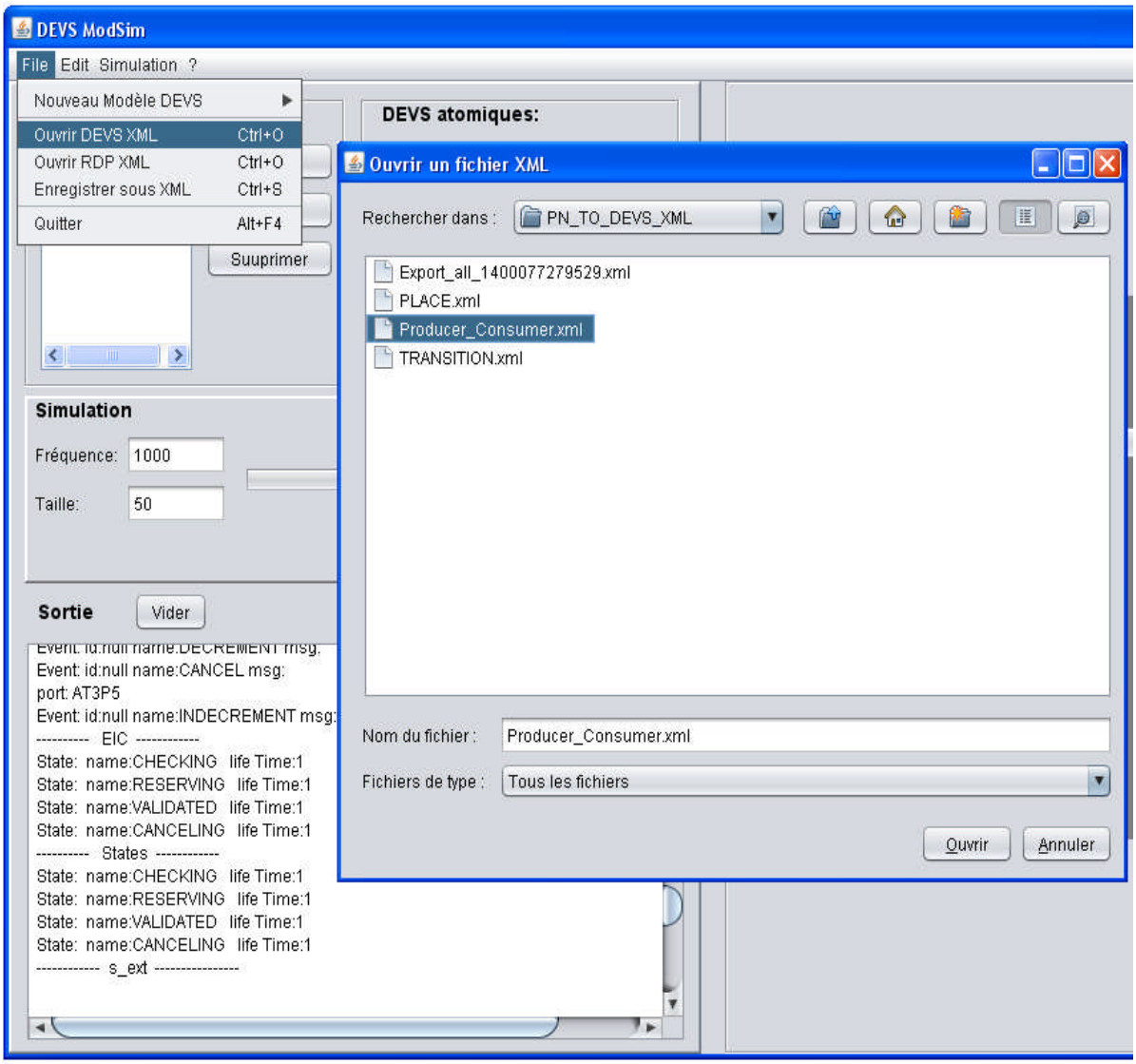

Fig.5.15 Interface de chargement d'un fichier XML

Les modèles DEVS enregistrés sous format XML ont une certaine structure bien déterminée. A noter qu'un fichier peut contenir plusieurs modèles qu'ils soient en relation ou indépendants les uns des autres. Eventuellement, il peut contenir même des modèle RDP. Nous présentons ci-dessous un aperçu d'un fichier XML des modèles DEVS (voir figure 5.16).

```
<?xml version="1.0" encoding="UTF-8" ?>
- <models>
  - <AtomicDEVS>
      <id>1</id>
      <name>P1</name>
    - <inPorts>
      - <port name="initP1">
          <event id="" name="INITIALIZE" msg="0" time="" />
        </port>
      - <port name="AT2P1">
          <event id="" name="INCREMENT" msg="0" time="" />
        </port>
      - <port name="CT1P1">
          <event id="" name="CHECK" msg="" time="" />
          <event id="" name="RESERVE" msg="" time="" />
          <event id="" name="DECREMENT" msg="" time="" />
          <event id="" name="CANCEL" msg="" time="" />
        </port>
      </inPorts>
    - <outPorts>
      - <port name="OUTP1" time="">
          <event id="" name="OUT" msg="OUTPUT_VALUE" />
        </port>
      - <port name="CP1T1" time="">
          <event id="" name="OK" msg="OK" />
          <event id="" name="FAIL" msg="FAIL" />
        </port>
      - <port name="AP1T1" time="">
          <event id="" name="FREE_TOKENS" msg="0" />
        </port>
      </outPorts>
    - <states>
        <state name="CHECKING" lifeTime="1" />
        <state name="RESERVING" lifeTime="1" />
        <state name="INCREMENTING" lifeTime="1" />
        <state name="DECREMENTING" lifeTime="1" />
      </states>
    - <s_ext>
        <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
        <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
        <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
        <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
        <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
        <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
        <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
        <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
        <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
        <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
        <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
        <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
        <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
      </s_ext>
    - <s_int>
        <line col0="CHECKING" col1="CHECKING" />
        <line col0="RESERVING" col1="CHECKING" />
        <line col0="INCREMENTING" col1="CHECKING" />
        <line col0="DECREMENTING" col1="CHECKING" />
      </s_int>
```

Fig.5.16 Aperçu d'un modèle DEVS sous format XML.

5.4. Conclusion

Dans ce chapitre nous avons présenté une application de modélisation et de simulation des modèles DEVS, à laquelle nous avons intégré l'algorithme de transformation des RDP en modèles DEVS. Ainsi, nous pouvons simuler au niveau de cette application des RdP sous la forme de modèles DEVS. Cette application a été développée en Java qui est un langage orienté objet supportant l'exécution multithreads. Notre application se caractérise par sa modularité. Ainsi des formalismes autres que les RdP et DEVS peuvent être intégrés grâce à un ensemble de classes qui jouent le rôle d'intégrateur de formalismes. Avec l'intégration des algorithmes de transformation on sera en mesure de réécrire de nouveaux modèles en DEVS. Ainsi, la simulation de ces modèles avec leur couplage avec d'autres modèles innés des formalismes hétérogènes sera une fonctionnalité de multi-modélisation que notre application assure.

Nous prévoyons de publier cette application sur le web pour la rendre disponible à la communauté des scientifiques qui s'intéressent au formalisme DEVS et à la multi-modélisation. De cette manière on aura ainsi, une large contribution qui va se répercuter positivement sur l'application.

Conclusion générale et perspectives

Ce mémoire rentre dans le cadre de la modélisation et de la simulation basées DEVS « Discret EVent System Specification » qui offre aujourd'hui une alternative très intéressante quant à la représentation des systèmes complexes où la notion de la multi-modélisation est souvent présente. Nous avons vu que le recours à des sous modèles basés sur différents formalismes n'est pas, en fait, une problématique, mais plutôt un besoin fondamental. De cette façon le modélisateur pourrait bénéficier de la force d'expression qu'offre chaque formalisme dans son domaine. Cependant, la problématique qui s'est posée et qui est celle du présent mémoire, porte sur la difficulté de faire coexister des modèles hétérogènes en un seul modèle. Parmi les techniques citées (voir [Vangheluwe ,2002]), nous avons donné plus d'intérêt à celle de la transformation entre modèles, du moment que notre contribution rentre dans ce cadre.

Au début de ce mémoire, nous avons commencé par une introduction concernant la multi- modélisation. Ensuite, nous avons spécialement mis l'accent sur la modélisation et la simulation des systèmes à évènements discrets. Ainsi nous avons détaillé le formalisme DEVS en tant que formalisme dédié à la modélisation de cette catégorie de systèmes. Dans le cadre de la multi-modélisation, nous avons proposé le formalisme DEVS en tant que formalisme universel [Touraille et al, 2010]. Et ce, en nous basant sur ses caractéristiques en matière de couplage et d'abstraction. Par la suite, nous avons vu un ensemble d'outils de modélisation basés sur le formalisme DEVS suivi par une synthèse générale dans laquelle nous avons montré les limites et les points forts des outils présentés. Ensuite nous avons entamé le concept de la transformation entre modèles de différents formalismes comme étant une réponse à la problématique de la coexistence de plusieurs modèles basés sur différents formalismes en un seul modèle qui englobe le tout. Donc à ce stade-là, nous avons rétréci la distance entre la problématique de la thèse portant sur la multi-modélisation et le souci de couplage entre différents modèles. Cela d'une part ; D'autre part, elle a servi au lecteur d'introduction à notre contribution qui consiste en une approche de transformation des modèles de réseaux de Petri en modèles DEVS. Nous avons défini cette approche pour

laquelle, nous avons spécifié les règles et les algorithmes de transformation. Nous avons, ainsi, soutenu notre approche par un exemple courant du « producteur-consommateur ».

Finalement nous avons présenté un outil de modélisation et de simulation de modèles DEVS auquel nous avons intégré l'algorithme de transformation. Nous avons développé cette application en Java tout en bénéficiant de la capacité de ce langage d'implémenter des programmes en modules. Ainsi, nous avons eu une application qui pourra supporter des éventuels formalismes et algorithmes de transformation. Par ailleurs, les modèles manipulés au niveau de cette application peuvent être lus ou stockés dans des fichiers sous format XML. Ainsi, la portabilité et la réutilisation des modèles sont bien respectées au niveau de cette application.

Nos perspectives portent sur la multi-modélisation et la problématique liées à l'interopérabilité entre les formalismes. Outre la transformation entre modèles basés sur différents formalismes, nos travaux vont aussi toucher la coopération des simulateurs. Autrement dit, comment faire marcher plusieurs simulateurs des plateformes différentes de manière conjointe. La co-simulation est souvent utilisée dans ce cadre.

Quant à notre application, nous envisageons de l'exploiter dans la modélisation et la simulation des systèmes complexes où les notions de distribution et d'interaction entre processus sont présentes. Nous prévoyons aussi de l'enrichir de telle sorte qu'elle soit un outil de vérification et de validation des réseaux de Petri ou d'autres formalismes éventuels.

Bibliographie

[Akplogan et al, 2009] M. Akplogan, F. Garcia, A. Joannon, R. Martin-Clouaire et G. Quesnel, *Un modèle DEVS d'agent intelligent: application à la conduite des systèmes de culture*, JFPDA, 2009.

[Baati, 2007] L. Baati, *Approche de modélisation DEVS à structure hiérarchique et dynamique*, Schedae, Republication N°15, Fascicule N°2, p. 61-70, 2007.

[Balci, 1988] O. Balci, *The implementation of four conceptual frameworks for simulation modeling in high-level languages*, Proceedings of the 20th conference on Winter simulation, ACM Press, P. 287–295, 1988.

[Barros, 1996] F. J. Barros, *Dynamic Structure Discret Event System Specification : Formalism, Abstract Simulators and Applications*, 13(1), p. 35–46, 1996.

[Barros, 1997] F. J. Barros, *Modeling Formalisms for Dynamic Structure Systems*. ACM Transactions on Modeling and Computer Simulation (TOMACS), 7, p. 501 – 515, October 1997.

[Bisgambiglia et al, 2012] P.A. Bisgambiglia, S. Mattei, M. Delhom et E. Vittori, *Towards Discrete Event Multi Agent Platform Specification*, Computation Tool: The third International Conference on Computational Logics, Algebras, Programming, Tools, and Benchmarking, p. 14-21, 2012.

[Bisgambiglia et al, 2013] P.A. Bisgambiglia et R. Franceschini, *Agent-Oriented Approach Based on Discrete Event Systems (WIP)*, University of Corsica, Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium, Article No. 27, 2013.

[Bézivin, 2002] J. Bézivin et X. Blanc, *Promesses et interrogations de l'approche MDA*, Journal Développeur Référence – Septembre 2002, 2002.

[Boissier et al, 2004] O. Boissier, S. Gitton et P. Glize, *Caractéristiques des systèmes et des applications, des Techniques Avancées*, Systèmes Multi-Agents (Série ARAGO), volume 29, pages 25–54.

[Bolduc et Vangheluwe, 2003] J.S. Bolduc et H. Vangheluwe, *Mapping odes to devs: Adaptive quantization*, Proceedings of the 2003 Summer Simulation MultiConference (SCSC'03), p. 401-407, Montréal, Canada, Juin 2003.

[Borland et al, 2003] S. Borland, et H. Vangheluwe, *Transforming statecharts to DEVS*, Summer Computer simulation Conference, Student Workshop - Society for Computer Simulation International, Montréal, Canada, 154–159, 2003.

[Boukelkoul et Redjimi, 2013] S. Boukelkoul et M. Redjimi, *mapping between petri nets and DEVS models*, ICTEES conference, Sousse, 2013.

[Bousquet et al, 1998] F. Bousquet, I. Bakam, H. Proton, et L. Cormas, *Common pool resources and multi-agent systems*, Lecture Notes in Artificial Intelligence 1416, 826-838, 1998.

[Capocchi et al., 2003] L. Capocchi, F. Bernardi et J. Santucci, *Transformation of VHDL descriptions into DEVS models for fault modeling*, Actes de la conférence 2003 IEEE International Conference on Systems, Man and Cybernetics, Washington, USA, 2003.

[Capocchi et al, 2011] L. Capocchi, J. F. Santucci, B. Poggi, et C. Nicolai, DEVSimPy: *A Collaborative Python Software for Modeling and Simulation of DEVS Systems*, Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on, Paris, p. 170 175, 2011.

[Cellier, 1986] F. Cellier, *Combined continuous/discrete simulation : applications, techniques and tools*, Proceedings of the 18th conference on Winter simulation, ACM Press, P. 24–33, 1986.

[Cellier, 1991a] F. Cellier, *Continuous System Modeling*. Springer-Verlag, 1991.

[Cellier, 1991b] F. Cellier, *Qualitative modeling and simulation: promise or illusion*, Proceedings of the 23rd conference on Winter simulation. IEEE Computer Society, P. 1086–1090, 1991.

[Chaib-Draa, 1999] B. Chaib-Draa, *Agent et Système Multi – Agents*, Université Laval, Quebec, Canada, November 1999.

[Chow et Barros, 1994] A.C. Chow et F.J. Barros, *Abstract simulator for the parallel DEVS formalism*, Proceedings of the Fifth Annual Conference on AI, Simulation and Planning in High Autonomy Systems, p. 157–163, 1994.

[Chow et Zeigler, 1994] A.C. Chow et B.P. Zeigler, *Parallel DEVS : a parallel, hierarchical, modular, modeling formalism*, Proceedings of the 26th conference on Winter simulation, Society for Computer Simulation International, P. 716–722, 1994.

[Cincom, 2003] Cincom, Cincomvisualworks, http://www.cincom.com/, 2003.

[Clavel, 1996] G. Clavel, N. Lopez et L. Veillon, *Programmer objets avec Small-talk*, Dunod. ISBN 2225851573, 1996.

[Czarnecki, 2003] K. Czarnecki et Simon Helsen, *Classification of Model Transformation Approaches*, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, 2003.

[Czarnecki, 2006] K. Czarnecki et S. Helsen, *Feature-based survey of model transformation approaches*, IBM Syst. J., 45(3): p. 621–645, ISSN 0018-8670, 2006.

[Davila et al, 2005] J. Davila, E. Gomez, K. Laffaille, K. Tucci, et M. Uzcategui, *MultiAgent Distributed Simulation with GALATEA*, p. 165 170, 2005.

[DEVS-Group, 2003] DEVS-Group, Groupe de standardisation de DEVS, http://www.sce.carleton.ca/faculty/wainer/standard/, 2003.

[Diaz, 2001] M. Diaz, *Les réseaux de Petri, modèles fondamentaux*. Hermes Sciences Publication. ISBN : 2746202506, 2001.

[Duboz et al, 2003] R. Duboz, F. Amblard, E. Ramat, G. Deffuant et P. Preux, *Utiliser les modèles individus-centrés comme laboratoires virtuels pour identifier les paramètres d'un modèle agrégé*, Conférence MOSIM'03, 2003.

[Duboz, 2004] R. Duboz, *Intégration de modèles hétérogènes pour la modélisation et la simulation de systèmes complexes : Application à la modélisation multi-échelles en écologie marine*, Thèse de Doctorat, Université du Littoral - Côte d'Opale, 2004.

[Dupuy, 1994] J.P. Dupuy, *Aux origines des sciences cognitives*, Seuil La découverte, 1994.

[Ecuyer et al, 1998] P. L'Ecuyer et P.Hellekalek, *Random Number Generators : Selection Criteria and Testing*, in Random and Quasi-Random Point Sets, Lectures Notes In Statistics, no. 138, Springer, p. 223–266, 1998.

[Ecuyer, 1998]. P. L'Ecuyer, *Efficient and Portable Combined Random Number Generators*, Communications of the ACM, 31, p. 742–749, 1998.

[Ferber, 1995] J. Ferber, *Les Systèmes Multi-Agents, vers une intelligence collective*, Inter-Éditions, 1995.

[Ferber, 1997] J. Ferber , *Les Systèmes Multi-Agents : Un Aperçu Général*, Revue Technique et Sciences Informatique, 16, 8, 1997.

[Filippi et al, 2001] J. Filippi, P. Bisgambiglia et M. Delhom, *Neuro-DEVS, an hybrid methodology to describe complex systems*, Actes de SCS ESS 2001 conference on simulation in industry, volume 1, p. 647-652, Marseille, France, 2001.

[Fishwick, 1989] P.A. Fishwick, *Process abstraction in simulation modeling*, Artificial intelligence, simulation & modeling, ISBN:0-471-60599-9, p. 93 – 131, 1989.

[Fishwick et Zeigler, 1992] P.A. Fishwick et B.P. Zeigler, *A multimodel methodology for qualitative model engineering*, ACM Transactions on Modeling and Computer Simulation (TOMACS), 2(1), 52–81, 1992.

[Fishwick, 1995] P.A. Fishwick, *Simulation Model Design and Execution: Simulation Model Design and Execution*, Building Digital Worlds, Prentice Hall, ISBN: 0130986097, 1995.

[Fishwick, 1997] P.A. Fishwick, *Computer simulation: growth through extension*, Transactions of the Society for Computer Simulation International, 14(1), 13–23, 1997.

[Fishwick, 1998] P.A. Fishwick, *An architectural design for digital objects*, Winter Simulation Conference, 1(1): 360-365, 1998.

[Hédi et Fauqué, 2001] S. Hédi et N. Fauqué, *La Tunisie antique : de Hannibal à Saint Augustin*, éd. Mengès, p. 21, 2001.

[Hill, 2000] D.R.C Hill, *Contribution à la modélisation des systèmes complexes : Application à la simulation d'écosystèmes*, Habilitation à diriger des recherches, Ecole Doctorale Sciences pour l'Ingénieur de Clermont-Ferrand, décembre 2000.

[Himmelspach and Uhrmacher, 2007] J. Himmelspach et A.M. Uhrmacher, *Plugin simulate*, Proceedings of the Spring Simulation Multiconf, IEEE Computer Society, p. 137–143, 2007.

[Himmelspach et al, 2008] J. Himmelspach, R. Ewald, et A.M. Uhrmacher, *A flexible and scalable experimentation layer*, Proceedings of the 2008 Winter Simulation Conference, ed. S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc, p. 827–835, 2008.

[Himmelspach et al, 2010] J. Himmelspach, M. Röhl et A. M. Uhrmacher, *COMPONENT-BASED MODELS AND SIMULATIONS FOR SUPPORTING VALID MULTI-AGENT SYSTEM SIMULATIONS*, Applied Artificial Intelligence: An International Journal, 24:5, p. 414-442, DOI: 10.1080/08839514.2010.481492, 2010.

[Hubbard, 1999] J. Hubbard, *Equations différentielles et systèmes dynamiques*. Vuibert. ISBN : 284225015X, 1999.

[Hunter et McLaughlin, 2012] J. Hunter et B. McLaughlin, JDOM Copyright (C) 2000-2012, 2012.

[Hymore, 1981] A. Hymore, *Applications of mathematical system theory to system design*, modeling and simulation, Proceedings of the 13th conference on Winter simulation, p. 209–219, 1981.

[IEEE, 2000] IEEE 1516-2000, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)- Framework and Rules*, Institute of Electrical and Electronics Engineers, 2000.

[Ilachinski, 2001] A. Ilachinski, *CELLULAR AUTOMATA, A Discrete Universe*, World Scientific Publishing Co, 2001, ISBN 981-02-4623-4, 2001.

[Ingalls, 2001] R.G. Ingalls, *Introduction to simulation*. Proceedings of the 33nd conference on Winter simulation. IEEE Computer Society, p. 7-16, 2001.

[Jacques et Wainer, 2002] C.J.D. Jacques et G.A. Wainer, *Using the CD++ DEVS Tookit to Develop Petri Nets*. Proceedings of the SCS Summer Computer Simulation Conference, San Diego, CA. U.S.A, 2002.

[James, 1990] F. James, *A review of pseudorandom number generators*, Comput. Phys. Comm. 60, no. 3, 329–344, 1990.

[Joannon, 1988] A. Joannon, *Modélisation et simulation d'agents en DEVS*. Application à la conduite de systèmes de culture. These SAD, janvier 2008.

[Klir, 1985] G.J. Klir, *Architecture of Systems Problem Solving*. Plenum Pub Corp, 1985.

[Kofman et al, 2000] E. Kofman, N. Giambiasi et S. Junco, *Fdevs : A general DEVSbased formalism for fault modeling and simulation*, Dans Actes de la conférence 2000 European Simulation Symposium, volume 1, p.77-82, Hamburg, Germany, 2000.

[Kuhn, 1972] T. S. Kuhn, *La structure des révolutions scientifiques*. Flammarion. Nouvelle traduction par Laure Meyer 1983, 1972.

[Lara et Vangheluwe, 2002] J. Lara et H. Vangheluwe, *Computer Aided Multi-Paradigm Modelling to Process Petri-Nets and Statecharts*. 2505, p.239-253, 2002.

[Lüscher, 1994] M. Lüscher, *A portable high-quality random number generator for lattice field theory simulations*, Comput. Phys. Comm.79, no. 1, p. 100–110, 1994.

[Mattei et al, 2012] S. Mattei, P.A. Bisgambiglia, M. Delhom et E. Vittori, *Twards Discret Event Multi Agent Platform Specification*, IARIA, ISBN: 978-1-61208-222-6, p. 14-21, 2012.

[Matsumoto et Nishimura, 1998] Matsumoto, T. Nishimura, *dimensionally equidistributed uniform pseudorandom number generator*", Mersenne Twister : A 623-ACM Transactions on Modeling and Computer Simulation, 8(1), 1998.

[Millischer, 2000] L. Millischer, *Modélisation individu centrée des comportements de recherche des navires de pêche,* Thèse de Doctorat, Ecole Nationale Supérieure Agronomique de Rennes, juin 2000.

[Miller et al, 2004] R.L. Miller, J.P. Perlwitz et I. Tegen, *Feedback upon dust emission by dust radiative forcing through the planetary boundary layer*. J. Geophys. Res., 109, D24209, doi:10.1029/2004JD004912, 2004.

[Mittal et al, 2007] S. Mittal, J.L. Risco-Martín et B. P. Zeigler, *DEVS-based simulation web services for net-centric T&E*, Proceedings of the 2007 Summer Computer Simulation Conference, San Diego, CA, USA, 2007.

[Nutaro, 2012] J. Nutaro, *A Discrete EVent system Simulator*, Chap3, Juin 2012.

[OMG, 2003] *Updated joint initial submission against the action semantics for UML RFP*, available at http://cgi.omg.org/cgi-bin/doc?ad/00-08-03, 2003.

[OMG, 2003a] *Object Management Group: OMG Unified Modeling Language pecification*, Version 1.5, mars 2003.

[OMG, 2004] *Object Management Group (OMG), Model Driven Architecture (MDA)*, http://www.omg.org/mda, 2004.

[Oracle, 2012] Java Platform Standard Edition 7 Documentation, Copyright © 1993, 2012.

[Pavé, 1994] A. Pavé, *Modélisation en biologie et en écologie*, Aléas, 1994.

[Peterson, 1977] J. L. Peterson, *Petri nets*, Computing Surveys, p. 223–252, 1977.

[Praehofer et al, 1993] H. Praehofer, F. Auering et G. Reisinger, *An Environment for DEVS-Based Multi-Formalism Simulation in Common Lisp/CLOS*, Discrete Event Dynamic Systems : Theory and Applications, 3(2/3), 119–149. 1993

[Quesnel, 2006] G. Quesnel, *Approche formelle et opérationnelle de la multimodélisation et de la simulation des systèmes complexes*, Laboratoire d'Informatique du Littoral, 2006.

[Quesnel et al, 2010] G. Quesnel, E. Ramat, J. C. Soulié, D. Duvivier et R. Duboz, *Virtual Laboratory Environment : un environnement de multimodélisation et de simulation de systèmes complexes*, Studia Informatica Universalis, p. 205 -234, 2010.

[Rahman, et al, 2003] J. Rahman, S. Cuddy et F. Watson, *Tarsier and Icms : Two approaches to framework development.* International Congress on Modelling and Simulation MODSIM, 1 : p. 1625-1630, 2003.

[Ramat, 2003] E. Ramat, *Contributions à la modélisation et à la simulation des systèmes complexes*, Habilitation à diriger des recherche, 2003.

[Redjimi et Boukelkoul, 2013] M. Redjimi et S. Boukelkoul, *Algorithmic tools for the transformation of Petri Nets to DEVS*, Informatica 37, p.411-418, ISSN 0350-5596, 2013.

[Roques, 2000] P. Roques and F. Vallée, *UML en action : De l'analyse des besoins à la conception en Java*, Eyrolles, 2000.

[Rozenblit et Zeigler, 1993] J.W. Rozenblit et B.P. Zeigler, *Representing and constructing system specifications using the system entity structure concepts*. P. 604–61, Proceedings of the 25th conference on Winter simulation, ACM Press, 1993.

[Sarjoughian et Zeigler, 1998] H. Sarjoughian et B. P. Zeigler, *Devsjava : Basis for a DEVSbased collaborative ms environment*. SCS International Conference on Web-Based Modeling and Simulation, San Diego, CA, volume 5, p. 29-36, 1998.

[Sarjoughian et Zeigler,1999] H. Sarjoughian et B. P. Zeigler, *Collaborative modeling: The missing piece of distributed simulation*. Actes de la conférence SPIE, EnablingTechnology for Simulation Science III, Orlando, FL, 3696 : p. 126-135, 1999.

[Schattenberg et Uhrmacher, 2001] B. Schattenberg et A. Uhrmacher, *Planning agents in James*. Actes de la conférence IEEE transacions on modeling and computer simulation, 89(2) : p. 158-173, 2001.

[Schmidt, 2006] D. C. Schmidt, *Model-Driven Engineering Guest Editor's Introduction* , IEEE Computer, Vol. 39, No. 2, 25-31, 2006.

[Servat, 2000] D. Servat, *Modélisation de dynamiques de flux par agents. Application aux processus de ruissellement, infiltration et érosion*, Thèse de Doctorat, Université de Paris VI, Novembre 2000.

[Shannon, 1976] R.E. Shannon, *Simulation modeling and methodology*, Proceedings of the 76 Bicentennial conference on Winter simulation, p. 9–15, 1976.

[Shannon, 1998] R.E. Shannon, *Introduction to the art and science of simulation*, Proceedings of the 30th conference on Winter simulation, p. 7–14, IEEE Computer Society Press, 1998.

[SWARM, 2002] SWARM (2002). Swarm development group, http://www.swarm.org/.

[Touil, 2007] Touil A. *rapport PFE: Spécification et implantation d'une chaîne de transformation*, IRISA : Institut de Recherche en Informatique et Systèmes Aléatoires, Septembre 2007.

[Touraille et al, 2010] L. Touraille, M. K. Traoré, et D. R. C. Hill, *SimStudio: une Infrastructure pour la modélisation, la simulation et l'analyse de systèmes dynamiques complexes*, Research Report LIMOS/RR-10-13, 2010.

[Uhrmacher et Arnold, 1994] A.M. Uhrmacher et R. Arnold, *Distributed and maintaining knowledge: Agents in variable structure environments*, p. 178-184, IEEE, Florida, Gainsville, 1994.

[Uhrmacher, 2001] A.M. Uhrmacher, *Dynamics Structure in Modeling and Simulation - A Reflective Approach*, ACM Transaction on Modeling and Simulation, 11(2): p. 206–232, 2001.

[Vangheluwe, 2000] H. Vangheluwe, *DEVS as a common denominator for multiformalism hybrid systems modelling*, Conférence IEEE International Symposium on Computer-Aided Control System Design, Alaska, 1(1) :129-134, 2000.

[Vangheluwe et Bolduc, 2002] H. L. Vangheluwe et J. S. Bolduc, *Pydevs*, McGill's, 2002.

[Vangheluwe et al, 2002] H. Vangheluwe, J. Lara, et P. Mosterman, *An introduction to multi-paradigm modelling and simulation*. AIS 2002 Conference, Lisboa, Portugal, volume 1, p. 9-20,2002.

[Von Neumann, 1966] J. Von Neumann et A. W. Burks, *Theory of Self-Reproducing Automata*, University of Illinois Press, 1966.

[Wainer, 2003] G. Wainer, *Liste des outils basés sur le formalisme DEVS*, http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm, 2003.

[Woodbury et al, 2002] P. Woodbury, R. Beloin, D. Swaney, B. Gollands, et D. Weinstein, *Using the ECLPSS software environment to build a spatially explicit component-based model of ozone effects on forest ecosystems*. Ecological modelling, 150(3) : p. 211-238,2002.

[Zeigler, 1972] B.P. Zeigler, *Toward a Formal Theory of Modeling and Simulation: Structure Preserving Morphisms*. Journal of the ACM (JACM), 19(4), p. 742–764, 1972.

[Zeigler, 1976] B. P. Zeigler, *Theory of Modeling and Simulation*, Wiley InterScience, 1976.

[Zeigler, 1984] B. P. Zeigler, *Theory of Modeling and Simulation*, Krieger Publishing Company, 2nd Edition, ASIN: 0471981524, 1984.

[Zeigler, 1984b] B. P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, 1984.

[Zeigler et Sarjoughian, 1998] B. P. Zeigler, S. Hall et H. Sarjoughian, *Exploiting HLA and DEVS to promote interoperability and reuse in lockheed'scorporate environment*. SIMULATION, Special Issue on The High Level Architecture, 74(4) : p. 288-295, 1999.

[Zeigler et al, 1999] B. P. Zeigler, S. HALL et H. Sarjoughian, *Exploiting HLA and DEVS to promote interoperability and reuse in lockheed's corporate environment*. Simulation, Special Issue on The High Level Architecture, 74(4): p. 288–295, 1999.

[Zeigler et al, 2000] B. P. Zeigler, T. G. Kim et H. Praehofer, Theory of Modeling and Simulation. Academic Press, ISBN 0127784551, 2000.

Travaux de publication

Publications

1- Redjimi Mohamed et Boukelkoul Sofiane,
"Algorithmic tools for the transformation of Petri Nets to DEVS",
Informatica 37, p.411-418, ISSN 0350-5596, 2013.

2- Seddari Nour Eddine, Redjimi Mohammed et Boukelkoul Sofiane,
"Using of DEVS and MAS Tools for the Modeling and the Simulation of an Industrial Steam Generator" (Article accepté à paraitre),
http://cit.srce.unizg.hr

Communications

1- Boukelkoul Sofiane et Redjimi Mohamed,
"*Mapping between Petri nets and DEVS models"*,
ICTEES conference, Sousse, 2013.

2- Boukelkoul Sofiane et Redjimi Mohamed,
Formal Approach for Petri Nets to DEVS Models Transformation,
ICIST'13 conference, Tangier, 22-24 Mars 2013.

3- Boukelkoul Sofiane et Redjimi Mohamed,
"*Approche pour la transformation des réseaux de Petri en modèles DEVS"*,
ICSIP'13, Guelma, 12-14 Mai 2013.

Annexe

Fichiers XML relatifs à la transformation du modèle producteur-consommateur (Chapitre 4)

```
<?xml version="1.0" encoding="UTF -8" ?>
- <models>
  - <PN>
      <id>PN_0</id>
      <name>PN_PROD_CONS</name>
    - <M0>
        <place mark="1" />
        <place mark="0" />
        <place mark="0" />
        <place mark="1" />
        <place mark="0" />
        <place mark="7" />
      </M0>
    - <P>
        <place >P1</place >
        <place >P2</place >
        <place >P3</place >
        <place >P4</place >
        <place >P5</place >
        <place >P6</place >
      </P>
    - <T>
        <transition >T1</transition>
        <transition >T2</transition>
        <transition >T3</transition>
        <transition >T4</transition>
      </T>
    - <pre>
        <line T0="1" T1="0" T2="0" T3="0" />
        <line T0="0" T1="1" T2="0" T3="0" />
        <line T0="0" T1="0" T2="1" T3="0" />
        <line T0="0" T1="0" T2="1" T3="0" />
        <line T0="0" T1="0" T2="0" T3="1" />
        <line T0="1" T1="0" T2="0" T3="0" />
      </pre>
    - <post>
        <line T0="0" T1="1" T2="0" T3="0" />
        <line T0="1" T1="0" T2="0" T3="0" />
        <line T0="0" T1="1" T2="0" T3="0" />
        <line T0="0" T1="0" T2="0" T3="1" />
        <line T0="0" T1="0" T2="1" T3="0" />
        <line T0="0" T1="0" T2="0" T3="1" />
      </post>
    </PN>
</models>
```

Fig A.1Le RDP Producteur-Consommateur sous XML

```
<?xml version="1.0" encoding="UTF-8" ?>
- <models>
  - <AtomicDEVS>
      <id>1</id>
      <name>P1</name>
    + <inPorts>
    + <outPorts>
    + <states>
    + <s_ext>
    + <s_int>
      <s_con />
    + <lambda>
    </AtomicDEVS>
  + <AtomicDEVS>
  + <AtomicDEVS>
  + <AtomicDEVS>
  + <AtomicDEVS>
  + <AtomicDEVS>
  - <AtomicDEVS>
      <id>7</id>
      <name>T1</name>
    + <inPorts>
    + <outPorts>
    + <states>
    + <s_ext>
    + <s_int>
      <s_con />
    + <lambda>
    </AtomicDEVS>
  + <AtomicDEVS>
  + <AtomicDEVS>
  + <AtomicDEVS>
  - <CoupledDEVS>
      <id>id_1</id>
      <name>Producer_Consumer</name>
    + <inPorts>
    + <outPorts>
    + <internalModels>
    + <EIC>
    + <IC>
    + <EOC>
      <states />
      <s_ext />
      <s_int />
      <s_con />
      <lambda />
    </CoupledDEVS>
  </models>
```

Fig A.2La structure générale des modèles DEVS sous XML

```
- <AtomicDEVS>
    <id>1</id>
    <name>P1</name>
  - <inPorts>
    - <port name="initP1">
        <event id="" name="INITIALIZE" msg="0" time="" />
      </port>
    - <port name="AT2P1">
        <event id="" name="INCREMENT" msg="0" time="" />
      </port>
    - <port name="CT1P1">
        <event id="" name="CHECK" msg="" time="" />
        <event id="" name="RESERVE" msg="" time="" />
        <event id="" name="DECREMENT" msg="" time="" />
        <event id="" name="CANCEL" msg="" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP1" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CP1T1" time="">
        <event id="" name="OK" msg="OK" />
        <event id="" name="FAIL" msg="FAIL" />
      </port>
    - <port name="AP1T1" time="">
        <event id="" name="FREE_TOKENS" msg="0" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="INCREMENTING" lifeTime="1" />
      <state name="DECREMENTING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
      <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
      <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
      <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
      <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="CHECKING" />
      <line col0="INCREMENTING" col1="CHECKING" />
      <line col0="DECREMENTING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="" col2="" col3="FREE_TOKENS" />
      <line col0="RESERVING" col1="" col2="OK/FAIL" col3="" />
      <line col0="INCREMENTING" col1="OUT" col2="" col3="" />
      <line col0="DECREMENTING" col1="OUT" col2="" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.3 Modèle DEVS atomique correspondant à la place P1

```
- <AtomicDEVS>
    <id>2</id>
    <name>P2</name>
  - <inPorts>
    - <port name="initP2">
        <event id="" name="INITIALIZE" msg="0" time="" />
      </port>
    - <port name="AT1P2">
        <event id="" name="INCREMENT" msg="0" time="" />
      </port>
    - <port name="CT2P2">
        <event id="" name="CHECK" msg="" time="" />
        <event id="" name="RESERVE" msg="" time="" />
        <event id="" name="DECREMENT" msg="" time="" />
        <event id="" name="CANCEL" msg="" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP2" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CP2T2" time="">
        <event id="" name="OK" msg="OK" />
        <event id="" name="FAIL" msg="FAIL" />
      </port>
    - <port name="AP2T2" time="">
        <event id="" name="FREE_TOKENS" msg="0" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="INCREMENTING" lifeTime="1" />
      <state name="DECREMENTING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
      <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
      <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
      <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
      <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="CHECKING" />
      <line col0="INCREMENTING" col1="CHECKING" />
      <line col0="DECREMENTING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="" col2="" col3="FREE_TOKENS" />
      <line col0="RESERVING" col1="" col2="OK/FAIL" col3="" />
      <line col0="INCREMENTING" col1="OUT" col2="" col3="" />
      <line col0="DECREMENTING" col1="OUT" col2="" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.4 Modèle DEVS atomique correspondant à la place P2

```
- <AtomicDEVS>
    <id>3</id>
    <name>P3</name>
  - <inPorts>
    - <port name="initP3">
        <event id="" name="INITIALIZE" msg="0" time="" />
      </port>
    - <port name="AT2P3">
        <event id="" name="INCREMENT" msg="0" time="" />
      </port>
    - <port name="CT3P3">
        <event id="" name="CHECK" msg="" time="" />
        <event id="" name="RESERVE" msg="" time="" />
        <event id="" name="DECREMENT" msg="" time="" />
        <event id="" name="CANCEL" msg="" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP3" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CP3T3" time="">
        <event id="" name="OK" msg="OK" />
        <event id="" name="FAIL" msg="FAIL" />
      </port>
    - <port name="AP3T3" time="">
        <event id="" name="FREE_TOKENS" msg="0" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="INCREMENTING" lifeTime="1" />
      <state name="DECREMENTING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
      <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
      <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
      <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
      <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="CHECKING" />
      <line col0="INCREMENTING" col1="CHECKING" />
      <line col0="DECREMENTING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="" col2="" col3="FREE_TOKENS" />
      <line col0="RESERVING" col1="" col2="OK/FAIL" col3="" />
      <line col0="INCREMENTING" col1="OUT" col2="" col3="" />
      <line col0="DECREMENTING" col1="OUT" col2="" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.5 Modèle DEVS atomique correspondant à la place P3

```
- <AtomicDEVS>
    <id>4</id>
    <name>P4</name>
  - <inPorts>
    - <port name="initP4">
        <event id="" name="INITIALIZE" msg="0" time="" />
      </port>
    - <port name="AT4P4">
        <event id="" name="INCREMENT" msg="0" time="" />
      </port>
    - <port name="CT3P4">
        <event id="" name="CHECK" msg="" time="" />
        <event id="" name="RESERVE" msg="" time="" />
        <event id="" name="DECREMENT" msg="" time="" />
        <event id="" name="CANCEL" msg="" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP4" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CP4T3" time="">
        <event id="" name="OK" msg="OK" />
        <event id="" name="FAIL" msg="FAIL" />
      </port>
    - <port name="AP4T3" time="">
        <event id="" name="FREE_TOKENS" msg="0" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="INCREMENTING" lifeTime="1" />
      <state name="DECREMENTING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
      <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
      <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
      <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
      <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="CHECKING" />
      <line col0="INCREMENTING" col1="CHECKING" />
      <line col0="DECREMENTING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="" col2="" col3="FREE_TOKENS" />
      <line col0="RESERVING" col1="" col2="OK/FAIL" col3="" />
      <line col0="INCREMENTING" col1="OUT" col2="" col3="" />
      <line col0="DECREMENTING" col1="OUT" col2="" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.6 Modèle DEVS atomique correspondant à la place P4

```
- <AtomicDEVS>
    <id>5</id>
    <name>P5</name>
  - <inPorts>
    - <port name="initP5">
        <event id="" name="INITIALIZE" msg="0" time="" />
      </port>
    - <port name="AT3P5">
        <event id="" name="INCREMENT" msg="0" time="" />
      </port>
    - <port name="CT4P5">
        <event id="" name="CHECK" msg="" time="" />
        <event id="" name="RESERVE" msg="" time="" />
        <event id="" name="DECREMENT" msg="" time="" />
        <event id="" name="CANCEL" msg="" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP5" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CP5T4" time="">
        <event id="" name="OK" msg="OK" />
        <event id="" name="FAIL" msg="FAIL" />
      </port>
    - <port name="AP5T4" time="">
        <event id="" name="FREE_TOKENS" msg="0" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="INCREMENTING" lifeTime="1" />
      <state name="DECREMENTING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
      <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
      <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
      <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
      <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="CHECKING" />
      <line col0="INCREMENTING" col1="CHECKING" />
      <line col0="DECREMENTING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="" col2="" col3="FREE_TOKENS" />
      <line col0="RESERVING" col1="" col2="OK/FAIL" col3="" />
      <line col0="INCREMENTING" col1="OUT" col2="" col3="" />
      <line col0="DECREMENTING" col1="OUT" col2="" col3="" />
    </lambda>
  </AtomicDEVS>
```


Fig A.7 Modèle DEVS atomique correspondant à la place P5

```
- <AtomicDEVS>
    <id>6</id>
    <name>P6</name>
  - <inPorts>
    - <port name="initP6">
        <event id="" name="INITIALIZE" msg="0" time="" />
      </port>
    - <port name="AT4P6">
        <event id="" name="INCREMENT" msg="0" time="" />
      </port>
    - <port name="CT1P6">
        <event id="" name="CHECK" msg="" time="" />
        <event id="" name="RESERVE" msg="" time="" />
        <event id="" name="DECREMENT" msg="" time="" />
        <event id="" name="CANCEL" msg="" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP6" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CP6T1" time="">
        <event id="" name="OK" msg="OK" />
        <event id="" name="FAIL" msg="FAIL" />
      </port>
    - <port name="AP6T1" time="">
        <event id="" name="FREE_TOKENS" msg="0" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="INCREMENTING" lifeTime="1" />
      <state name="DECREMENTING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="ALL" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="CHECKING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="CHECKING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="CHECKING" col4="INCREMENTING" />
      <line col0="" col1="" col2="DECREMENT" col3="RESERVING" col4="DECREMENTING" />
      <line col0="" col1="" col2="INCREMENT" col3="RESERVING" col4="INCREMENTING" />
      <line col0="" col1="" col2="CANCEL" col3="RESERVING" col4="CHECKING" />
      <line col0="" col1="" col2="CHECK" col3="INCREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="INCREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="DECREMENT" col3="INCREMENTING" col4="DECREMENTING" />
      <line col0="" col1="" col2="CHECK" col3="DECREMENTING" col4="CHECKING" />
      <line col0="" col1="" col2="RESERVE" col3="DECREMENTING" col4="RESERVING" />
      <line col0="" col1="" col2="INCREMENT" col3="DECREMENTING" col4="INCREMENTING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="CHECKING" />
      <line col0="INCREMENTING" col1="CHECKING" />
      <line col0="DECREMENTING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="" col2="" col3="FREE_TOKENS" />
      <line col0="RESERVING" col1="" col2="OK/FAIL" col3="" />
      <line col0="INCREMENTING" col1="OUT" col2="" col3="" />
      <line col0="DECREMENTING" col1="OUT" col2="" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.8 Modèle DEVS atomique correspondant à la place P6

Fig A.9 Modèle DEVS atomique correspondant à la transition T1

```
- <AtomicDEVS>
    <id>8</id>
    <name>T2</name>
  - <inPorts>
    - <port name="initT2">
        <event id="" name="INITIALIZE" msg="" time="" />
        <event id="" name="PAUSE" msg="" time="" />
        <event id="" name="STOP" msg="" time="" />
        <event id="" name="RELEASE" msg="" time="" />
      </port>
    - <port name="CP2T2">
        <event id="" name="OK" msg="" time="" />
        <event id="" name="FAIL" msg="" time="" />
      </port>
    - <port name="AP2T2">
        <event id="" name="FREE_TOKENS" msg="0" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTT2" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CT2" time="">
        <event id="" name="CHECK" msg="" />
        <event id="" name="RESERVE" msg="" />
        <event id="" name="DECREMENT" msg="" />
        <event id="" name="CANCEL" msg="" />
      </port>
    - <port name="AT2P1" time="">
        <event id="" name="INDECREMENT" msg="" />
      </port>
    - <port name="AT2P3" time="">
        <event id="" name="INDECREMENT" msg="" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="VALIDATED" lifeTime="1" />
      <state name="CANCELING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="" col4="CHECKING" />
      <line col0="PAUSE" col1="" col2="" col3="" col4="PAUSED" />
      <line col0="RELEASE" col1="" col2="" col3="" col4="CHECKING" />
      <line col0="STOP" col1="" col2="" col3="" col4="STOPED" />
      <line col0="" col1="" col2="FREE_TOKENS" col3="RESERVING" col4="RESERVING" />
      <line col0="" col1="OK" col2="" col3="RESERVING" col4="RESERVING/VALIDATED" />
      <line col0="" col1="FAIL" col2="" col3="RESERVING" col4="CANCELING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="RESERVING" />
      <line col0="VALIDATED" col1="CHECKING" />
      <line col0="CNCELING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="OUT" col2="CHECK" col3="" />
      <line col0="RESERVING" col1="OUT" col2="RESERVE" col3="" />
      <line col0="VALIDATED" col1="OUT" col2="DECREMENT" col3="INCREMENT" />
      <line col0="CANCELING" col1="OUT" col2="CANCEL" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.10 Modèle DEVS atomique correspondant à la transition T2

```
- <AtomicDEVS>
    <id>9</id>
    <name>T3</name>
  - <inPorts>
    - <port name="initT3">
        <event id="" name="INITIALIZE" msg="" time="" />
        <event id="" name="PAUSE" msg="" time="" />
        <event id="" name="STOP" msg="" time="" />
        <event id="" name="RELEASE" msg="" time="" />
      </port>
    - <port name="CP3T3">
        <event id="" name="OK" msg="" time="" />
        <event id="" name="FAIL" msg="" time="" />
      </port>
    - <port name="CP4T3">
        <event id="" name="OK" msg="" time="" />
        <event id="" name="FAIL" msg="" time="" />
      </port>
    - <port name="AP3T3">
        <event id="" name="FREE_TOKENS" msg="0" time="" />
      </port>
    + <port name="AP4T3">
    </inPorts>
  - <outPorts>
    + <port name="OUTT3" time="">
    - <port name="CT1" time="">
        <event id="" name="CHECK" msg="" />
        <event id="" name="RESERVE" msg="" />
        <event id="" name="DECREMENT" msg="" />
        <event id="" name="CANCEL" msg="" />
      </port>
    - <port name="AT3P5" time="">
        <event id="" name="INDECREMENT" msg="" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="VALIDATED" lifeTime="1" />
      <state name="CANCELING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="" col4="" col5="" col6="CHECKING" />
      <line col0="PAUSE" col1="" col2="" col3="" col4="" col5="" col6="PAUSED" />
      <line col0="RELEASE" col1="" col2="" col3="" col4="" col5="" col6="CHECKING" />
      <line col0="STOP" col1="" col2="" col3="" col4="" col5="" col6="STOPED" />
      <line col0="" col1="" col2="" col3="FREE_TOKENS" col4="FREE_TOKENS" col5="RESERVING" col6="RESERVIN
      <line col0="" col1="OK" col2="OK" col3="" col4="" col5="RESERVING" col6="RESERVING/VALIDATED" />
      <line col0="" col1="FAIL" col2="FAIL" col3="" col4="" col5="RESERVING" col6="CANCELING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="RESERVING" />
      <line col0="VALIDATED" col1="CHECKING" />
      <line col0="CNCELING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="OUT" col2="CHECK" col3="" />
      <line col0="RESERVING" col1="OUT" col2="RESERVE" col3="" />
      <line col0="VALIDATED" col1="OUT" col2="DECREMENT" col3="INCREMENT" />
      <line col0="CANCELING" col1="OUT" col2="CANCEL" col3="" />
    </lambda>
  </AtomicDEVS>
```


Fig A.11 Modèle DEVS atomique correspondant à la transition T3

```
- <AtomicDEVS>
    <id>10</id>
    <name>T4</name>
  - <inPorts>
    - <port name="initT4">
        <event id="" name="INITIALIZE" msg="" time="" />
        <event id="" name="PAUSE" msg="" time="" />
        <event id="" name="STOP" msg="" time="" />
        <event id="" name="RELEASE" msg="" time="" />
      </port>
    - <port name="CP5T4">
        <event id="" name="OK" msg="" time="" />
        <event id="" name="FAIL" msg="" time="" />
      </port>
    - <port name="AP5T4">
        <event id="" name="FREE_TOKENS" msg="0" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTT4" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="CT4" time="">
        <event id="" name="CHECK" msg="" />
        <event id="" name="RESERVE" msg="" />
        <event id="" name="DECREMENT" msg="" />
        <event id="" name="CANCEL" msg="" />
      </port>
    - <port name="AT4P6" time="">
        <event id="" name="INDECREMENT" msg="" />
      </port>
    - <port name="AT4P4" time="">
        <event id="" name="INDECREMENT" msg="" />
      </port>
    </outPorts>
  - <states>
      <state name="CHECKING" lifeTime="1" />
      <state name="RESERVING" lifeTime="1" />
      <state name="VALIDATED" lifeTime="1" />
      <state name="CANCELING" lifeTime="1" />
    </states>
  - <s_ext>
      <line col0="INITIALIZE" col1="" col2="" col3="" col4="CHECKING" />
      <line col0="PAUSE" col1="" col2="" col3="" col4="PAUSED" />
      <line col0="RELEASE" col1="" col2="" col3="" col4="CHECKING" />
      <line col0="STOP" col1="" col2="" col3="" col4="STOPED" />
      <line col0="" col1="" col2="FREE_TOKENS" col3="RESERVING" col4="RESERVING" />
      <line col0="" col1="OK" col2="" col3="RESERVING" col4="RESERVING/VALIDATED" />
      <line col0="" col1="FAIL" col2="" col3="RESERVING" col4="CANCELING" />
    </s_ext>
  - <s_int>
      <line col0="CHECKING" col1="CHECKING" />
      <line col0="RESERVING" col1="RESERVING" />
      <line col0="VALIDATED" col1="CHECKING" />
      <line col0="CNCELING" col1="CHECKING" />
    </s_int>
    <s_con />
  - <lambda>
      <line col0="CHECKING" col1="OUT" col2="CHECK" col3="" />
      <line col0="RESERVING" col1="OUT" col2="RESERVE" col3="" />
      <line col0="VALIDATED" col1="OUT" col2="DECREMENT" col3="INCREMENT" />
      <line col0="CANCELING" col1="OUT" col2="CANCEL" col3="" />
    </lambda>
  </AtomicDEVS>
```

Fig A.12 Modèle DEVS atomique correspondant à la transition T4

```
- <CoupledDEVS>
    <id>id_1</id>
    <name>Producer_Consumer</name>
  - <inPorts>
    - <port name="initP">
        <event id="" name="INITIALIZE" msg="INITIALIZE" time="" />
      </port>
    - <port name="initT">
        <event id="" name="INITIALIZE" msg="INITIALIZE" time="" />
        <event id="" name="PAUSE" msg="PAUSE" time="" />
        <event id="" name="STOP" msg="STOP" time="" />
        <event id="" name="RELEASE" msg="RELEASE" time="" />
      </port>
    </inPorts>
  - <outPorts>
    - <port name="OUTP1" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUTP2" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUTP3" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUTP4" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUTP5" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUTP6" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUT1" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUT2" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUT3" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    - <port name="OUT4" time="">
        <event id="" name="OUT" msg="OUTPUT_VALUE" />
      </port>
    </outPorts>
  - <internalModels>
      <model name="P1" />
      <model name="P2" />
      <model name="P3" />
      <model name="P4" />
      <model name="P5" />
      <model name="P6" />
      <model name="T1" />
      <model name="T2" />
      <model name="T3" />
      <model name="T4" />
    </internalModels>
  - <EIC>
      <coupling fromModel="Producer_Consumer" from="initP" toModel="P1" to="initP1" />
      <coupling fromModel="Producer_Consumer" from="initP" toModel="P2" to="initP2" />
      <coupling fromModel="Producer_Consumer" from="initP" toModel="P3" to="initP3" />
      <coupling fromModel="Producer_Consumer" from="initP" toModel="P4" to="initP4" />
      <coupling fromModel="Producer_Consumer" from="initP" toModel="P5" to="initP5" />
```

Fig A.13 Modèle DEVS couplé correspondant au RDP Producteur_Consommateur (partie 1).

```
  </internalModels>
- <EIC>
    <coupling fromModel="Producer_Consumer" from="initP" toModel="P1" to="initP1" />
    <coupling fromModel="Producer_Consumer" from="initP" toModel="P2" to="initP2" />
    <coupling fromModel="Producer_Consumer" from="initP" toModel="P3" to="initP3" />
    <coupling fromModel="Producer_Consumer" from="initP" toModel="P4" to="initP4" />
    <coupling fromModel="Producer_Consumer" from="initP" toModel="P5" to="initP5" />
    <coupling fromModel="Producer_Consumer" from="initP" toModel="P6" to="initP6" />
    <coupling fromModel="Producer_Consumer" from="initT" toModel="T1" to="initT1" />
    <coupling fromModel="Producer_Consumer" from="initT" toModel="T2" to="initT2" />
    <coupling fromModel="Producer_Consumer" from="initT" toModel="T3" to="initT3" />
    <coupling fromModel="Producer_Consumer" from="initT" toModel="T4" to="initT4" />
  </EIC>
- <IC>
    <coupling fromModel="P1" from="AP1T1" toModel="T1" to="AP1T1" />
    <coupling fromModel="P6" from="AP6T1" toModel="T1" to="AP6T1" />
    <coupling fromModel="T1" from="AT1P2" toModel="P2" to="AT1P2" />
    <coupling fromModel="P2" from="AP2T2" toModel="T2" to="AP2T2" />
    <coupling fromModel="T2" from="AT2P1" toModel="P1" to="AT2P1" />
    <coupling fromModel="T2" from="AT2P3" toModel="P3" to="AT2P3" />
    <coupling fromModel="P3" from="AP3T3" toModel="T3" to="AP3T3" />
    <coupling fromModel="P4" from="AP4T3" toModel="T3" to="AP4T3" />
    <coupling fromModel="T3" from="AT3P5" toModel="P5" to="AT3P5" />
    <coupling fromModel="P5" from="AP5T4" toModel="T4" to="AP5T4" />
    <coupling fromModel="T4" from="AT4P6" toModel="P6" to="AT4P6" />
    <coupling fromModel="T4" from="AT4P4" toModel="P4" to="AT4P4" />
    <coupling fromModel="T1" from="CT1" toModel="P1" to="CT1P1" />
    <coupling fromModel="T1" from="CT1" toModel="P6" to="CT1P6" />
    <coupling fromModel="T2" from="CT2" toModel="P2" to="CT2P2" />
    <coupling fromModel="T3" from="CT1" toModel="P3" to="CT3P3" />
    <coupling fromModel="T3" from="CT1" toModel="P4" to="CT3P4" />
    <coupling fromModel="T4" from="CT4" toModel="P5" to="CT4P5" />
    <coupling fromModel="P1" from="CP1T1" toModel="T1" to="CP1T1" />
    <coupling fromModel="P6" from="CP6T1" toModel="T1" to="CP6T1" />
    <coupling fromModel="P2" from="CP2T2" toModel="T2" to="CP2T2" />
    <coupling fromModel="P3" from="CP3T3" toModel="T3" to="CP3T3" />
    <coupling fromModel="P4" from="CP4T3" toModel="T3" to="CP4T3" />
    <coupling fromModel="P5" from="CP5T4" toModel="T4" to="CP5T4" />
  </IC>
- <EOC>
    <coupling fromModel="P1" from="OUTP1" toModel="Producer_Consumer" to="OUTP1" />
    <coupling fromModel="P2" from="OUTP2" toModel="Producer_Consumer" to="OUTP2" />
    <coupling fromModel="P3" from="OUTP3" toModel="Producer_Consumer" to="OUTP3" />
    <coupling fromModel="P4" from="OUTP4" toModel="Producer_Consumer" to="OUTP4" />
    <coupling fromModel="P5" from="OUTP5" toModel="Producer_Consumer" to="OUTP5" />
    <coupling fromModel="P6" from="OUTP6" toModel="Producer_Consumer" to="OUTP6" />
    <coupling fromModel="T1" from="OUTT1" toModel="Producer_Consumer" to="OUT1" />
    <coupling fromModel="T2" from="OUTT2" toModel="Producer_Consumer" to="OUT2" />
    <coupling fromModel="T3" from="OUTT3" toModel="Producer_Consumer" to="OUT3" />
    <coupling fromModel="T4" from="OUTT4" toModel="Producer_Consumer" to="OUT4" />
  </EOC>
  <states />
  <s_ext />
  <s_int />
  <s_con />
  <lambda />
</CoupledDEVS>
```

Fig A.14 Modèle DEVS couplé correspondant au RDP Producteur-Consommateur (partie 2).