Outils formels et opérationnels pour la modélisation et la simulation des systèmes complexes

Noureddine / Seddari

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université 20 Août 1955-Skikda
Faculté des Sciences
Département d'Informatique

THESE

En vue de l'obtention du diplôme de :

Doctorat de 3° cycle (LMD) en Informatique

Option : Computation et Cognition des systèmes informatiques

Présentée par :

Noureddine SEDDARI

Thème :

Outils formels et opérationnels pour la modélisation et la simulation des systèmes complexes

Soutenu publiquement le : 03/06/2015

Devant le jury composé de :

***Président* :**	B. BOUCHEHAM	MC 'A', Université 20 Août 1955 – Skikda
***Examinateurs* :**	M. BENMOHAMMED	Professeur, Université Abdelhamid Mehri – Constantine 2
	M. BATOUCHE	Professeur, Université Abdelhamid Mehri – Constantine 2
	S. MAZOUZI	MC 'A', Université 20 Août 1955 – Skikda
***Rapporteur* :**	M. REDJIMI	Professeur, Université 20 Août 1955 – Skikda

A mes parents…

Remerciements

Je tiens à remercier,

Tout d'abord ALLAH le tout-puissant pour la puissance, la volonté, le courage et la patience qu'il m'a donné pour accomplir ce travail

Je tiens aussi à remercier respectueusement mon directeur de thèse, Monsieur REDJIMI Mohammed, professeur à l'Université 20 Août 1955 Skikda, pour m'avoir conseillé depuis mes années de Master. Je le remercie tout particulièrement pour l'attention qu'il m'a porté ainsi que son soutien inconditionnel au cours de ces années. Nos nombreuses discussions et nos échanges de points de vue et d'idées m'ont permis d'avancer dans mes recherches.

Mes vifs remerciements vont également à tous les membres du jury :

- Monsieur BOUCHEHAM Bachir, Maitre de conférences 'A' à l'université 20 Août 1955- Skikda, Président du jury.

- Monsieur BENMOHAMMED Mohamed, Professeur à l'université Abdelhamid Mehri, Constantine2,

- Monsieur BATOUCHE Mohamed Chawki, Professeur à l'université Abdelhamid Mehri, Constantine 2,

- Monsieur MAZOUZI Smaine, Maitre de conférences 'A' à l'université 20 Août 1955- Skikda.

qui m'on fait l'honneur d'accepter de participer à mon jury.

Je voudrais adresser un remerciement à tous les enseignants et les chercheurs du département informatique de l'université 20 Aout 1955-Skikda. Particulièrement, Mr. BENOUDINA Lazhar, Mr. LAROUM Toufik, Mr. BOUKELKOUL Sofiane ainsi que Mr. BELAOUED Mohamed pour leur aide et conseils.

Je remercie très chaleureusement l'ensemble du staff administratif et technique du complexe de liquéfaction du gaz naturel GL1/K-Sonatrach de Skikda particulièrement les départements technique et de production pour m'avoir permis

d'accéder à ce complexe et pour tous les détails techniques mis à ma disposition, en particulier: Mr. R.CHAOUACH et Mr. R.BOUREFIS et l'inoubliable Mr. S.Taghane chef du département informatique au niveau de l'entreprise SOMIK-Sonatrach de Skikda qui m'a guidé et m'a encouragé.

Un grand merci également à Monsieur le Directeur de la Réglementation et des Affaires Générales (DRAG) de la wilaya de Skikda, TAIEF Lakhdar; et à tous les collègues et personnels de la direction DRAG pour leur extrême gentillesse.

Je remercie tous mes amis pour leur aide et encouragement.

Je remercie fortement tous les membres de ma famille qui ont toujours été à mes cotés et qui m'ont soutenu dans mes choix au cours de ces longues années d'études.

Skikda, le 2 Décembre 2014.

Résumé

La modélisation et la simulation des systèmes revêtent une grande importance dans l'étape de la conception et de la représentation de tels systèmes. En effet, ces techniques permettent de fournir un prototype formel ou matériel conforme aux spécifications du système réel ou à une ou plusieurs de ses parties et dont le comportement, quand il est soumis à un ensemble d'entrées provoque un ensemble de sorties, qui permettent lorsqu'elles sont projetées sur le système réel d'en déduire un ensemble de conclusions. La modélisation et la simulation des systèmes complexes sont basées principalement sur leur décomposition en sous-systèmes plus simples à manipuler. Ce découpage doit se faire de façon méthodique par l'identification et la définition complète des différentes structures, actions et interactions qui régissent ces sous-systèmes. Généralement, le comportement du modèle global peut se déduire de l'émergence des comportements des modèles représentant ces sous-systèmes.

Plusieurs travaux de recherche et de nombreuses implémentations ont été conduits sur le thème de la modélisation et de la simulation des systèmes complexes. De ce fait, il existe aujourd'hui une grande panoplie de méthodes, d'outils et de produits logiciels dans ce cadre.

Le travail présenté dans cette thèse est une contribution dans le domaine de la modélisation et de la simulation des systèmes complexes ; particulièrement, les processus industriels complexes.

Après avoir mené une étude approfondie dans le cadre de la modélisation et de la simulation des systèmes complexes, l'approche proposée se base principalement sur la transformation des modèles DEVS (Discrete EVent system Specification) en modèles agent afin qu'ils soient simulables et implémentables au sein de plateformes multi-agents. En particulier, le modèle AGR (Agent/Groupe/Rôle) qui est un modèle conceptuel sous-jacent du modèle organisationnel AALAADIN, permet une décomposition d'un système complexe en groupes d'agents tout en déterminant toutes les interactions entre les groupes et les agents qui les composent ainsi que les rôles joués par les agents dans ces groupes.

Dans l'approche préconisée ; la décomposition du système global en sous-systèmes apparait à deux niveaux : Au niveau le plus bas ; la décomposition porte sur la division du processus global en modèles atomiques et couplés basés sur le formalisme DEVS. Ces modèles atomiques et couplés obtenus peuvent alors être formellement vérifiés et validés. A un niveau plus élevé, ces modèles sont mis en œuvre et implémentés grâce aux Systèmes Multi-Agents (SMA). Ainsi, les modèles atomiques sont transformés en agents et les modèles couplés en groupes d'agents. L'environnement dans lequel évoluent les agents est représenté par l'ensemble des données et informations partagé par les agents et qui leur sont accessibles par les transmissions de messages. Une version de ce travail a été implémentée sur la plate-forme multi-agents MAD-KIT et a abouti à l'implémentation d'un simulateur industriel.

L'avantage de cette démarche est son adaptabilité ainsi que les possibilités d'extension. De plus la décomposition en sous-systèmes réduit considérablement la complexité des

éléments mis en œuvre et permet, ainsi une grande modularité et une meilleure lisibilité du système.

Ce travail est réalisé en collaboration avec les départements de production et technique de liquéfaction du gaz naturel (GL1/k Skikda), un des principaux pôles d'hydrocarbures du complexe SONATRACH en Algérie.

Mots clés: Modélisation et Simulation, Systèmes Multi-Agents (SMA), Systèmes Industriels, Systèmes complexes, Discrete EVent systems Specification (DEVS), Modèles atomiques et couplés, AGR (Agent/Groupe/Rôle), MAD-KIT, AALAADIN.

Abstract

The modeling of complex systems is generally based on their decomposition in sub-systen easier to handle. This division has to be made in a methodical way by identification and complete definition of the various structures, actions, and interactions of these sub-systems. The behaviour of overall model arises from the emergence of the model's behaviours of these sub-systems.

The work presented in this thesis is a contribution in the modeling and simulation field of the complex systems particularly, the complex industrial processes. Our approach is based mainly on the transformation of a DEVS model (Discrete EVent system Specification) into agent model so that it can be simulated in the multi-agents platforms in particular, based on AGR model (Agent/Group/Role) which is the subjacent conceptual model of organizational model AALAADIN. The use of this model is for the organization of the system to be simulated during the transformation phase into groups of agents while determining all the interactions between the groups and the agents as well as the roles played by the agents in these groups seems to be the most suitable solution. This is concerns an algorithm with a set of functions and procedures which make it possible to transform these models systematically.

In the recommended approach and on the implementation step; the decomposition of those systems into sub-systems appears on two levels: On the lower level; the decomposition concerns the division of global system into atomic and coupled models based on DEVS formalism. Both obtained atomic and coupled models are formally verified and validated. At the higher lever, the implementation of those models is done through the use of Multi-Agents Systems (MAS). Thus, the atomic models are transformed into agents and coupled models into groups of agents. A version of this work was implemented on multi-agents platform MAD-KIT to realize an industrial simulator an industrial simulator.

The advantage of this approach is its adaptability as well as the possibilities of extension. Moreover, the decomposition into sub-systems reduces significantly the complexity of the elements being implemented and therefore, allows a great modularity and a best legibility to the system.

Our work is realized in collaboration with the production and technique departments of natural gas liquefaction (GL1/K Skikda), one of the principal hydrocarbons poles from SONATRACH complex in Algeria.

Keywords: Modeling and simulation, Multi-Agent Systems (MAS), Industrial systems Complex systems, Discrete Event System Specification (DEVS), Atomic and coupled models, AGR (Agent/Group/Role), MAD-KIT, AALAADIN.

ملخص

نمذجة الأنظمة المعقدة يمكن أن تكون مرتكزة أساسا على تقسيمها إلى أنظمة جزئية سهل التعامل معها. هذا التقسيم يجب أن يتم بطريقة منهجية مع تحديد و تعريف كامل لمختلف الهياكل، العمليات و التفاعلات التي تقوم عليها هذه الأنظمة.

العمل المقدم في هذه الأطروحة هو مساهمة في مجال نمذجة ومحاكاة الأنظمة المعقدة و خاصة العمليات الصناعية المعقدة . أطروحتنا مرتكزة أساسا على تحويل نموذج DEVS « Discrete EVent System Specification » إلى نموذج وكيل قابل للمحاكاة في قواعد متعددة الوكلاء، على وجه الخصوص المرتكزة على النموذج AGR (Agent/Groupe/Rôle) الذي يمثل النموذج التصوري للنموذج التنظيمي الأساسي AALAADIN . إستعمال هذا النموذج خلال مرحلة التحويل إلى مجموعة من الوكلاء مع تحديد جميع التفاعلات بين مختلف المجموعات و الوكلاء، و كذلك الأدوار التي تقوم بها تعتبر الحل الأنسب. يتعلق الأمر بمجموعة من الخوارزميات و الإجراءات التي تسمح بالتحويل النظامي لهذه النماذج.

في مرحلة التطبيق؛ تقسيم النظام الكلي إلى أنظمة جزئية يظهر على مستويين:على مستوى أدنى؛ يتم تقسيم النظام إلى نماذج بسيطة و مركبة مرتكزة على الشكلية DEVS. هذه النماذج الناتجة يمكن التحقق و المصادقة عليها. على مستوى أعلى؛ يتم تنفيذها باستعمال الأنظمة متعددة الوكلاء (SMA). لذا، يتم تحويل النماذج البسيطة إلى عملاء و المركبة إلي مجموعات من العملاء. نسخة من هذا العمل تم إنجازها على مستوى القاعدة المتعددة الوكلاء MAD-KIT و التي أسفرة على إنجاز محاكي صناعي.

ميزة هذه المنهجية المتبعة تكمن في مرونتها و إمكانية التوسع فيها. وعلاوة على ذلك ميزة التقسيم إلى أنظمة جزئية يقلل بدرجة كبيرة التعقيد في العناصر المنفذة مما يسمح من رؤية واضحة للنظام.

العمل المنجز تم بالتعاون مع مركب تميع الغاز الطبيعي (GL1/k) واحد من أهم أقطاب المحروقات لمجمع سوناطراك في الجزائر.

كلمات مفتاح : النمذجة و المحاكاة ، الأنظمة متعددة الوكلاء (SMA) ، الأنظمة الصناعية ، الأنظمة المعقدة ، Discrete EVent system Specification (DEVS)، نماذج بسيطة و مركبة، AGR (Agent/Groupe/Rôle) ، MAD-KIT ،AALAADIN.

Table des matières

Liste des figures

Liste des tables

Liste des algorithmes

Liste des Abréviations

Abréviation	Nom complet
DEVS	Discrete EVent system Specification
PDEVS	Parallel Discrete EVent system specification
Cell-DEVS	Cellular Discrete EVent system specification
RTDEVS	Real-Time Discrete EVent system specification
DS-DEVS	Dynamic Structure Discrete EVent system specification
DSDE	parallel Dynamic Structure Discrete Event system specification
MOOSE	Multi model Object Oriented Simulation Environment
Eclpss	Ecological Component Library for Parallel Spatial Simulation
ADEVS	A Discrete EVent system Simulator
AToM3	A Tool for Multi-formalism and Meta-Modeling
SMA	Système Multi-Agent
AGR	Agent, Groupes, Rôles
MadKit	Multi-agent development Kit
FIPA	Foundation for Intelligent Physical Agents
KQML	Knowledge Query and Manipulation Language
ACC	Agent Communication Chanel
ACL	Agent Communication Language
BDI	Believe Desire and Intention
ADELFE	Atelier pour le DEveloppement de Logiciels à Fonctionnalité Emergente
VLE	Virtual Laboratory Environment
IDM	Ingénierie Dirigée par les Modèles
OMG	Object Management Group
MDA	Model Driven Architecture
CIM	Computation Independent Model
PIM	Platform Independent Model
PSM	Plat-form Specific Model
JAMES	Java-based Agent Modeling Environment for Simulation
GALATEA	GLIDER with Autonomus, Logic-based Agents, Temporal reasoning and Abducation
UML	Unified Modelling Language
AAII	Australian Artificial Intelligence Institute Methodology
MaSE	Multi agent Software Engineering
DESIRE	DEsign and Specification of Interacting REasoning framework

PASSI	Process for Agent Societies Specification and Implementation
MESSAGE	Methodology for Engineering Systems of Software AGEnts
OAA	Open Agent Architecture
RADL	Agent Definition Language
RTAE	Run-Time Agent Engine
JADE	Java Agent DEvelopment framework
JAVA	JVM, Java Virtual Machine
SWARM	Swarm (Software for Agent-based Modeling Ressource
CORMAS	Common-pool Resources and Multi-Agent Systems
XML	eXtensible Markup Language
DCS	Distributed Control System
APM	Advanced Process Manager
AM	Application Module
GL1/K	Complexe de Liquéfaction du Gaz Naturel SKIKDA
FIC	Flow Indicator Controller
FV	Flow Valve
HM	History Module
API	Application Program Interface
HPM	High Performance Manager
GUS	Global Use Station
LCN	Local Control Network
LIC	Level Indicator Controller
AIC	Air Indicator Controller
PIC	Pression Indicator Controller
HIC	Hand Indicator Controller
LV	Level Valve
NIM	Network Interface Module
TIC	Temperature Indicator Controller
PM	Process Manager
BMS	Burner Management system
UCN	Universal Command Network
US	Universal Station
PIN	Plant Information Network
PC	Partie Comande
PLNM	Plan Local Network Module
PID	Process Instrument Diagramm
LCV	Level Controller Valve
ACV	Alarm Controller Valve
TOR	Tout Ou Rien
SdC	Salle de contrôle/Commande

Introduction générale

Un système complexe est un système composé d'un grand nombre d'entités en interaction. Dans un tel système, le comportement global émerge de l'interaction des entités qui le composent [Simon, 1969; Bertalanffy, 1968]. Cette notion d'émergence mérite d'être bien définie et complètement déterminée, car, 'le tout n'est pas forcément l'ensemble des parties'.

A ce stade, nous pouvons considérer deux grandes approches ; la première dite approche descendante considère un système dans sa totalité puis tend à le décomposer en un ensemble de sous-systèmes tout en établissant les actions qui régissent chacune de ces parties ainsi que les relations et interactions horizontales et verticales qui existent entre les différents composants. La deuxième approche dite ascendante part de composants de base, dont les différentes structures, actions et comportements sont connus et parfaitement déterminés pour construire le système global. Nous trouvons ; à ce niveau ; une notion très utilisée actuellement qui est la notion de réutilisation. Nous pouvons aussi considérer des approches hybrides dans lesquelles ces méthodes seraient combinées.

Le domaine de la modélisation/simulation tire un profit considérable de la décomposition des systèmes concernés. Il est ; en effet ; plus simple et plus facile d'agir sur un élément atomique d'un système que de considérer le système dans sa totalité.

La modélisation et la simulation [Fishwick 1995, 1997; Sonnessa, 2004; Shannon 1976, 1998; Oussalah, 1998; Ingalls, 2001; Vangheluwe, 2008] des systèmes permettent de manipuler, d'observer et d'améliorer la compréhension des phénomènes mis en jeu. Ces derniers sont étudiés à différentes échelles comme pour la simulation des changements climatiques, des phénomènes physiques, des systèmes de trafic urbain…. La plupart des phénomènes modélisés atteignent aujourd'hui des complexités et des degrés de finesse élevés qui imposent l'utilisation de modèles et d'outils informatiques de plus en plus performants, flexibles et d'une complexité généralement assez élevée.

La modélisation consiste en la représentation d'un système par un modèle théorique ou pratique dont la finalité est de déterminer une ou plusieurs 'entités' théoriques ou pratiques simplifiées, fidèles et observables de la structure et du comportement du système réel ou d'une ou plusieurs de ses parties [Oussalah, 1988]. Le concept de modélisation a été toujours utilisé par l'homme, en particulier, dans le but d'avoir un prototype sur lequel il pourrait mettre en œuvre une série d'expériences dans des conditions différentes et d'en analyser les résultats afin d'en tirer des conclusions qui seront projetées sur le système réel. Cette deuxième étape est appelée simulation.

La simulation consiste, donc, à faire «exécuter» le modèle pour différentes entrées et à en observer les sorties associées à des moments et des durées précis afin d'admettre une série de expériences.

Plusieurs auteurs s'accordent à définir les notions de modélisation et de simulation comme étant le processus de conception d'un modèle du monde réel puis d'en conduire un ensemble d'expériences dans le but d'en comprendre le comportement ou pour évaluer différentes stratégies dans les limites d'un ensemble de critères fixés pour l'évolution du système [Fishwick, 1995; Ingalls, 2001;Shannon, 1998].

Pratiquement, dès leurs apparitions, les outils informatiques ont été appliqués dans le domaine de la modélisation/simulation. Le choix d'éléments adéquats assure de bonnes solutions à moindres coûts. De ce fait, la première approche (le choix du formalisme de modélisation et les outils informatiques adaptés) est très importante pour la suite du projet dont l'amélioration doit être constante.

Notre travail rentre dans ce cadre ; nous avons opté pour une méthode hybride qui combine le formalisme DEVS (Discrete EVent system Specification) [Zeigler *et al.,* 2000a] et le modèle AGR (Agent/Groupe/Rôle).

Une application de cette approche, a été menée pour la modélisation et la simulation de procédés industriels. Selon son importance, un système industriel peut contenir des milliers d'éléments et même plus. C'est le cas, par exemple des complexes pétroliers.

Aujourd'hui, il existe, une panoplie très importante de méthodes, d'outils, de produits, de plate-formes et d'environnements de développement destinés à la modélisation et à la simulation. Ainsi, dans l'application que nous présentons ; il s'agit de prendre en charge un ensemble d'éléments passifs dans le sens où ces éléments forment la partie opérative du

système (électrovannes, pompes, ballons, turbocompresseurs, chaudières, ventilateurs, fours,…), ainsi que des éléments actifs dans le sens où ces éléments forment la partie contrôle (régulateurs, contrôleurs, serveurs,…), ainsi que des éléments d'interaction et de communication (tuyaux, fils, signaux de commande, signaux de contrôle, alarmes, écrans de visualisation, synoptiques, claviers…) et autres éléments de mémorisation, de stockage, d'archivage et de traitement de l'information.

L'idée que nous développons ici se base sur la décomposition du système en employant le formalisme DEVS. Ce formalisme est un concept présenté et développé dans les années 70 par le professeur B.P Zeigler [Zeigler *et al.,* 2000a]. DEVS définit aussi bien les structures que les évolutions (comportements) des entités qui composent un système en employant deux éléments de base : les modèles atomiques et couplés. Le fonctionnement de l'ensemble des modèles dans DEVS est décrit d'une manière mathématique qui peut être formellement vérifiée et validée.

D'autre part, les systèmes industriels complexes sont généralement classés en tant que systèmes à événement discrets (SED) et selon B.P. Zeigler [Zeigler *et al.,* 2000b], ce formalisme est adéquat pour représenter l'ensemble de ce type de systèmes.

Dans un deuxième temps, les modèles obtenus sont transformés en modèle AGR pour leur implémentation sur des plateformes multi-agents. Les approches des systèmes multi-agents (SMA) présentent les outils théoriques et pratiques assez robustes pour la simulation. Un système multi-agents inclut des outils évolués pour la gestion et le contrôle des agents (création, destruction, interactions entre les agents...). A ce niveau, l'exécution du système peut être facilement réalisée.

L'implémentation du système est faite grâce à la plateforme MadKit ; cette plateforme a été mise au point par l'équipe du professeur J. Ferber au LIRMM [Gutknecht et Ferber, 2000] et repose sur les principes d'agent, de rôle et de groupe. Ceci, dans le but de profiter de la puissance des outils évolués cités précédemment.

Un ensemble de procédures permettant d'effectuer des passages entre modèles DEVS et AGR a été ainsi déterminé et mis en œuvre. Ces procédures sont détaillées dans la suite de cette Thèse. L'implémentation de ce processus elle donne un simulateur industriel dans le domaine des hydrocarbures.

Notre manuscrit est organisé comme suit :

- Le premier chapitre présente quelques concepts de base concernant les systèmes et leur complexité. nous nous y intéressons, en particulier, à la modélisation et à la simulation des systèmes complexes. Dans la deuxième partie de ce chapitre, nous nous attachons à présenter un ensemble de techniques de modélisation. Á la fin, nous abordons la simulation informatique, ses catégories et ses processus de conception.

- Dans le second chapitre ; nous introduisons le paradigme multi-agent et ses caractéristiques. Différentes architectures, interactions et méthodologies de conception des systèmes multi-agents seront ensuite étudiés. La fin de ce chapitre est consacrée aux outils et environnements des SMA.

- Le troisième chapitre est dédié à la présentation du formalisme DEVS, ainsi que quelques unes de ses extensions. Quelques plateformes et environnements basés sur ce formalisme seront étudiés à la fin de ce chapitre.

- Dans le quatrième chapitre, nous présenterons l'approche de transformation de DEVS vers le modèle AGR qui constitue notre contribution. Cette transformation est mise en œuvre et validée sur la plateforme Mad-Kit. Une brève introduction de différentes approches autour des modèles, en particulier l'Ingénierie Dirigée par les Modèle (IDM) et l'Architecture Dirigée par les Modèles (ADM) est proposée au début de ce chapitre. De même, un ensemble de travaux similaires de transformations entre DEVS et les SMA sont détaillés dans ce chapitre. Les algorithmes et l'ensemble des fonctions et procédures de notre approche de transformation sont détaillés en fin de chapitre.

- Le dernier chapitre illustre l'application de notre approche sur une étude de cas qui consiste en l'application de notre approche à la modélisation et à la simulation de procédés relatifs à l'industrie pétrochimique au sein de l'unité GNL de SONATRACH Skikda. Le simulateur obtenu, qui est opérationnel y est détaillé.

- Une conclusion générale et quelques perspectives futures de ce travail sont présentées à la fin de cette thèse.

CHAPITRE 1

MODELISATION ET SIMULATION DES SYSTEMES COMPLEXES

1.1 Introduction

Aujourd'hui quand on parle de modélisation et de simulation, beaucoup de gens ont tendance à confondre ces deux termes en les réduisant à une même étape, la représentation de la dynamique d'un phénomène physique, or la différence est que la modélisation est l'étape qui consiste à trouver une autre forme de cette dynamique au contraire la simulation représente le résultat de l'utilisation de ce modèle. Une autre façon de dire et que la modélisation représente une première étape dans le processus de simulation. Ce chapitre constitue une introduction à notre travail par la présentation de quelques notions, qui nous semblent essentielles, sur la modélisation et la simulation d'une façon générale. Nous étudierons différentes techniques, paradigmes et formalismes pour la modélisation, ainsi que, les processus de conception de simulations informatiques.

1.2 Les systèmes complexes

1.2.1 Notion de système

Dans la littérature, on trouve plusieurs aspects liés à la notion de système et elle revêt plusieurs définitions. Une définition assez généralement acceptée est qu'un système peut être considéré comme un ensemble d'éléments qui interagissent pour atteindre un objectif défini [Rowe, 1965]. Cette notion de système se rencontre dans tous les domaines de la vie de tous les jours, de l'économie, de la science,… On parle de système digestif pour caractériser l'ensemble des organes et des fonctions nécessaires à l'opération de s'alimenter et de digérer les aliments. Le système éducatif est caractérisé, quant à lui, par l'ensemble des structures, des infrastructures, des ressources humaines et matérielles ainsi que l'ensemble des programmes, textes et autres ressources pédagogiques propres à la formation des élèves. Le système économique d'un pays comprend l'ensemble des activités, des ressources, des méthodes et autres études concernant le domaine économique d'un pays. Ainsi, un système n'est pas un assemblage disparate d'éléments. Il consiste en un groupement de composants bien déterminés qui fonctionnent ensemble pour atteindre un but ou un objectif commun. Pour 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."

D'autre part, selon A. Hall et R. Fagen [Hall et Fagen, 1956] *"A system is a set of objects together with relationships between the objects and between their attributes."* Dans cette définition les attributs sont les propriétés des objets. Les relations relient les parties du système. Un système ne peut pas être considéré en tant que tel s'il n'a pas un but. Cette affirmation implique également qu'un système a des propriétés, des fonctions ou des buts distincts de ses objets, attributs et relations. Les objets peuvent être abstraits tels que des variables ou des équations ou des parties physiques. Les relations sont celles qui relient le système.

A un instant donné, le système est dans un état particulier défini par l'état de ses composants et des relations qui les relient, cet état change lorsque l'état de ses composants et/ou l'état des relations qui les relient change.

Généralement, les systèmes peuvent être classés en systèmes discrets ou continus [Law et Kelton, 2000]. Un système est dit discret si ses variables d'état changent à des points discrets du temps. Dans un système continu, ces variables peuvent évoluer de façon continue. Certains systèmes ne sont ni complètement discrets ni complètement continus.

1.2.2 Types de systèmes

Il existe plusieurs types de systèmes : statiques ou dynamiques, déterministes ou stochastiques, continus ou discrets.

1.2.2.1 Système déterministe

Il fonctionne de manière prévisible, l'interaction entre les différentes parties est connue avec exactitude. Si l'on possède une description de l'état du système à un moment donné, le prochain état du système peut être donné exactement et sans erreur.

1.2.2.2 Système probabiliste (stochastique)

Il est décrit en termes de comportement probable (qui dépend du hasard). Une certaine marge d'erreur accompagne toujours la prédiction de ce que fera le système.

1.2.2.3 Système continu

Les changements de l'état du système se font en permanence avec le temps (dépend de variables liées au temps).

1.2.2.4 Système discret (discontinu)

Le système est caractérisé par des événements qui surviennent à des instants non fixes et causent un changement de l'état jusqu'au prochain événement.

1.2.2.5 Système mixte

La séparation entre système continu et discret est en quelque sorte artificielle, car en réalité la plupart des systèmes possèdent des composantes continues et discrètes à la fois. De tels systèmes sont dits mixtes.

1.2.3 Notion de système complexe

Un système complexe caractérisé par un grand nombre d'entités en interaction. Dans un tel système, le comportement global émerge de l'interaction des entités qui le composent. Cette notion d'émergence mérite d'être bien définie et complètement déterminée. Deux grandes approches sont considérées à ce stade ; la première approche, dite « descendante ou top-down», considère un système dans sa totalité puis tend à le décomposer en un ensemble de sous-systèmes tout en établissant les actions qui régissent chacune de ces parties ainsi que les relations et interactions verticales et horizontales qui existent entre ses différents composants. La deuxième approche, dite « ascendante ou bottom-up », part de composants de base dont les différentes structures, actions et comportements sont connus et parfaitement déterminés pour construire le système global.

1.3 Modélisation et simulation

1.3.1 Modélisation informatique

La modélisation informatique consiste en la construction d'un ensemble de programmes qui 'décrivent' la structure ou architecture du système réel et l'agencement des différents composants de ce système ainsi que les interactions entre ces composants. L'exécution de ces programmes sur un ordinateur pour des valeurs des données déterminées conduit à un ensemble d'états de ces programmes et abouti à des ensembles de résultats. Cette dernière étape est appelée simulation du système. La modélisation du moteur d'une voiture, par

exemple commence par la construction des modèles correspondants aux différents composants de ce moteur (pièces: chemises, bielles, pistons, culasses, bougies, ….) ainsi que les différents agencements de ces composants (Une chambre de combustion contient des soupapes, des bogies, des pistons,….) et les interactions entre ces différents composants (l'allumage d'une bougie provoque une étincelle qui enflamme le mélange air-carburant qui provoque une explosion qui pousse les pistons d'une position a (état actuel) vers une position b (état futur), …). Ainsi, le programme informatique devra modéliser tout ces phénomènes.

1.3.2 Notion de modèle

D'une façon générale un modèle est une représentation d'un système réel (physique, économique, humain,…etc) réalisé dans le but de mieux étudier ce système est expliquer certains de son comportement.

D'apprêt P. Hubert [Hubert, 1969] **«** *un modèle et défini comme un objet ou une personne à imiter, un exemple ou un archétype. Il est devenu dans le domaine scientifique une construction matérielle ou abstraite "ressemblant" à l'objet modélisé, selon un certain nombre de caractéristiques pertinentes eu égard aux données disponibles et à l'objectif poursuivi* **».**

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 ».*

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

Parfois, il est impossible d'étudier le système directement du fait qu'il soit inaccessible (système solaire), trop couteux, il change trop rapidement (tir nucléaire) ou lentement (mouvement d'une comète), dans ce cas, l'étude est faite sur un modèle, c'est-à-dire un deuxième système conforme à l'original est aussi parfait que l'étude l'exige.

Un modèle résulte des fonctions et actions menées sur le système réel ou une de ses parties qui intéressent le domaine d'étude. Ces fonctions et actions sont menées dans un cadre expérimental et sont exprimées à l'aide de formalismes et de paradigmes selon une ou

plusieurs méthodologies. La notion de paradigme se substitue ici à la notion de vérité dans le cadre général. La figure 1.1 représente l'activité de modélisation.

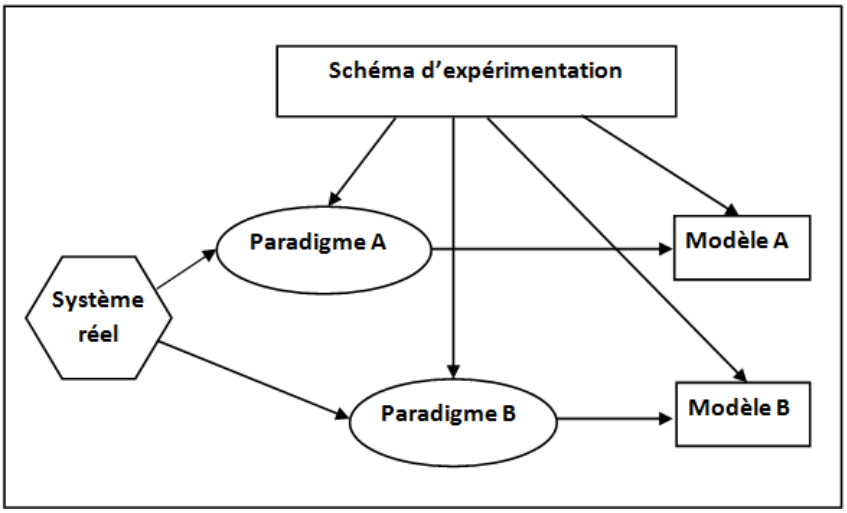

FIG. 1.1 – L'activité de modélisation [Quesnel, 2006]

En théorie des systèmes, on distingue, généralement deux points de vues complémentaires l'un de l'autre : Un point de vue structurel qui a trait à l'architecture même du système considéré et un point de vue comportemental qui considère les évolutions du système dans le temps.

1.3.2.1 Modélisation de l'aspect comportemental du système

On s'intéresse ici à l'aspect dynamique du système c.à.d. son comportement et ses évolutions dans le temps. Ainsi, le modèle est caractérisé par un ensemble de variables d'état : le temps ainsi que l'ensemble des fonctions qui expriment des relations entre les variables et le temps.

Le modèle est sensé évoluer d'un état S_i vers un état S_j à l'occurrence d'un ensemble d'évènements. Il est nécessaire, donc, de définir un sous-ensemble l'état du modèle par un sous ensemble de variables pour connaître le prochain état. Ce sous ensemble est nommé ensemble des variables d'état.

Selon N. Giambiasi : L'ensemble des variables d'état permet de prévoir (calculer) à l'instant 't' la valeur future de l'état du système à l'instant t+1 du modèle [Giambiasi, 2001].

Ainsi, le comportement du modèle sera déterminé par un ensemble de fonctions qui expriment le moyen de passer d'un état au suivant. Le modèle comportemental est régi par un ensemble de règles pour faire évoluer le modèle d'un état au suivant et exprimer, ainsi, sa dynamique.

1.3.2.2 Modélisation de l'aspect structurel du système

Le modélisateur définit la structure du modèle et s'intéresse aux sous-modèles qui le composent ainsi qu'à leurs interconnexions. Ainsi, le modélisateur détermine les niveaux d'abstraction pour en déduire les sous-modèles interconnectés qui définissent le comportement global du modèle.

La notion de décomposition est un concept important qui permet de définir comment le système peut être décomposé en sous-composants systèmes. A l'opposé, le concept de composition précise comment des modèles peuvent être couplés pour former des modèles plus complexes. Ces concepts utilisent la propriété de « fermeture sous la composition » [Zeigler, 1999] issue de la théorie des systèmes. Cette théorie repose sur une démarche formelle pouvant être mathématiquement prouvée.

En général, le modèle obtenu est un modèle ayant une construction hiérarchique composée de sous-modèles modulaires munis de ports d'entrée et de sortie interconnectés pouvant communiquer pour reproduire un comportement global.

1.3.3 Les formalismes de modèles

Les modèles peuvent être spécifiés par l'utilisation d'une grande variété de formalismes. Ces formalismes, offrent des outils de spécifications mathématiques. Des sous-classes de systèmes dynamiques sont choisis en fonction du système à modéliser et des objectifs à atteindre, utilisent des concepts et des propriétés mathématiques et possèdent deux dimensions : l'espace et le temps [Orën, 1987].

1.3.3.1 La dimension temporelle

Le temps peut être représenté de façon continue ou discrète.
Modèle à temps continu : la variable qui représente le temps prend des valeurs réelles.
Modèle à temps discret : le temps progresse par pas temporels, les valeurs prises par la variable temps sont des multiples du pas.

1.3.3.2 La dimension spatiale

La deuxième dimension de classification des modèles concerne l'espace où les variables descriptives du modèle peuvent prendre leurs valeurs dans un ensemble fini ou infini.
Modèle à variables continues : les variables descriptives prennent des valeurs réelles.
Modèle à variables discrètes : l'ensemble des valeurs des variables est fini.

1.3.4 Paradigmes et formalismes pour la modélisation

1.3.4.1 Le paradigme

La notion de paradigme désigne un système de représentations utilisé dans un domaine scientifique ou technique déterminé. Dans le domaine de l'activité de modélisation, cette notion renvoie à l'ensemble des définitions et formalismes ainsi que les outils, méthodes et techniques utilisés et acceptés dans cette activité. Cette notion suppose, donc, une certaine vision du monde, des outils et des méthodes valables pour le modélisateur [Kuhn, 1972]. Le paradigme agent, par exemple, est caractérisé par les notions d'autonomie, d'interaction, de pro-activité,…

Ce paradigme de modélisation est formalisé par *l'Unified Modelling Language* (UML). Plusieurs implémentation de ce paradigme en langages objets ont été faites telles que : Java, C++, Smalltalk... Ce paradigme se prête très bien à une modélisation discrète des entités d'un système et des scénarios d'interactions entre ces entités [Hill, 1996].
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* ".

1.3.4.2 Formalisme

Un formalisme est composé d'un ensemble d'outils et de règles mathématiques permettant la spécification formelle d'un modèle. Ainsi, un formalisme est utile à la spécification abstraite d'un modèle suivant un paradigme donné. Le langage formel particulier permettant cette description est appelé formalisme. Ainsi plusieurs formalismes peuvent exister pour un paradigme donné. Ceux-ci sont généralement classifiés. Les critères de la représentation du modèle, la gestion du temps, la représentation de l'espace, des objets modélisés ainsi que le rapport au hasard sont clairement spécifiés [Miller *et al.*, 2004; Quesnel, 2006].

Dans la suite nous citerons les formalismes les plus utilisés dans le cadre de la modélisation des systèmes.

1.3.4.2.1 Modélisation objet

Dans la modélisation orientée-objet, les systèmes sont uniquement constitués d'entités appelées objets. Ces objets sont définis par des types qui définissent de façon syntaxique et sémantique les propriétés que représenteront les objets du type ; ces propriétés permettent de délimiter, de qualifier les objets et de savoir comment on peut communiquer avec eux.

Ensuite, la mise en œuvre et l'implémentation de ces propriétés sont fournies par une classe, une structure de données qui lie à une propriété une implémentation possible. Cette implémentation peut être représentée soit sous forme d'attribut, soit sous forme de méthode. Dans ce formalisme de modélisation, si un type définit l'interface de l'objet, la classe lui fournit une implémentation. Pour cette raison, la classe est le moule par lequel est construit un objet. On dit alors d'un objet qu'il est une instance de telle classe. Les objets interagissent par l'envoi de message.

Le langage UML est devenu, aujourd'hui, une norme pour la spécification objet. Ainsi, le vocabulaire, les concepts et les formalismes objets sont normalisés grâce à l'OMG (Objet Management Group) [Booch *et al.*, 1997, Jacobson *et al.*, 1997, Rumbaugh *et al.*, 1997]. UML est un langage graphique proposant un ensemble de diagrammes. Les diagrammes statiques, ou diagrammes de classes et d'objets, représentent la structure statique du système et permettent d'identifier les entités et les relations entre les entités. On distingue, ainsi, Deux points de vue : le point de vue « objet », où ce sont les entités du système qui sont considérées dans une situation donnée et le point de vue « classe », où ce sont les classes d'objets qui sont considérées. Les objets sont des réalisations particulières de classes. Une classe est une abstraction d'un ensemble d'objets qui possèdent des caractéristiques identiques en termes d'attributs et de méthodes.

Dans la modélisation avec UML, les entités sont décrites à l'aide des attributs qui les caractérisent et les relations peuvent être typées. Ce type de modélisation est applicable quel que soit le domaine de modélisation, d'une part. D'autre part, il est possible d'avoir plusieurs niveaux d'abstractions : le niveau objet, le niveau classe, mais aussi des niveaux représentant des schémas de classes dits méta-classes.

1.3.4.2.2 La modélisation agent

La modélisation orientée agent permet une décomposition en sous-systèmes et une délégation naturelle des tâches, buts et/ou rôles à des sous-systèmes multi-agents et leurs agents.

Cette décomposition en sous-systèmes permet de concevoir et améliorer la compréhension des phénomènes de systèmes complexes par la décomposition du problème en sous-problèmes. Les sous-systèmes peuvent être distribués sur plusieurs machines et les interactions et communications se réalisent via des langages de communication qui sont généralement basés sur la théorie des actes de langages.

Plusieurs auteurs rapprochent la notion d'agent de celle de l'objet. Cependant, et en examinant plus attentivement ces deux notions, on remarque que les similitudes entre elles sont assez faibles. Un agent est par définition une entité autonome, active et réactive. Ainsi un objet ne peut pas s'exécuter de façon autonome et sans être invoqué contrairement à l'agent. Ce dernier est capable de prendre des décisions et d'établir des plans d'actions pour accomplir des activités complexes. Les agents peuvent être classifiés selon trois groupes distincts : les agents cognitifs qui sont des agents 'intelligents' et qui peuvent prendre des décisions de façon autonome et selon leurs propres buts de tels agents existent en nombre réduit dans un système de façon générale. La deuxième catégorie d'agents concerne les agents réactifs qui eux ont des marges de manœuvre plus réduites et répondent à des stimuli externes en déclenchant des actions. La troisième catégorie d'agents concerne les agents hybrides qui sont cognitifs et réactifs à la fois. Une synthèse de ces travaux peut être trouvée dans [Ferber, 1995; Grimm, 1999].

Bien que l'approche agent soit relativement récente, son application dans les domaines de la modélisation et la simulation de systèmes est assez courante.

1.3.4.2.3 Les systèmes d'équations différentielles

Une équation différentielle met en jeu une fonction inconnue *y*, ses dérivées jusqu'à un ordre *N* donné, et, explicitement ou non, un ensemble de variables ou de fonctions. La forme la plus générale d'une telle équation est :

$$F(x, y, y', y'', \ldots, y^{(n)}) = 0 \qquad (1.1)$$

La solution consiste à trouver l'intégrale de l'équation et c'est donc toute fonction y(x) qui satisfait la relation identiquement, c.-à-d. pour toute valeur de la variable *x*. Il n'est pas dans

nos propos de décrire ici les différents types d'équations différentielles, seuls certains aspects de ces systèmes nous intéressent.

Pour la modélisation des systèmes dynamiques, les équations différentielles sont des outils formels très puissants qui permettent de représenter leurs évolutions au cours du temps. Le système est ainsi représenté par un ensemble de paramètres caractérisés par des variables d'état. Comme, par exemple, la caractérisation de la trajectoire d'un point matériel faisant intervenir des distances, des vitesses ainsi que des accélérations en évolution dans le temps et pouvant se déduire par intégration et dérivation les unes des autres. Ainsi et le plus souvent, les équations différentielles correspondent à un < modèle agrégé >, entendons par là un modèle qui représente l'ensemble des mêmes entités composant le système par une grandeur moyenne. D'où la notion de modélisation descendante (top down). Les équations différentielles représentent l'outil privilégié de nombreuses disciplines scientifiques [Pavé, 1994] pour la modélisation de systèmes dynamiques tels que les systèmes naturels, en théorie et analyse des systèmes, ...Ce type de modélisation présente plusieurs avantages :

- Les techniques d'analyse numérique sont utilisées pour la résolution des systèmes d'équations différentielles lorsque les méthodes analytiques s'avèrent être insuffisantes ou inadaptées (trop complexes par exemple). Les techniques numériques ont bénéficié de grandes avancées méthodologiques qui leur ont permis de remplacer de façon efficace les méthodes analytiques en profitant des développements des outils informatiques,

- Les équations différentielles qui manipulent des grandeurs continues pour modéliser un système représentent ce dernier d'un point de vue phénoménologique, c.-à-d. en tenant compte de ses manifestations externes mesurables,

- Cet outil permet ainsi de faire varier le nombre d'entités composant le système considéré sans aucune charge de calcul supplémentaire.

Les équations différentielles s'adaptent à un grand nombre de systèmes. En contrepartie, elles imposent un point de vue sur les systèmes modélisés. Ceci étant données que les grandeurs considérées représentent souvent les systèmes comme des sous-ensembles ayant des caractéristiques communes. On parle, alors dans ce cas, de variables agrégées. Ceci pose le problème des interactions entre ces sous-ensembles qui composent le système global, considérées comme continues et réparties de façon homogène dans l'espace. Ainsi

l'adaptation de cette dernière hypothèse aux systèmes de haut niveau hiérarchique comme les écosystèmes où les systèmes sociaux posent un certain nombre de problèmes. Par exemple, il est souvent nécessaire de réinitialiser des variables du système pour prendre en considération des évènements ou perturbations ponctuels.

Un des intérêts majeurs des systèmes d'équations différentielles est de mettre en équation plusieurs grandeurs qui évoluent dynamiquement en conservant un ensemble d'interrelations. Ceci fournit des éléments de formalisation très appréciables pour le modélisateur qui peut ainsi étudier les évolutions de façon théorique. De telles études sont à la base des travaux concernant les écosystèmes théoriques et ont permis de mieux comprendre les équilibres dynamiques comme la stabilité quantitative ou la stratégie de résilience (permanence du réseau d'interaction) qui sont à la base de la permanence des écosystèmes [Frontier et Pinchod-Viale, 1995]. Cependant, lorsque le nombre d'interactions et d'espèces devient très grand, ces études deviennent trop complexes et difficiles voire même souvent impossibles [Pavé, 1994]. La résolution numérique des systèmes d'équations différentielles, c.-à-d. leur simulation, devient le seul moyen d'explorer leurs dynamiques. La simulation devient alors souvent le seul moyen de vérifier les hypothèses de fonctionnement émises sur un système complexe.

1.3.4.2.4 Les automates à états finis et à événements finis

Les équations différentielles, décrites dans la partie précédente, ne prennent pas en considération les 'traces' de la dynamique d'un processus. En ce sens que si l'on s'intéresse aux évolutions (exécutions) d'un processus d'un état S_i vers l'état suivant S_j, il faut calculer les valeurs de l'équation différentielle à ces deux instants. Pour pallier cet inconvénient, la notion d'automate inscrit les notions de grandeurs d'entrée, d'état, de transition et de grandeurs de sortie. Ainsi, dans le courant de pensées de B. P. Zeigler et de la spécification formelle de systèmes dynamiques, les notions d'états, de fonctions de transitions et de sortie doivent faire partie du formalisme.

Les automates à états finis constituent un outil simple et très utilisé qui est mis à la disposition du modélisateur. Ils permettent de décrire la dynamique d'un système donné à l'aide des notions d'état et de transition. Ainsi la structure d'un automate est donnée par l'équation suivante :

$$A = <X, S, \Delta, Y> \quad (1.2)$$

Où *X* représente l'ensemble des entrées du système.

S l'ensemble des états du système.

La fonction Δ concerne l'ensemble des transitions en indiquant l'état suivant en fonction de l'état courant et de l'entrée du système.

Y représente l'ensemble des grandeurs de sortie.

X est un ensemble de symbole qui peut se substituer à un ensemble d'événements et de disposer, ainsi, d'un outil de spécification à événements discrets. Un automate d'état fini est souvent représenté par un graphe où les sommets représentent les états et les arcs les transitions.

Les automates à événements finis et à événements finis constituent un point de vue supplémentaire 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 s'est écoulé entre les deux événements.

Il existe plusieurs sortes d'automates dont les automates non déterministes, où les transitions sont exprimées à l'aide de probabilités. Dans ce cas, les changements d'états suivent un modèle stochastique. Cette sorte d'automate est utilisée lorsque l'on ne connaît pas les processus qui régissent les changements d'états du système.

Le formalisme DEVS que nous présenterons de façon approfondie par la suite est un outil de spécification formelle de la dynamique des systèmes basé sur la notion d'automates à événements finis. Cet outil permet de spécifier formellement la dynamique du système en prenant en compte, explicitement ou non, les aspects temporels. La description de la dynamique peut faire partie, ou non, d'une description structurelle, c'est-à-dire, que la dynamique modélisée peut être celle d'une entité ou non. Nous pouvons, en effet, décrire les états du système global sans pour autant décrire le système en termes d'entités.

1.3.4.2.5 Les diagrammes dynamiques UML : les state charts

La modélisation par UML offre une extension objets des automates à états finis : les *state charts* où l'expression des transitions peut se construire autour des attributs de la classe des entités.

Les *state charts* peuvent être utilisés comme un automate à états finis. Ce formalisme s'intègre dans une démarche de modélisation orientée-objets dans sa version complète. Les diagrammes dynamiques UML est une combinaison des formalismes à base d'états et de transitions. On trouve parmi ces transitions : les transitions déclenchées sur des événements, les transitions temporisées et les transitions conditionnelles.

– Les transitions déclenchées sur événements : La transition est franchie à l'occurrence d'un évènement (un message dans le vocabulaire objets) provenant d'un autre *state chart*.

– Les transitions temporisées : la transition est franchie au bout d'un temps déterminé.

– Les transitions conditionnelles : La transition est franchie si la condition exprimée en fonction du vecteur d'états est vraie.

La syntaxe proposée par les *state charts* est relativement riche pour la modélisation de la dynamique d'un système. Cependant, un cadre formel pour la spécification proposée n'est pas intégré aux systèmes dynamiques. Il est vrai, néanmoins, que moyennant quelques précautions, les *state charts* peuvent offrir un cadre formel. Il est possible, en effet, de définir une spécification DEVS d'une partie des *state charts* [Borland et Vangheluwe, 2003]. Ainsi, les *state charts* forment un outil important pour la description de la dynamique et offrent des capacités d'expression parfaitement intégrées à une démarche de modélisation objets du système.

1.3.4.2.6 Les réseaux de Petri

Les Réseaux de Pétri (RdP) sont des modèles mathématiques à variables discrètes qui constituent un outil formel très puissant pour la spécification et la formalisation de la dynamique des systèmes. Proposés en 1962 par Carl Adam Petri [Petri, 1962], un RdP est un graphe biparti composé d'arcs orientés relient certaines places (représentées par des cercles) et des transitions (représentées par des barres). A chaque arc on associe un poids (entier positif). Deux places ne peuvent pas être reliées entre elles, ni deux transitions. Les places peuvent contenir un ou plusieurs jetons (représentés par des points), représentant généralement des ressources disponibles.

Les RdP [Peterson, 1977] peuvent être considérés comme une extension des automates à états finis. Nous y retrouvons les notions d'état, rôle joué par le marquage, et de transition. La logique de changement d'états est guidée par l'algorithme de franchissement des transitions. Les réseaux de Petri offrent, ainsi, une autre manière d'exprimer les transitions entre états. Les RdP ressemblent aux automates à états finis et à événements finis. La seule différence, c'est que le franchissement des transitions est fonction du marquage du modèle et que la description de ces franchissements se fait à l'aide de différentes interconnexions des places.

Les réseaux de Petri sont classés dans les formalismes hybrides du fait que les modélisateurs utilisant les réseaux de Petri affectent une sémantique aux places du réseau. La sémantique est souvent liée à l'état d'une entité du système.

D'autres formes des réseaux de Petri ont été formalisées par la suite. Par exemple, les réseaux de Petri colorés, où l'état du système devient beaucoup plus complexe puisque les jetons transportent des informations. Ces informations vont conditionner les transitions.

1.3.4.2.7 Les automates cellulaires

Les automates cellulaires forment un formalisme particulier pour la description des systèmes où l'espace est une composante importante. La notion d'espace considère aussi bien l'espace de variations d'une variable que l'espace topologique de la localisation des entités [von Neumann et Burks, 1966]. Ce formalisme représente la structure spatiale du système en utilisant une grille de cellules interconnectées et décrit le comportement local de chaque cellule de l'automate en fonction de son voisinage. Les automates cellulaires sont un des formalismes les plus utilisés en modélisation d'écosystèmes [Quesnel, 2006].

La figure 1.2, dresse une classification des formalismes basée sur trois critères : la manipulation de variables continues ou discrètes, la représentation de l'espace continu ou discret et la présence du temps continu ou discret. Cette classification dédiée à la spécification de la dynamique des systèmes [Ramat, 2003].

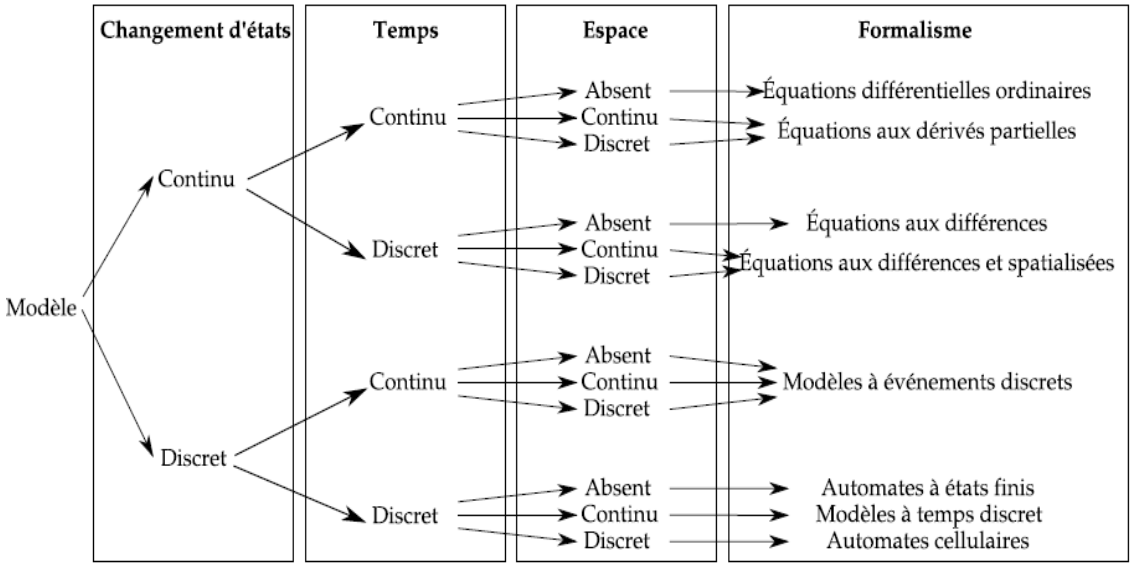

FIG. 1.2 – Classification des formalismes [Ramat, 2003]

1.3.3 La simulation informatique de systèmes complexes

La simulation informatique c'est la démarche scientifique (une procédure de recherche scientifique et technique) qui consiste à faire une reproduction numérique (artificielle), appelée modèle, du phénomène que l'on désire étudier, Elle consiste donc à observer le comportement de ce modèle lorsqu'on fait varier les actions que l'on peut implémenter et exercer sur ceci, et à en induire ce qui se passerait dans la réalité sous l'influence d'actions similaires. Cette démarche passe donc par trois étapes :

1. La modélisation : cette étape consiste à construire le modèle du phénomène à étudier,
2. L'expérimentation : cette étape consiste à soumettre ce modèle à un certain type de variations,
3. La validation : cette étape consiste à confronter les données expérimentales obtenues avec le modèle réel.

Selon Shannon [Shannon, 1998], la simulation peut être définie de la façon suivante :

> *" the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system ".*

Pour 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 "*.

Fishwick illustre par ailleurs ses propos à l'aide du schéma suivant (figure 1.3) :

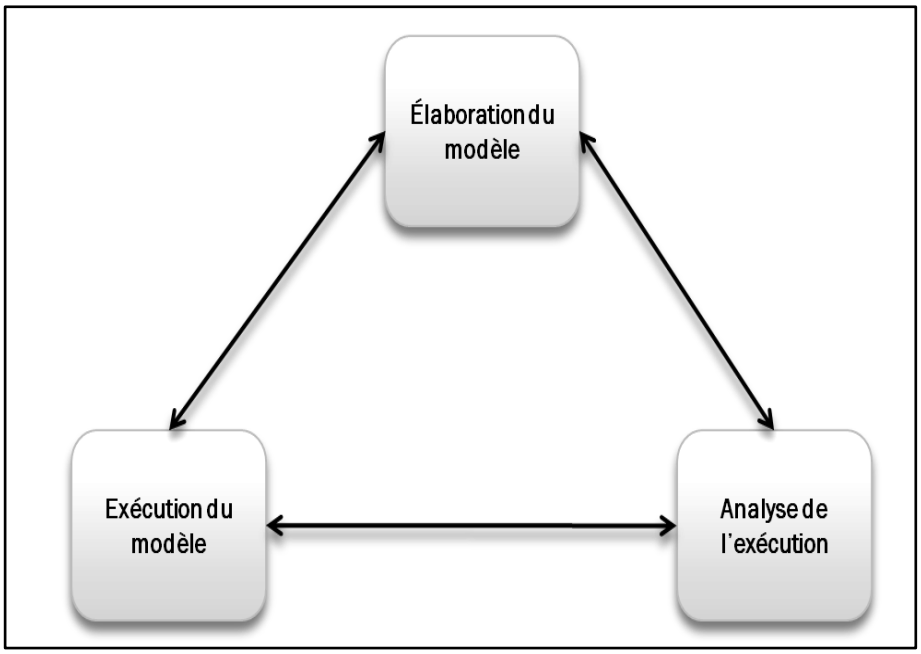

FIG. 1.3 – La simulation informatique selon [Fishwick, 1997]

Ces deux définitions proposées par Shannon et Fishwick caractérisant la simulation peuvent être interprétées comme suit : Le but d'une simulation est d'imiter un phénomène en proposant un cadre expérimental, en se basant sur une représentation formelle de la façon dont le phénomène est supposé fonctionner ; c'est le modèle du phénomène. Un simulateur permet d'effectuer des expériences [Fishwick, 1994].

La simulation informatique est l'un des outils permettant de simuler des phénomènes réels afin d'en comprendre le fonctionnement interne et/ou à en prévoir l'évolution sous certaines conditions. La simulation peut aussi servir le cadre pour la conception de systèmes en vue de leur mise en œuvre et de leur réalisation ou à des fins pédagogiques.

Zeigler [Zeigler *et al.*, 2000b] propose le schéma ci-dessous :

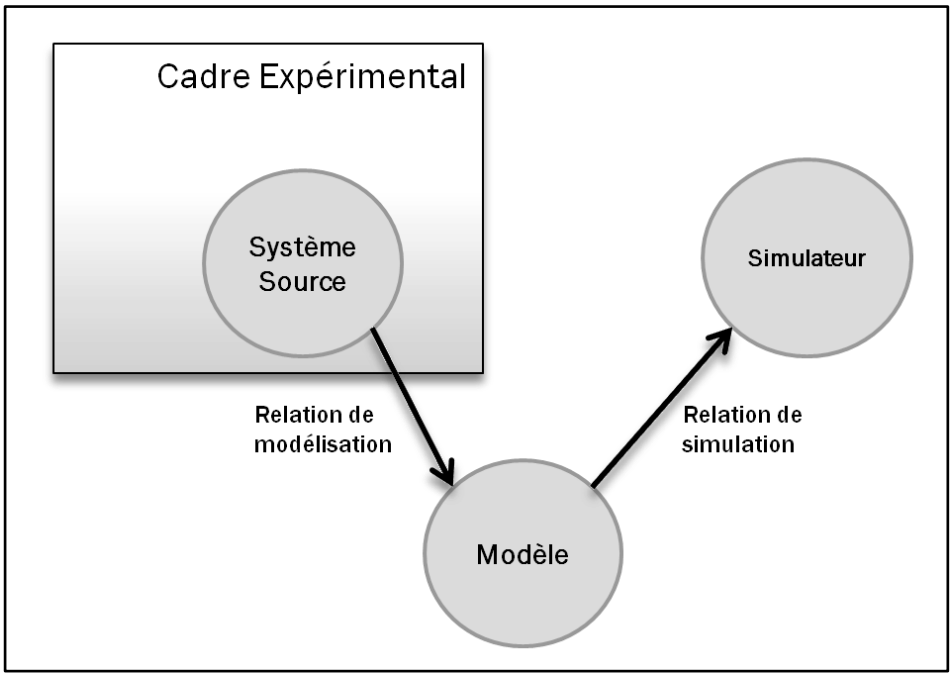

FIG. 1.4 - Relations de modélisation [Zeigler *et al.*, 2000b]

- **Système source :** le phénomène que l'on souhaite étudier.
- **Cadre expérimental :** spécifications des conditions d'observation du système et des objectifs de la simulation.
- **Modèle :** l'ensemble des instructions qui permettent de générer - à l'aide d'un programme informatique - le comportement du système au cours du temps.
- **Simulateur :** le programme informatique capable d'exécuter le modèle et de produire son comportement.

1.3.3.1 Catégories de la simulation informatique

On peut distinguer trois catégories de simulations :

- **La simulation continue** : ce type de simulation s'applique aux systèmes dynamiques, caractérisés par des changements qui se font en permanence avec le temps. Ces changements peuvent être représentés par des équations différentielles qui vont théoriquement permettre aux variables d'être calculées à tout moment.

- **La simulation discrète :** où le système est soumis à une suite d'évènements qui le changent. Ce type de simulations a vocation à appliquer des principes faciles et simples à des systèmes complexes, caractérisés par leur grande taille, l'hétérogénéité des éléments qui les composent et le nombre élevé d'interactions entre ses composants. Principalement, cette simulation se distingue en deux grandes catégories : 1) **asynchrone ou time-slicing :** dans cette catégorie on simule à chaque moment le passage d'une unité de temps sur tout le système. 2) **synchrone ou event-sequencing :** dans ce type de simulation discrète on calcule l'arrivée d'un événement suivant, et on ne simule qu'événement par événement, cette démarche permet des simulations très rapides.

- **La simulation par agents :** dans laquelle la simulation est segmentée en plusieurs entités qui interagissent entre elles. Elle est essentiellement utilisée dans les simulations sociales et économiques, dans lesquelles chaque agent décrit un individu ou un ensemble (groupe) d'individus. Par nature, son fonctionnement est asynchrone.

1.3.3.2 Le processus de simulation

La notion de simulation fait référence à l'exécution d'un modèle pour des entrées particulières et l'interprétation et l'analyse des sorties du système lorsqu'il est soumis à ces entrées. Bien que la grande majorité des définitions de la simulation s'accordent sur le fait que leur conception fait partie d'un processus particulier, aucune définition détaillant ce processus ne fait le consensus [Anderson, 1974; Gilbert et Troitzsch, 2005; Law et McComas, 1991; Shannon, 1998]. L'étude de différentes travaux et publications concernant la notion de simulation permet d'en identifier les principales étapes. Nous étudions dans la suite de ce chapitre deux de ces définitions, qui décrivent le processus de simulation selon la réalisation d'expériences d'une part et la conception du programme de simulation d'autre part.

- **Un processus expérimental**

Shannon [Shannon, 1998] définit le processus de simulation informatique comme un processus expérimental composé de plusieurs étapes qui suivent :

1. Définition du problème (Problem definition) : Définir de façon précise le but de la simulation, c'est-à-dire pourquoi étudie-t-on ce phénomène et à quelles questions désirer-t-on répondre,

2. Planification du projet (Project planning) : Déterminer si les moyens logistiques et techniques à disposition permettent de déterminer la résolution d'un tel problème,

3. Définition du système (System definition) : Déterminer les informations pertinentes à prendre en compte dans le modèle,

4. Formulation du modèle conceptuel (Conceptual model formulation) : Définir un modèle de la simulation (modèle préliminaire) qui décrit les variables, les composants ainsi que les interactions et interconnections qui constitue le système dans un pseudo-code ou d'une manière graphique,

5. Conception préliminaire de l'expérimentation (Preliminary Experimental Design) : Établir sur quels critères la simulation est évaluée, quels paramètres faire varier, comment ces paramètres faire varier et combien d'expériences il est obligatoire d'exécuter,

6. Préparation des données d'entrée (Input Data preparation) : Identifier et sélectionner les valeurs pertinentes utilisées initialement pour chaque paramètre de la simulation,

7. Traduction du modèle (Model Translation) : Implémenter le modèle dans un langage d'implémentation particulier,

8. Vérification et validation (Verification and Validation) : Vérifier que l'implémentation est appropriée au modèle et que les résultats de simulation obtenus sont appropriés à ce qui est désiré d'une manière quantitative et qualitative,

9. Conception de l'expérimentation finale (Final experimental design) : Affiner les critères qui décrivent la qualité d'une simulation ainsi que les paramètres utilisés et le nombre d'expériences à exécuter,

10. Expérimentation (Experimentation) : Exécuter l'ensemble des expériences prévues et collecter les résultats de simulation obtenus,

11. Analyse et interprétation (Analysis and Interpretation) : Analyser les résultats de simulation et déterminer si ces résultats répond au problème posé,

12. Implémentation et documentation (Implementation and Documentation) : Documenter et publier les résultats de simulation.

La figure 1.5 montre sa représentation schématique.

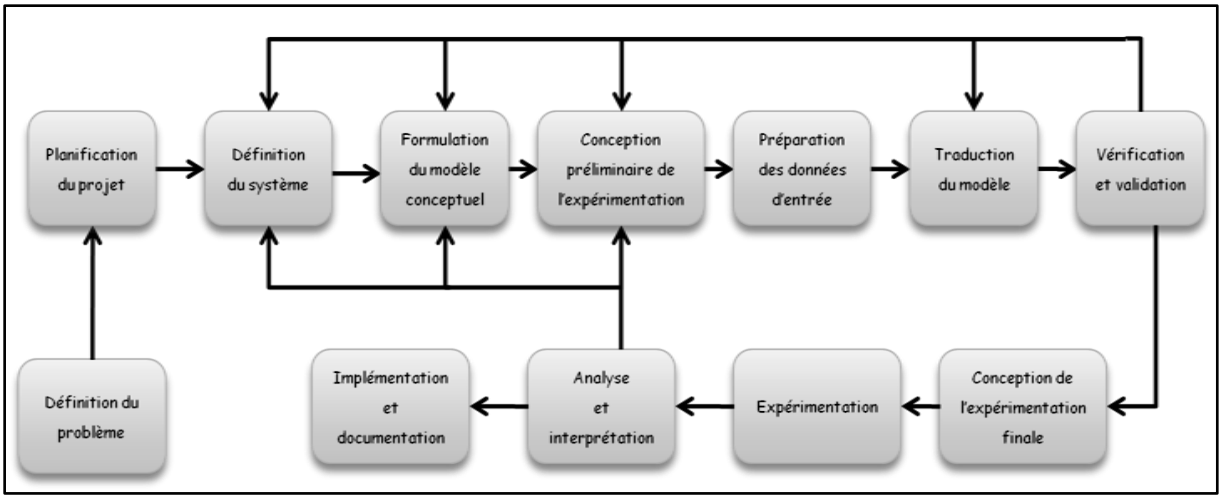

FIG. 1.5 – Schéma du processus de conception de simulations [Shannon, 1998]

- **Un processus de conception**

D'autre définitions du processus de simulation se focalisent sur une vision orientée 'expertise informatique' [Zeigler, 2000a ; Meurisse, 2004 ; Edmonds, 2005 ; Galán *et al.*, 2009]. Parmi ces définitions, celle de Galán [Galán *et al.*, 2009] (voir figure 1.6), nous paraître intéressante puisque le changement de perspective permet de mettre en valeur l'un des problèmes principaux de la simulation. Dans cette définition, le processus de simulation est composé de quatre modèles suivants :

1. Le modèle non-formel (No-formal model) : Le modèle est exprimé en langage naturel, il est décrit par des experts du domaine afin de définir l'objectif, le but, les principaux phénomènes ainsi les différentes relations causales et leurs interactions qui interviennent dans le processus de simulation,

2. Le modèle formel (Formal model) (appelé aussi modèle conceptuel) [Robinson, 2006, Sargent, 1998] : Ce modèle correspond à la réécriture du modèle non-formel en utilisant un formalisme de modélisation particulier. Lors de la phase conception, les choix effectués par les experts dans le modèle non-formel sont complétés pour qu'ils puissent être exprimés dans le formalisme choisi,

3. Le modèle exécutable (Executable model) (appelé aussi modèle computationnel [Edmonds et Hales, 2003] ou modèle de la simulation [Sargent, 1998]) : Le modèle exécutable complète le modèle conceptuel, en lui ajoutant tous les éléments qui permettront son exécution,

4. Le programme informatique (Computer program) : Est une implémentation du modèle computationnel par l'utilisation d'une plateforme de simulation ou un langage de programmation particulier.

Dans la définition de Galán, Les informations extraites de chaque modèle permettent de décrire le modèle suivant et se rapproche graduellement du programme informatique. En outre, les représentations abstraites du phénomène réel (les modèles) peuvent être en implémentation disjoints, ou décrire plusieurs parties d'un même modèle global. Il est intéressant de noter que ces modèles décrivent aussi plusieurs rôles joués par le modélisateur au cours de la simulation. Chaque rôle ne peut être joué qu'avec des compétences adéquates, variant grandement d'un rôle à un autre. [Kubera, 2010]

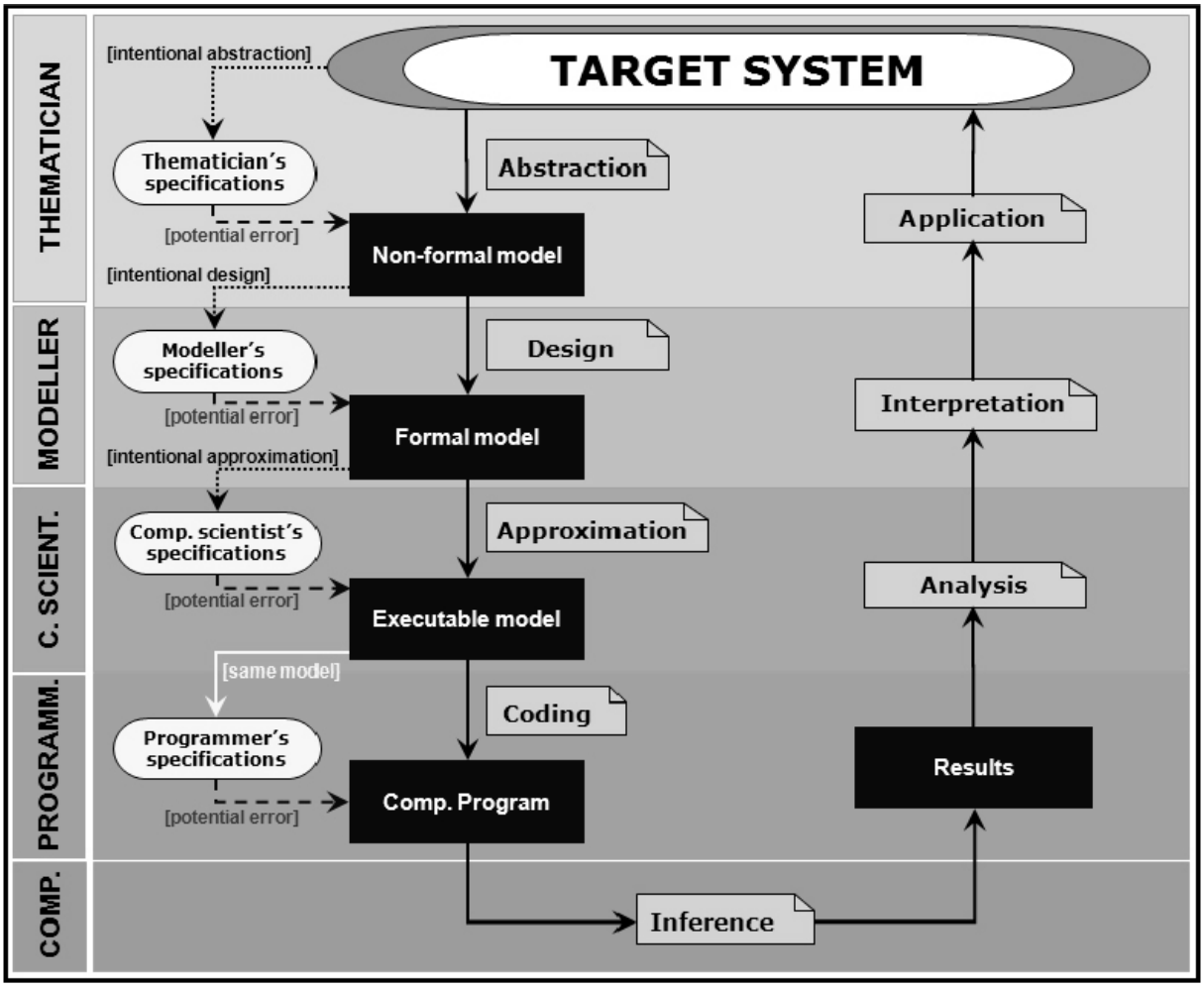

FIG. 1.6 – Les différentes étapes du processus de conception de simulations [Galán *et al.*, 2009]

1.4 Conclusion

Les notions liées aux systèmes complexes ainsi qu'à leur modélisation et simulation s'appuient sur un ensemble de définitions pour la plupart non standardisées. Actuellement, plusieurs définitions différentes existent dans la littérature bien qu'intuitivement les idées sont claires et loin d'être ambigües. Le problème qui constitue ces nuances semble être beaucoup plus lié à la grande diversité des phénomènes considérés ainsi qu'aux différentes approches, paradigmes et formalismes utilisés. Car il existe, comme nous venons de le voir dans ce chapitre une grande diversité d'outils, de moyens et de méthodes.

CHAPITRE 2

LES SYSTEMES MULTI-AGENTS (SMA)

2.1 Introduction

Nous présentons, dans ce chapitre, quelques notions de base concernant le paradigme agent, notion qui a été initialement utilisée dans la résolution de problèmes liés à l'intelligence artificielle distribuée (IAD). Un Système Multi-Agents (SMA) est un système qui comporte un ensemble d'agents (au moins un agent) qui interagissent et coopèrent à l'exécution de tâches. Nous présenterons quelques outils et environnements pour le développement de tels systèmes.

2.2 Le concept agent

Dans la littérature, plusieurs définitions ont été faites concernant la notion d'agent. Ce concept a acquis une grande généralité ce qui en rend la définition assez difficile. Pour certains, il n'est pas possible de dissocier l'agent du système multi-agents et de l'environnement dans lequel il est censé évoluer. D'autres réduisent l'agent à un simple processus agissant de façon autonome et communicante.

2.2.1 Définitions

D'après Erceau [Erceau, 1993] *"Les agents sont des entités physiques (capteurs, processeurs,...) ou abstraites (tâches à réaliser, déplacements,...), qui sont capables d'agir sur leur environnement et sur elles-mêmes, c'est-à-dire de modifier leur propre comportement. Elles disposent, pour ce faire, d'une représentation partielle de cet environnement et de moyens de perception et de communication*."

Selon Yves Demazeau [Demazeau et Costa, 1996] *" Un agent est une entité réelle ou virtuelle dont le comportement est autonome, évoluant dans un environnement qu'il est capable de percevoir et sur lequel il est capable d'agir et d'interagir avec les autres agents. "*

Selon Maes [Maes, 1995] *"An agent is a computational system that inhabits a complex, dynamic environment. The agent can sense, and act on, its environment, and has a set of goals or motivations that it tries to achieve through these actions."*

Pour Jacques Ferber [Ferber, 1995, 1999] *" On appelle agent une entité physique ou virtuelle :*

- ✓ *qui est capable d'agir dans un environnement,*
- ✓ *qui peut communiquer directement avec d'autres agents,*
- ✓ *qui est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),*
- ✓ *qui possède des ressources propres,*

- ✓ *qui est capable de percevoir (mais de manière limitée) son environnement,*
- ✓ *qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),*
- ✓ *qui possède des compétences et offre des services,*
- ✓ *qui peut éventuellement se reproduire, mourir et changer d'état*
- ✓ *dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit".*

Il ressort de ces définitions trois notions fondamentales permettant de caractériser un agent : perception, délibération et action (Figure 2.1) :

La perception : La perception permet à l'agent d'avoir conscience de l'univers dans lequel il évolue. Ainsi, l'agent peut percevoir les actions des autres agents ainsi que tout type de modification ou transformation de son environnement. On peut ainsi parler du domaine de perception de l'agent caractérisé par l'ensemble des événements ayant eu un effet sur l'environnement dans un temps donné.

La délibération : L'agent dispose d'un ensemble de connaissances ainsi que de plans et de méthodes d'exploitation de ces connaissances qui lui permettent d'induire un ensemble de raisonnements pour aboutir à la concrétisation d'objectifs. En cela, l'agent serait capable de raisonner et donc de décider en fonction de ses perceptions.

L'action : L'agent peut influer sur son environnement et sur les autres agents, il peut ainsi agir en manipulant des objets et en communiquant avec les autres agents en engageant des négociations des coopérations, des conflits ou toute autre action décidée par l'agent.

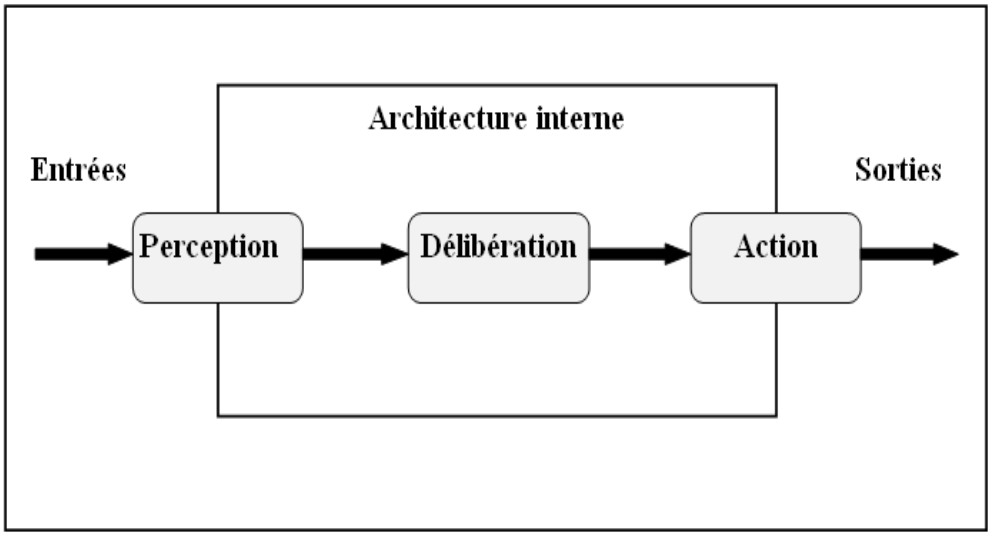

FIG. 2.1 – Processus d'un agent délibératif

En outre, Jennings et Wooldridge [Jennings et Wooldridge, 1998] définissent l'agent comme suit :*" Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu ".* (Figure 2.2)

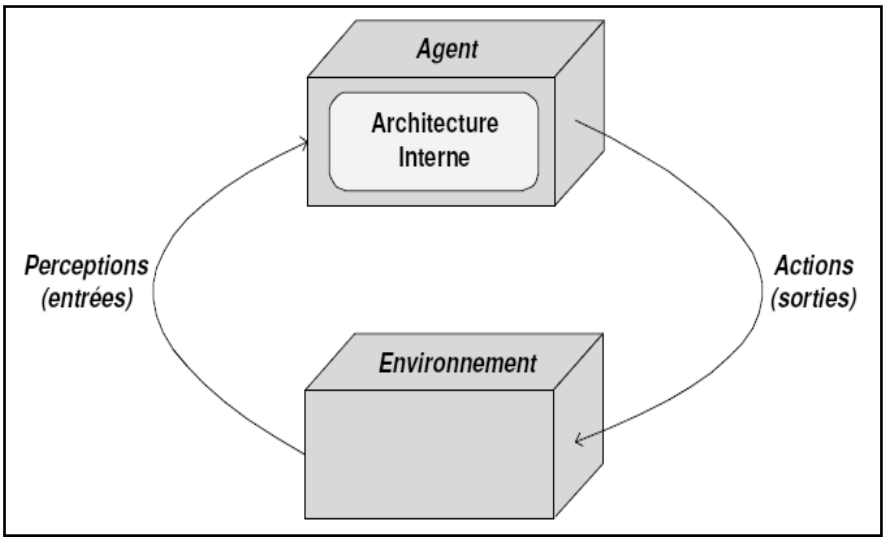

FIG. 2.2 – Agent et environnement [Wooldridge, 2000 ; Russell et Norvig, 2003]

Ainsi l'agent est :

- *Situé : qui détermine la capacité de l'agent à agir sur son environnement à partir des perceptions de ce même environnement. Exemples : contrôle de processus industriels, systèmes embarqués dans des véhicules, etc.*

- *Autonome : l'agent a des capacités propres lui permettant d'agir sans l'intervention d'autres agents. Il est capable de contrôler ses propres actions ainsi que son état interne.*

- *Flexible : l'agent est capable d'interagir avec les autres agents et exhiber un comportement proactif et opportuniste. Il est capable de prendre les initiatives nécessaires à l'atteinte de ses objectifs.*

Deloach [Deloach *et al.,* 2001] définit un agent comme un ensemble de processus qui communiquent entre eux pour atteindre un objectif donne.

Pour G.Weiss [Weiss, 1999] *" Agents are autonomous, computational entities that can be viewed as perceiving their environment through sensors and acting upon their environment through effectors"*

Pour [Jeennings et Wooldridge, 2000]**,** un agent est un système informatique encapsulé situé dans un environnement dans lequel il est capable d'effectuer une action flexible et autonome, compatible aux objectifs de la conception. Ainsi, les agents sont considérés comme des entités identifiables ayant des bornes et des interfaces bien déterminées et pouvant résoudre un ensemble de problèmes. Ces agents œuvrent dans un environnement déterminé avec lequel ils interagissent grâce à des entrées de perception (capteurs) et des sorties d'action (effecteurs).
Les agents se fixent des buts et des objectifs à atteindre. Ils sont autonomes et peuvent prendre des décisions et adopter des comportements pour résoudre les problèmes (ils sont capables, ainsi, de s'adapter aux changements d'état de leur environnement) d'où la notion de réactivité. De même, les agents sont capables d'adopter de nouveaux objectifs (notion de proactivité). Dans un système multi-agents, les agents disposent d'outils leur permettant de dialoguer, de communiquer, de coopérer de se coordonner et de négocier les uns avec les autres.

2.2.2 Les typologies des agents

On distingue trois grandes catégories d'agents : les agents cognitifs qui sont conçus comme des entités intelligentes et qui sont en général ; en nombre réduit dans un système. La deuxième catégorie comprend les agents réactifs qui sont des entités simples réagissant aux stimuli émis par leur environnement ; on trouve généralement un nombre important de tels agents dans un système multi-agents. Enfin la troisième catégorie concerne les agents hybrides qui intègrent à la fois les aspects cognitifs et réactifs des agents. Il va sans dire que c'est cette catégorie qui est la plus importante mais qui est assez complexe et gourmande quant à son implémentation.

2.2.2.1 Les agents cognitifs

Les systèmes multi-agents d'agents cognitifs sont formés d'agents coopérant pour effectuer des tâches complexes. Un tel système est formé d'un petit nombre d'agents [Dieng, 1990] disposant de grandes capacités de raisonnement ainsi que de grandes aptitudes de traitement des informations ayant trait à leur domaine de compétence ainsi que la possibilité d'obtenir des informations pour interagir avec les autres agents et l'environnement [Labidi *et al.*, 1993]. Ces agents gèrent des bases de connaissances.

Les agents BDI (Beliefs-Desirs-Intentions : Croyances-Désirs-Intentions) forment une catégorie d'agents cognitifs où chaque agent est doté d'une représentation de l'environnement

dans lequel il évolue, dispose d'un ensemble de croyances sur lui-même et sur les autres agents du systèmes et posse de un ensemble d'intentions (buts à atteindre).

Il est, ainsi, clair que l'architecture et le comportement d'un agent cognitif sont plus complexes que celles d'un agent réactif [Mandiau *et al.*, 2002].

Les agents cognitifs regroupent plusieurs sous-types d'agents tels que définis dans ce qui suit :

- **Les agents intelligents**

Les agents intelligents [Tranvouez et Espinasse, 1999] sont dotés des caractéristiques d'autonomie, de coopération et d'adaptabilité à leur plus haut niveau. Ils sont dotés de capacités de négociation avec d'autres agents ainsi que de capacités d'apprentissage leur permettant d'acquérir ou de modifier leurs connaissances. Ils planifient, ainsi, leurs actions. De ce fait, la notion même d'agent intelligent est souvent liée à capacité d'apprentissage des agents.

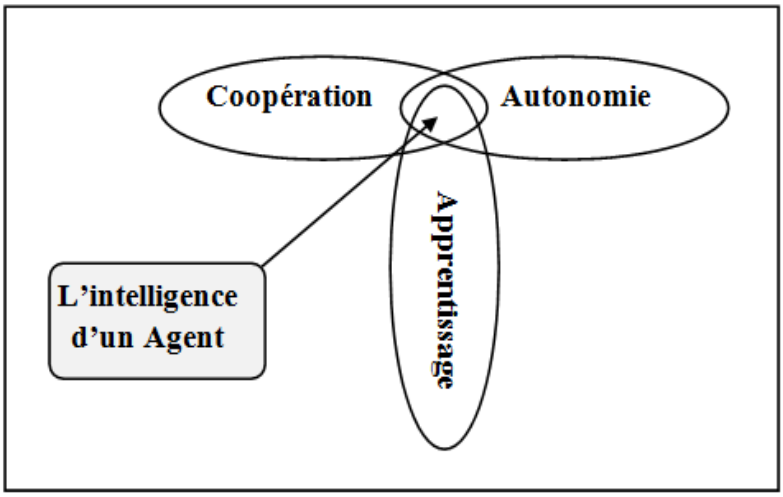

FIG. 2.3 – Agent Intelligent

- **Les agents collaboratifs**

Les agents collaboratifs [Nwana, 1996] sont des agents dotés de capacités d'autonomie ainsi que de possibilités de collaboration et de communication avec d'autres agents en vue de la résolution des problèmes. Le domaine de l'Intelligence Artificielle Distribuée (IAD) est un domaine de prédilection pour ce type d'agents. De manière plus générale, ce type d'agents est utilisé pour la résolution de problèmes distribués. La conception de ce type d'agents se heurte à plusieurs difficultés qui concernent la coordination des actions, la définition des protocoles de communication entre les agents, la négociation entre les agents pour l'allocation des tâches, le partage des ressources (qui sont limitées), la synchronisation des actions, etc.

- **Les agents interface**

Ce type d'agent est, de façon générale, conçu comme un assistant personnel doté de capacités d'autonomie assez faibles [Maes, 1995]. Les agents interface utilisent des techniques d'apprentissage pour effectuer un ensemble de taches pour l'utilisateur. Comme, par exemple, e-commerce, filtrages d'e-mails, faciliter et accélérer quelques tâches spécifiques de l'utilisateur,… La plupart du temps, les agents interface observent et enregistrent les actions de l'utilisateur. Ils pourront, par la suite, lui suggérer une meilleure façon d'effectuer un travail. Les modes d'apprentissages de ces agents peuvent se faire de différentes manières. Parmi les plus courantes, on retrouve l'apprentissage par observation ou par imitation de l'utilisateur, l'apprentissage par réception de *feedback* (négatif ou positif) de l'utilisateur ainsi que par réception d'instructions explicites de l'utilisateur (Figure 2.4).

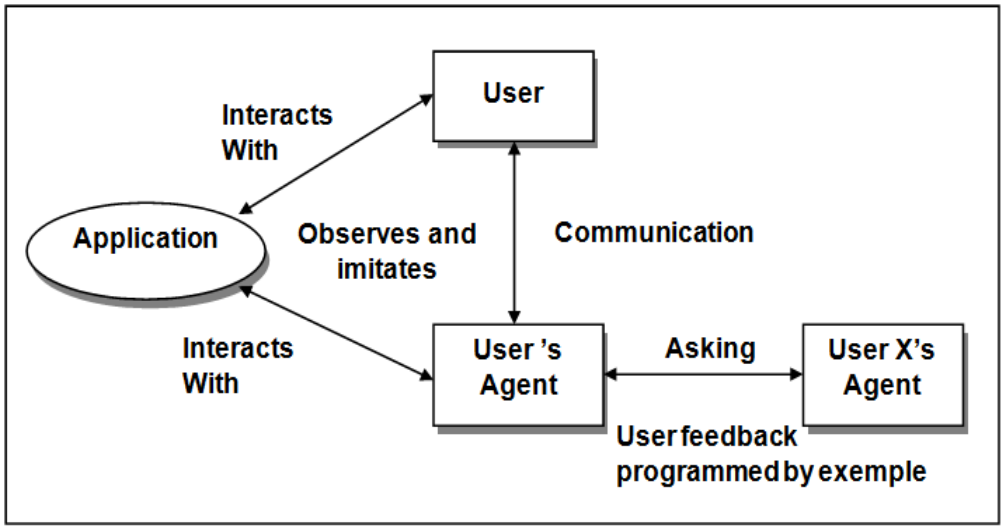

FIG. 2.4 – Architecture d'un agent interface simple

- **Les agents information**

Ce type d'agent [Rhodes, 2000 ; Rus *et al.*, 1997] est très populaire pour retrouver et analyser de grandes quantités d'information [Klusch, 1998]. Les agents information traitent de grandes quantités d'informations qu'ils vont chercher sur des pages Web, sur des serveurs ou dans des bases de données. Les moteurs de recherche les utilisent pour indexer des sites Internet, pour vérifier les mises à jour des sites, etc. De plus, ils apportent une aide précieuse aux utilisateurs qui recherchent des informations sur un sujet, un service ou un bien en particulier. Ces agents récupèrent les informations, les filtrent, les classifient et les renvoient à l'usager qui peut déterminer plus rapidement et facilement si les informations recueillies contiennent les renseignements désirés.

- **Les agents BDI**

L'architecture BDI [Rao et Georgeff, 1992] acronyme qui signifie, en anglais, Beliefs, Desires, Intentions. Ce qui se traduit en français par croyances, désirs et intentions, est une autre approche utilisée dans la conception des agents cognitifs. Les agents se basent donc sur ces trois aspects pour choisir leurs actions. La figure 2.5 donne un exemple d'architecture BDI.

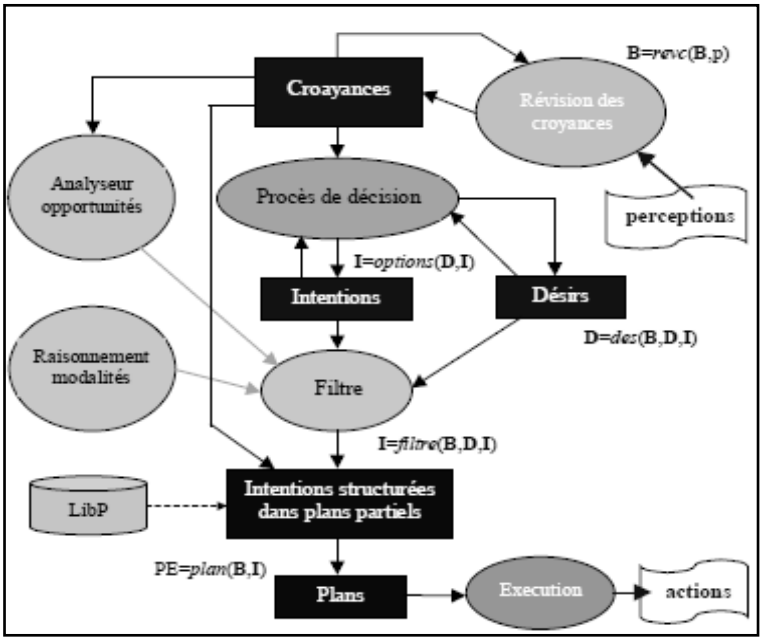

FIG. 2.5 – Un exemple d'architecture BDI.

Différentes formes et modèles d'apprentissage et de coopération des différents types d'agents cognitifs sont résumés dans la figure 2.6. Les agents 'collaborant' sont dotés d'autonomie et sont coopérants, leur adaptabilité et leur possibilité de modification de leur comportement leur permet des capacités de négociation. Les agents interfaces sont ceux qui sont les plus sollicités par l'utilisateur, de ce fait, Ils s'adaptent à son comportement, en proposant différentes formes de représentation. Ils peuvent aussi, exhiber un ensemble de déductions sur l'état de l'utilisateur et proposer des interfaces appropriées à chaque type d'utilisateur notamment dans les cas d'apprentissage automatique. Ceci tout en communicant des informations à d'autres agents du système. Les agents information, qui sont très utilisés dans les moteurs de recherche du web peuvent agir de façon très autonome pour la recherche des informations au travers de différents sites, mais souvent indépendamment des autres agents. Les notions de rapidité, de disponibilité, de cohérence et de fiabilité de l'information sont

essentielles dans ce cas. Ces agents sont capables d'adapter leur stratégie de recherche en fonction des informations trouvées ou non trouvées.

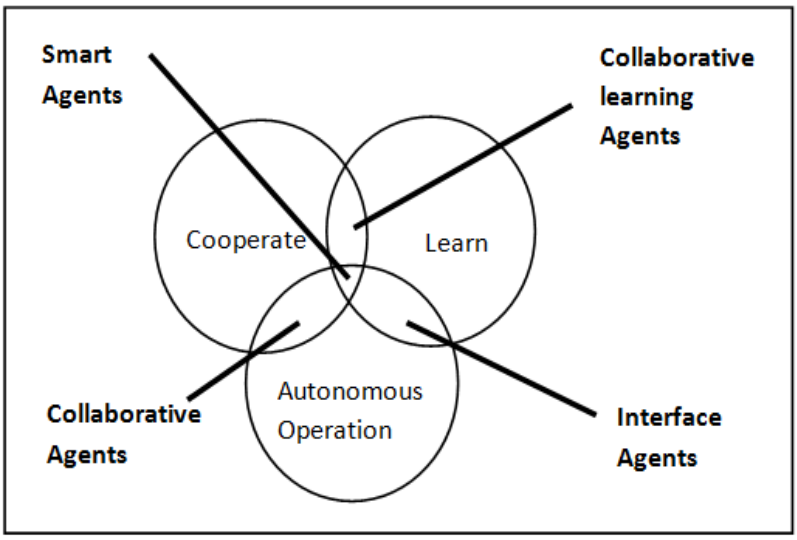

FIG. 2.6 – Degrés d'autonomie, de coopération et d'adaptabilité des principaux agents cognitifs [Nwana, 1996]

2.2.2.2 Les agents réactifs

Les agents réactifs [Ferber, 1995] sont des agents dont la structure et le comportement sont moins complexes que les agents cognitifs. Ces agents réagissent uniquement à leurs perceptions de l'environnement et agissent en fonction de cette perception. Ce type d'agents n'est pas qualifié d'intelligent vu que leur principe de fonctionnement est basé particulièrement sur la notion de stimulus/action. Ce principe permet aux agents d'agir grâce à des réflexes totalement conditionnés. A la perception d'un stimulus particulier, l'agent fournit une réponse bien déterminée. La communication entre agents réactifs et l'environnement est très rudimentaire. Certaines de leurs caractéristiques telles que leur simplicité de conception ainsi que l'émergence de comportements globaux restent, néanmoins, intéressantes [Drogoul, 1993]. Notons que plusieurs travaux utilisant des agents réactifs ont été appliqués en robotique [Brooks, 1989 ; Steels, 1994], dans le domaine de la planification [Gallone *et al.*, 1994] ou la résolution collective de problèmes [Deneubourg *et al.*, 1991]. On utilise parfois le terme d'animats.

L'architecture de *subsomption* proposée par Brooks (figure 2.6) est une architecture utilisée pour la conception d'agents réactifs [Brooks et Connell, 1986]. Ainsi le processus de la prise de décision est basé sur la consultation d'un ordre de priorité entre des modules qui représentent les tâches pouvant être accomplies par l'agent. Cet ordre de priorité fixe les

conditions d'activation qui sont calculées en fonction des perceptions et elles sont alors évaluées suivant cet ordre de manière à sélectionner la tâche à effectuer en priorité. Par exemple, pour un robot, il est vital de recharger ses batteries si elles faiblissent. Cette tache devient alors prioritaire.

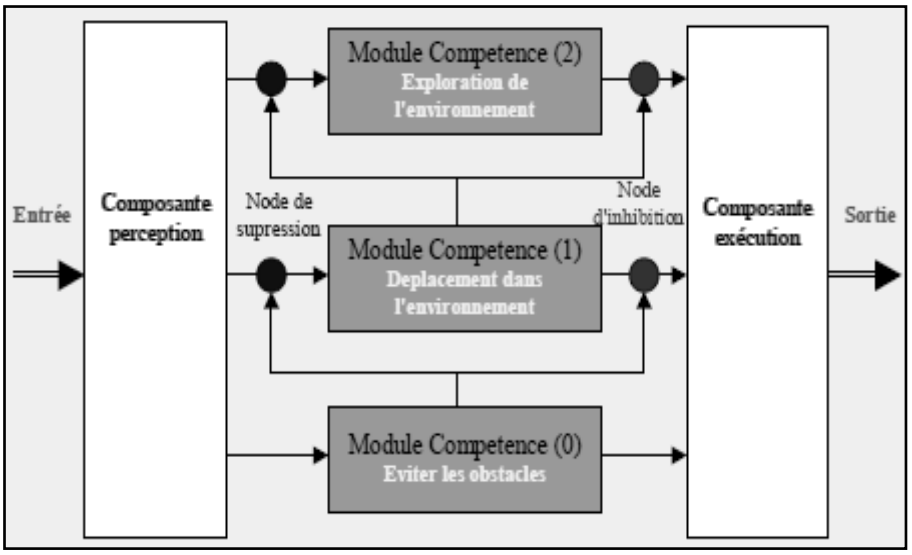

FIG. 2.6 – Un exemple de l'architecture de subsomption.

Le tableau 2.1 suivant résume un ensemble de différences entre les agents cognitifs et les agents réactifs.

Système d'agents cognitifs	Système d'agents réactifs
Représentation explicite de l'envirennement	Pas de representation explicite
Peut tenir compte de son passé	Pas de mémoire de son historique
Agent complexe	Fonctionnement stimulus/action
Petit nombre d'agents	Grand nombre d'agents

TAB. 2.1 – Différences entre agents cognitifs et agents réactifs [Reichgelt, 1990]

2.2.2.3 Agents hybrides

Les agents hybrides [Jennings *et al.*, 1998] définissent un type d'agent supplémentaire qui reprend les propriétés et les caractéristiques des agents réactifs et celles des agents cognitifs. Ce type d'agent forme une entité capable de se comporter encore différemment. L'architecture de ce type d'agents est formée généralement de plusieurs couches logicielles. Ces couches peuvent être arrangées selon une structure verticale (une seule couche a accès aux capteurs et aux effecteurs) ou horizontale (toutes les couches ont accès aux entrées et aux sorties), (voir figure 2.7).

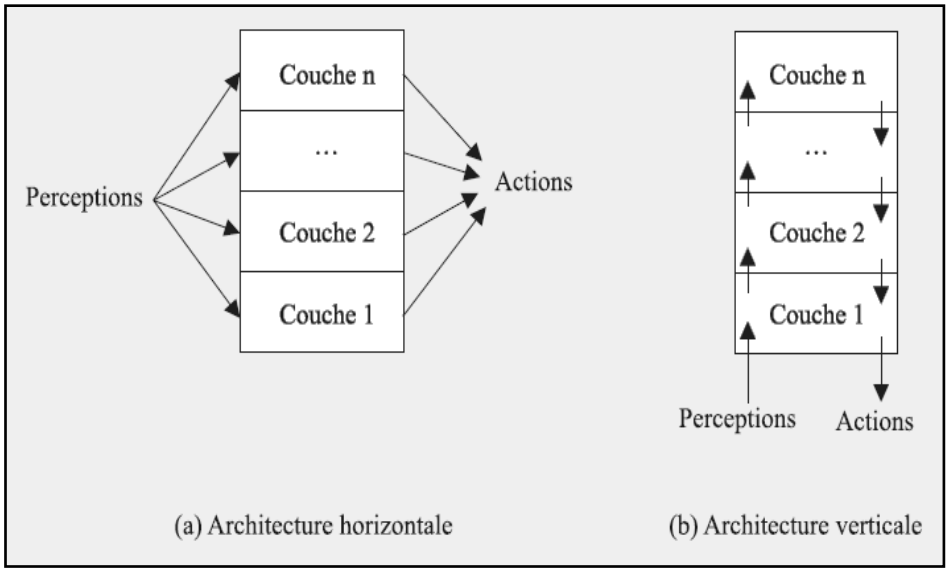

FIG. 2.7 – Architectures d'agents en couches [Jennings *et al.*, 1998].

2.2.3 Architectures d'agents

Il existe plusieurs façons de concevoir l'architecture d'un agent. Globalement, l'agent peut-être vu comme une entité dotée d'un ensemble de fonctions qui lient les entrées (perceptions) aux sorties (actions) tel que défini dans la figure 2.1. La perception de l'environnement dans lequel évolue l'agent se fait grâce à un ensemble d'entrées (capteurs) et il influe sur ce même environnement à l'aide de ses sorties (effecteurs). Une manière de différencier les architectures d'agents consiste à distinguer la façon avec laquelle les perceptions sont liées aux actions **[**Chaib-draa et Jarras, 2002**]**.

On peut distinguer les architectures d'agents suivantes pouvant être regroupées en agents réactifs et agents cognitifs :

a- Les agents réactifs.

1- Les agents à réflexes simples,
2- Les agents réflexes à états.

b- Les agents cognitifs.

1- Les agents à base de buts,
2- Les agents à base d'utilité,
3- Les agents du type BDI (Beliefs-Desires-Intentions).

Les sections suivantes détaillent ces types d'architectures.

2.2.3.1 Les agents à réflexe simple

L'agent à réflexe simple [Russell et Norvig, 1995] est le type d'agent le plus simple. Ainsi, ce type d'agent ne fait que réagir à des stimuli provenant de son environnement. L'architecture de ces agents est dépourvue de toute représentation de leur environnement. Ils ne disposent d'aucune base de connaissances et d'aucune forme de mémoire. Trois composantes caractérisent ces agents (figure 2.8) :

- Les capteurs (senseurs) qui servent à percevoir les changements opérés sur leur environnement.

- Un ensemble de règles et de conditions (ou une fonction) qui déterminent les actions appropriées à effectuer lorsqu'un changement se produit dans l'environnement.

- Les effecteurs qui leur permettent d'agir sur leur environnement.

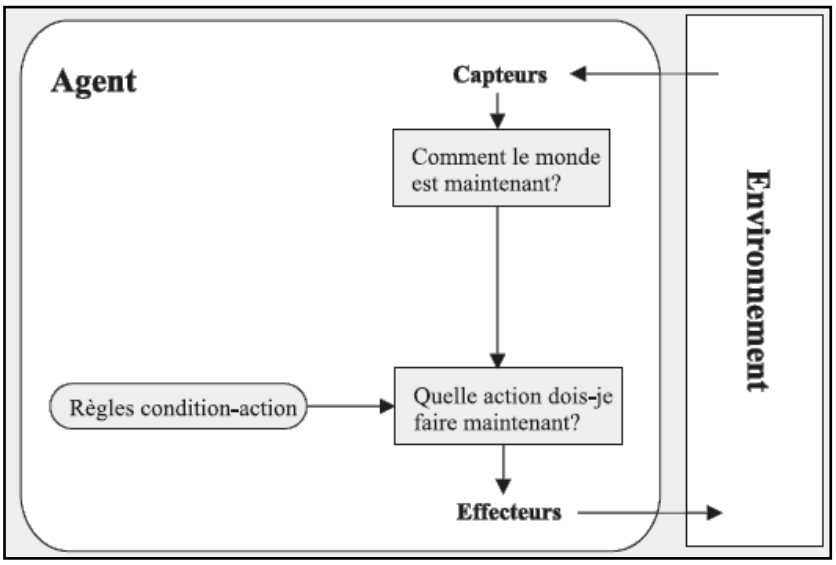

FIG. 2.8 – Agent à réflexes simples [Russell et Norvig, 1995]

2.2.3.2 Les agents réflexes à états

Ce type d'agents présente une version améliorée de l'architecture des agents réflexe simple [Russell et Norvig, 1995]. En plus des capteurs, des effecteurs et des règles (ou fonctions), ce type d'agent possède différents états et il peut prédire les résultats des actions qu'il peut exécuter.

Ces agents peuvent donc mieux déterminer quelle action poser en fonction des stimuli qu'ils perçoivent de leur environnement grâce à leurs capteurs.

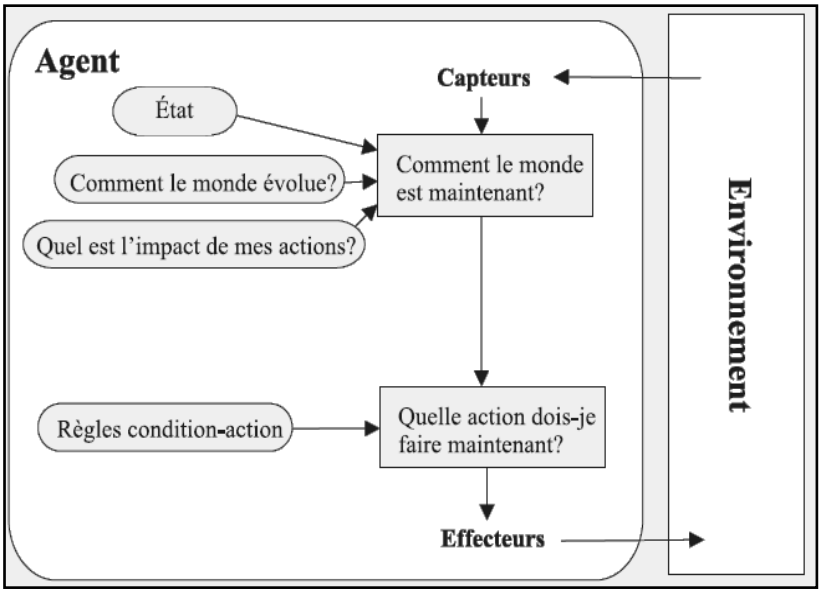

FIG. 2.9 – Agent réflexe avec états [Russell et Norvig, 1995]

2.2.3.3 Les agents à base de buts

En plus des composantes des agents réactifs précédents, ces agents connaissent les conséquences de leurs actions sur leur environnement. Ce type d'agent agit en fonction de l'atteinte d'un but [Russell et Norvig, 1995]. De cette façon, lorsqu'un tel agent détermine que l'exécution de telle ou telle action lui permettra d'atteindre son but, alors il exécutera cette action (lorsque cela est possible bien entendu).

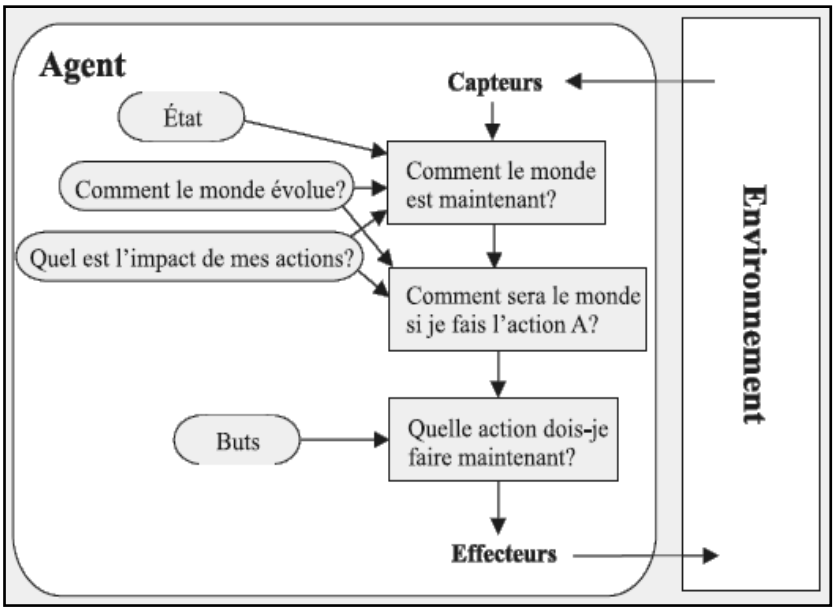

FIG. 2.10 – Agent à base de buts [Russell et Norvig, 1995]

2.2.3.4 Les agents à base d'utilité

Les agents à base d'utilité possèdent des fonctions qui leur permettent d'évaluer l'utilité de chaque action. Cette notion d'utilité d'une action peut être définie comme étant la distance déterminant leur rapprochement du but, dépendamment de l'action posée. Ces agent peuvent déterminer les actions à effectuer pour se rapprocher le plus possible du but à atteindre. Ainsi, si un agent ne peut pas atteindre son but en exécutant une action donnée, alors il va élire l'action qu'il jugera la plus appropriée à le rapprocher du but à atteindre. Cette sélection de la nouvelle action se fait grâce à une fonction dite fonction d'utilité [Russell et Norvig, 1995].

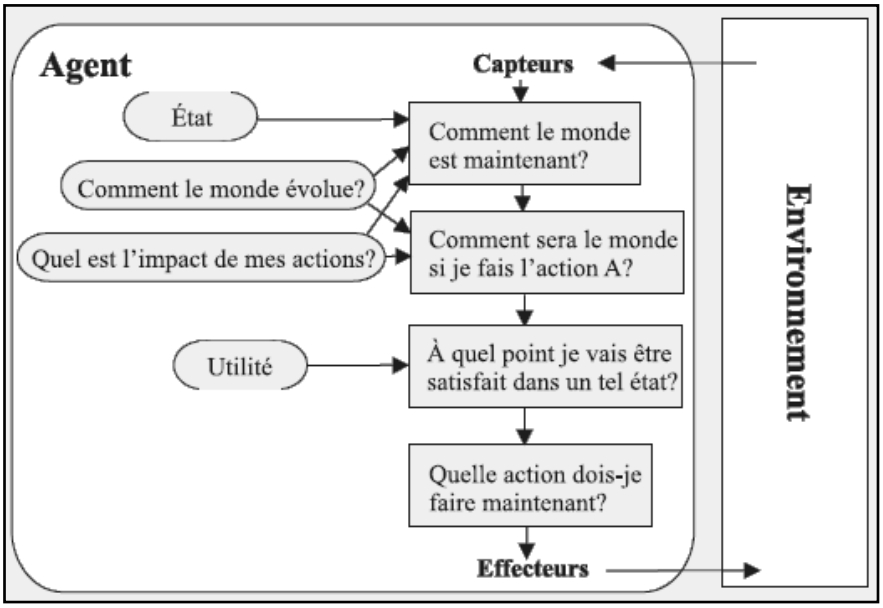

FIG. 2.11 – Agent à base d'utilité [Russell et Norvig, 1995]

2.2.3.5 Les agents B-D-I

Les agents B-D-I [Rao et Georgeff, 1992] sont des agents cognitifs de complexité supérieurs à tous les agents vus précédemment. Leurs architectures sont assez élaborées et ils représentent une catégorie d'agent ayant des propriétés d'intelligence et d'autonomie très élaborées. Ainsi, ce genre d'agent possède des bases de connaissances ainsi que des possibilités d'exploitation fine de ces bases. Ces agents évoluent selon un ensemble de plans de recherche en ayant des connaissances sur les autres agents (croyances) et leur environnement. Ils élaborent des plans selon leurs souhaits et intentions. Nous avons présenté sommairement ces agents dans la section 2.2.2.1 précédente.

2.2.4 Caractéristiques des agents

Plusieurs caractéristiques et propriétés sont liées à la notion d'agent, nous pouvons en identifier les suivantes (figure 2.13) :

- *Autonomie :* les agents sont capables de contrôler leurs états et leurs actions. Cette notion est centrale pour l'agent qui désigne, en fait, que l'agent peut évoluer, réagir et atteindre ses buts sans l'intervention humaine ou d'autres agents.
- *Réactivité :* Les agents perçoivent leurs environnements et réagissent aux changements qui s'y produisent dans les temps requis ;
- *Initiative* : face à des situations inattendues et pour atteindre les buts assignés, l'agent adopte des comportements et produit des actions qui ne sont pas seulement des réponses à leurs environnements.
- *Habilité sociale* : L'agent est une entité sociale dans le sens où pour satisfaire ses buts un agent peut demander la collaboration d'autres agents à qui il délègue ou avec lesquels il partage la réalisation de tâches. Pour cela il a besoin de coordonner, négocier soit interagir avec les autres agents et l'environnement.
- *Raisonnement* : L'agent est doté d'un pouvoir de décision qui lui permet de se fixer les buts à poursuivre et les évènements auxquels il doit réagir pour atteindre un objectif précis ou abandonner voire suspendre des buts pour se dédier à d'autres.
- *Apprentissage* : Cette notion détermine l'adaptation progressive de l'agent aux changements qui s'opèrent dans ses environnements dynamiques.
- *Mobilité* : Dans un réseau, les agents peuvent être appelé à se déplacer (migrer) d'un nœud à un autre pour des applications distribuées tout en préservant leur état lors de sauts entre nœuds**.**

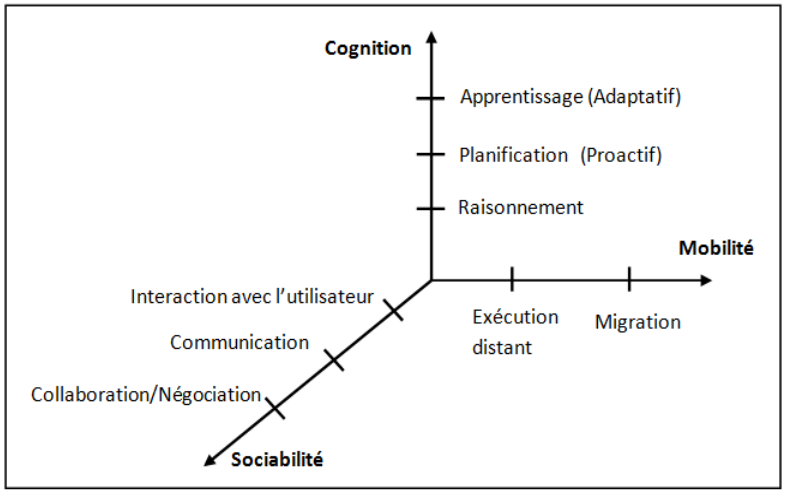

FIG. 2.13 – Caractéristiques des agents [Pavon, 2006]

2.3 Les Systèmes Multi-Agents (SMA)

2.3.1 Définition

Dans la littérature, il existe plusieurs définitions des systèmes multi-agents. Dans ce qui suit ; nous présentons quelques-unes afin de montrer les différents aspects des SMA et leurs caractéristiques les plus importantes.

D'après [Chaib-Draa *et al.*, 2001] : un système multi-agents est un système distribué composé d'un ensemble d'agents interagissant, le plus souvent, selon des modes de coopération, de concurrence et/ou de coexistence.

Selon Yves Demazeau [Demazeau, 1995] : un SMA est un système compose d'agents qui évoluent dans un environnement et interagissent de manière organisée. Ainsi, un SMA peut être représenté comme suit :

$$SMA = <Agents + Environnement + Interactions + Organisations\ (AEIO)> \qquad (2.1)$$

• **Agent** : définition des modèles ou des architectures des composants du système.

• **Environnement** : milieu dans lequel sont plongés les agents, composé d'objets qui sont perçus et manipulés par les agents, et qui obéit à un ensemble de lois.

• **Interactions** : ensemble des infrastructures (canaux de communication, langages et protocoles de communication) entre les agents et l'environnement.

• **Organisation** : topologie et structure des agents en groupes, hiérarchies, relations, etc.

J.Ferber [FERBER, 1995] (figure 2.14) : définit un système multi-agents (ou SMA) *comme étant composé*

1- D'un environnement (E) qui est un espace disposant généralement d'une métrique.

2- D'un ensemble d'objets (O), ces objets sont situés dans l'environnement (E) et à ces objets sont associés des positions dans (E). (E) agit donc comme un repéré pour les objets (O). Les agents peuvent agir sur ces objets en les créant, les détruisant, les percevant et en les modifiant.

3. Un ensemble (A) d'agents, qui représentent les entités actives du système et qui peuvent être perçues comme étant des objets particuliers.

4. un ensemble de relations (R) entre (O) et (A).

5. un ensemble d'opérations (Op) qui représentent les opérations de base que peuvent effectuer les agents telles que : percevoir, produire, consommer, transformer et manipuler des objets de (O).

6. Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette technique de modification, que l'on appellera les lois de l'univers.

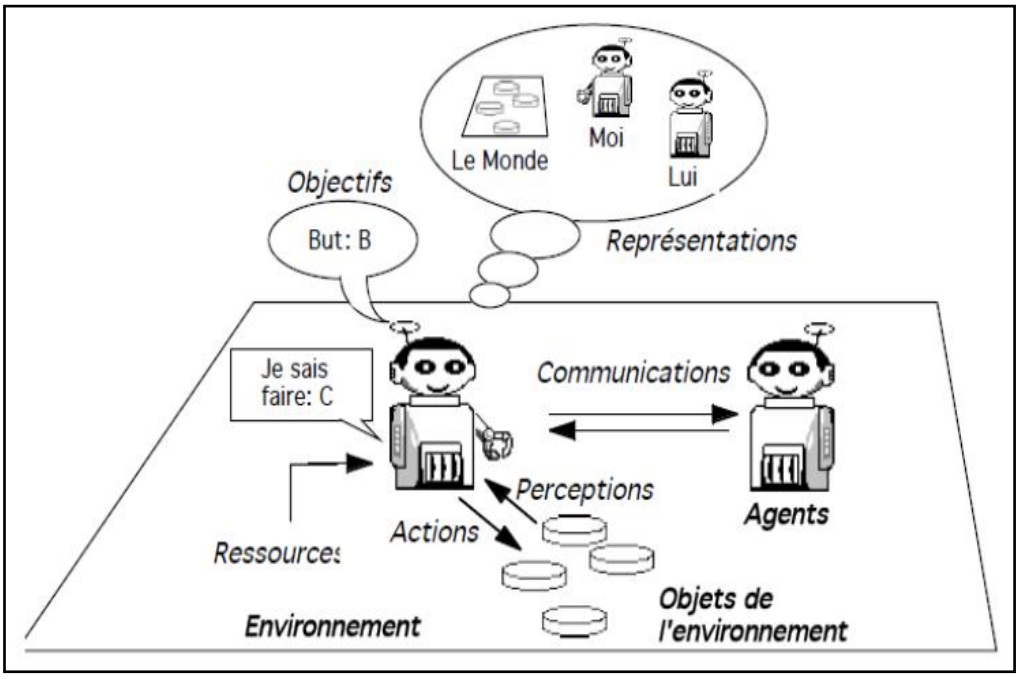

FIG. 2.14 – Représentation d'un système multi-agents [Ferber, 1995]

Une autre façon de formaliser un agent d'après [Ferber, 1997 ; Michel *et al.*, 2009], consiste à représenter un système multi-agent par le couple <*A*, *W*> où A est un agent et W un environnement tels que décrits figure 2.15 :

$$A = <Pa, Percept_a, F_a, Infl_a, S_a> \qquad (2.2)$$

Avec

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

$$W = <E, \Gamma, \Sigma, R> \qquad (2.3)$$

Avec

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

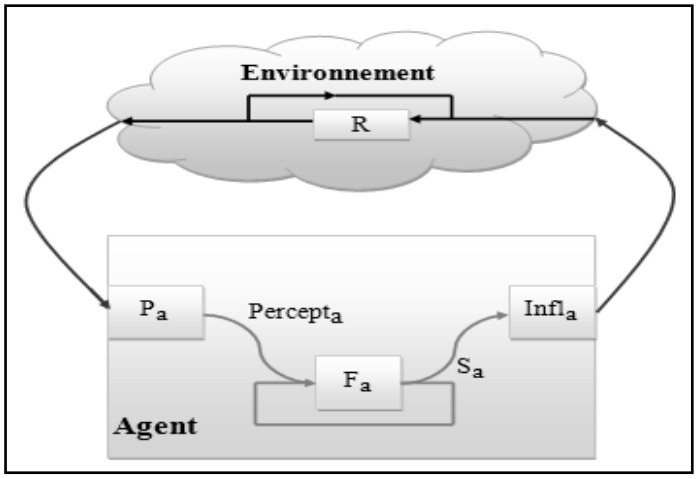

FIG. 2.15 – Modèle influence réaction

2.3.2 Caractéristiques d'un système multi-agents

D'après [Jarras et Chaib-draa, 2002] un système multi-agents est caractérisé par les faits suivants :

- Chaque agent accède à un ensemble d'informations et dispose de capacités de résolution de problèmes limitées, ainsi chaque agent dispose d'une un vue partielle de l'environnement dans lequel il évolue et des autres agents du système,
- Il n'y a pas de contrôle global du système multi-agents,
- Les données sont décentralisées.

De même d'autres caractéristiques peuvent être liées aux agents [Boissier *et al.*, 2004] :

- Distribution : l'élément de base du système est l'agent. Le système est modulaire.
- Autonomie : un agent prend ses propres décisions en fonction de ses buts, de ses objectifs et en fonction de ses capacités et connaissances. Son activité est permanente et il est en évolution permanente du point de vue apprentissage.
- Décentralisation : les agents agissent et réagissent de façon indépendante, il n'y a pas de décisions centrales valables pour tout le système.
- Échange de connaissances : les agents sont capables de s'échanger des informations entre eux en utilisant des langages et des protocoles plus ou moins élaborés.
- Interaction : les agents ont des influences les uns sur les autres ce qui permet d'engendrer des comportements menant à la réalisation de buts communs.
- Organisation : les structures matérielles et logicielles des agents créent des relations entre les agents et le réseau de ces relations forme une organisation qui peut évoluer au cours du temps.

- Situation dans un environnement : l'environnement dans lequel évoluent les agents crée les agents une source de données, de contraintes et d'incertitudes. C'est un lieu d'actions et d'influences entre agents. L'évolution du SMA est la combinaison des évolutions des agents et de l'environnement.
- Ouverture : le système est en continuelle mutation et évolution, de nouveaux agents peuvent être crées d'autres détruits dans le SMA ou encore certains peuvent être modifiés.
- Émergence : dans un SMA, la notion d'émergence est primordiale. Cette notion peut être vue comme étant la résultante des comportements et actions individuels des agents qui ne possèdent qu'une vue partielle du système et dont le comportement global mène à la résolution d'un problème distribué.
- Adaptation : spécifier le but global du système est une tâche difficile et ardue. Cependant, le système adapte son comportement à l'environnement en cours de fonctionnement, et offre une robustesse de ce comportement, à défaut d'une optimisation.
- Délégation : le contrôle de l'application est délégué aux agents du système.
- Personnalisation : un agent du système peut être amené à jouer un rôle qui lui est propre.
- Intelligibilité : les SMA sont considères comme des outils permettant de modéliser des systèmes en proposant leurs découpages en entités 'maitrisables' et en proposant des applications pour les appréhender de façon simple et naturelle.

2.3.3 Architectures d'un SMA

On distingue deux types d'architectures pour les systèmes multi-agents (figures 2.16 et 2.17).

- **Les SMA à contrôle distribué :** dans ce type d'architecture, les agents peuvent communiquer directement entre eux par le mécanisme d'envoi de messages. Le langage d'acteurs est la technique la plus utilisée pour la mise en œuvre de ce type d'architecture, qu'il est caractérisé essentiellement par :
  - Distribution total des connaissances et du contrôle.
  - Communication par envoi de message.
  - Traitement local.

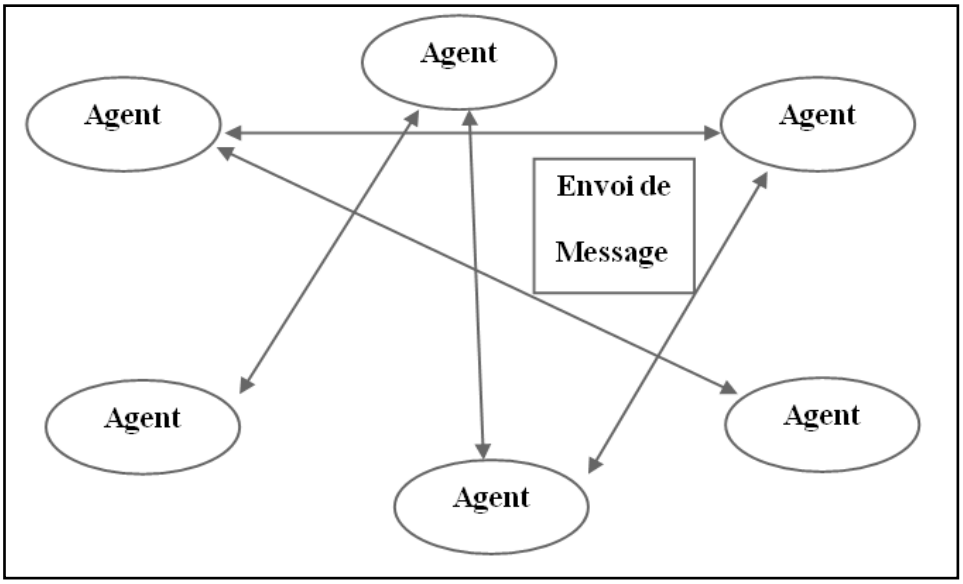

FIG. 2.16 – SMA à contrôle distribué

- **Les SMA à contrôle centralisé**

Dans ce genre de contrôle, les agents se partagent les données. Ce concept est assez simple et puissant. Dans ces systèmes, les agents ne se communiquent pas directement les données entre eux. Ils envoient et obtiennent des données grâce à un tableau noir. Ce tableau garde et partage les données pour les agents du système (Knowledge Source (KS)). Les conflits d'accès aux ressources entre ces agents sont gérés par un contrôleur.

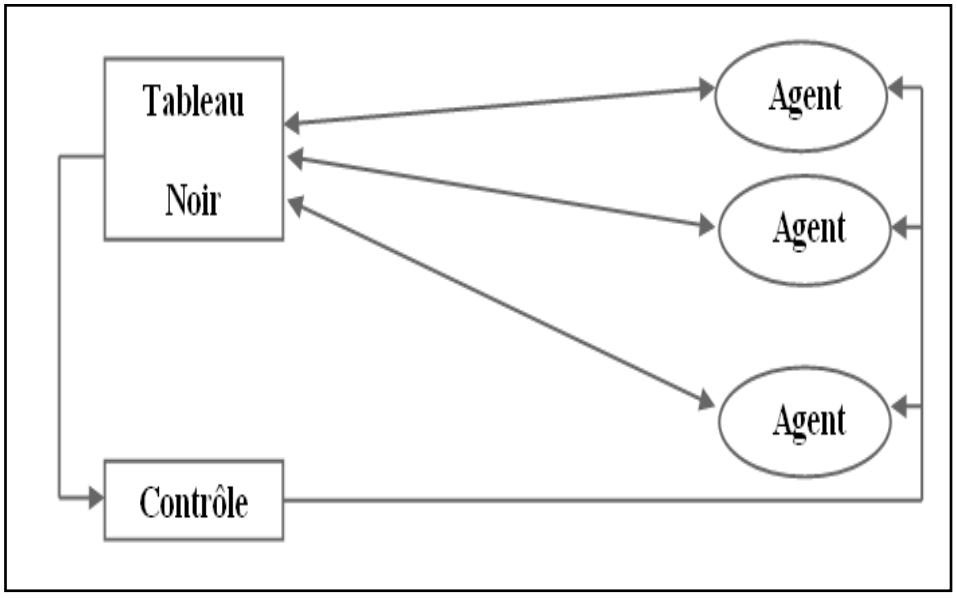

FIG. 2.17 – SMA à contrôle centralisé

Dans la pratique, pour certaines applications nécessitant la mise en œuvre des SMA, il est courant de recourir à des architectures hybrides combinant contrôle centralisé et distribué.

2.3.4 Interaction entre agents

On trouve dans [Ferber, 1999] la définition suivante de l'interaction :
''Une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques''

Toujours d'après [Ferber, 1999], la notion d'interaction suppose :
a. La présence d'agents aptes à agir et/ou à communiquer.
b. Des situations susceptibles de servir de point de rencontre entre agents.
c. Des éléments dynamiques permettant des relations locales et temporaires entre agents : communication, choc, champ attractif ou répulsif, etc.

L'interaction est par nature distribuée et elle a lieu généralement entre une entité informatique autonome (un agent) et son environnement.

L'interaction est importante dans la mesure où c'est à travers elle que les agents peuvent combiner les efforts et s'entraider ou bien entrer en compétition et négocier [Chaib-Draa et Demolombe, 2002]. En effet, dans un SMA, les agents peuvent coexister, être en compétition ou coopérer.

- **Situations d'interactions**

Parmi les différentes formes d'interactions, les plus visibles sont la collaboration qui implique la répartition du travail entre un ensemble d'agents et la coordination d'actions qui implique synchronisation entre agents qui agissent sur des sites différents.

[Ferber, 1995] définit une situation d'interaction comme '*un ensemble de comportements résultant du regroupement d'agents qui doivent agir pour satisfaire leurs objectifs en tenant compte des contraintes provenant des ressources plus ou moins limitées dont ils disposent et de leurs compétences individuelles*'.

Le tableau 2.2 montre une classification des situations d'interaction selon Ferber [Ferber, 1995] en fonction de la compatibilité des buts des intervenants, de la disponibilité des ressources et de la capacité des agents à atteindre leur but en termes de compétences individuelles.

Buts	**Ressources**	**Compétences**	**Type de situation**	**Catégorie**
compatibles	suffisantes	suffisantes	indépendances	indifférence
compatibles	suffisantes	insuffisantes	collaboration simple	coopération
compatibles	insuffisantes	suffisantes	encombrement	
compatibles	insuffisantes	suffisantes	collaboration coordonnée	
incompatibles	suffisantes	suffisantes	compétition individuelle pure	antagonisme
incompatibles	suffisantes	insuffisantes	compétition collective pure	
incompatibles	insuffisantes	suffisantes	conflits individuelle pour des ressources	

TAB. 2.2 – Classification des situations d'interaction [Ferber, 1995]

Dans ce qui suit, nous allons aborder de façon brève les notions de coopération, de négociation, de coordination et de communication.

- **La communication**

La communication dans un système multi-agents fournit un ensemble d'outils et de mécanismes nécessaires aux interactions entre agents et à l'organisation de ces derniers. La communication permet aussi la résolution coopérative des problèmes distribués [Vernad et Azena, 1992]. C'est aussi grâce à la communication entre agents qu'il est possible d'assurer la synchronisation des actions des agents et de résoudre les conflits de ressources et de buts par la négociation. Les agents sont incapables de coopérer, de négocier, de coordonner leurs actions ou de réaliser des tâches en commun sans possibilités de communication. On distingue, généralement, deux modes de communication :

- La communication se faisant de façon indirecte par transmission d'informations via l'environnement ou communication indirecte.

- La communication directe qui se base sur les échanges de messages entre les agents et qui prend en considération trois éléments essentiels :

  a) Le langage de communication : Il fournit les outils linguistiques nécessaires à la structuration des messages qui sont échangés entre les agents. Parmi les langages de communication les plus utilisés, nous pouvons citer KQML [Finin *et al.*, 1995] et FIPA ACL [FIPA, 1999]. Tous les deux sont basés sur la théorie des actes de langages [Austin, 1962 ; Searle, 1969 ; Finin, 1993,1994a, 1994b, 1994c].

b) L'ontologie : Son rôle est de fournir un vocabulaire et une terminologie compréhensibles par tous les agents. Un ensemble de règles et de contraintes régissent cette sémantique qui permettra de définir un consensus sur le sens des termes. L'ontologie concerne les contenus des messages.

c) Les mécanismes de communication : Ce sont les outils qui permettent le stockage, la recherche et l'adressage des messages aux agents. Dans les SMA tels que les plates-formes multi-agents comme Jade ou MadKit, ces mécanismes sont prévus.

- **La coopération**

La coopération présente une forme générale d'interaction. Une telle activité [Bond, 1990] est nécessaire lorsque les activités d'un groupe particulier d'agents convergent vers un but global par l'achèvement de leurs propres buts locaux. La coopération est dans ce cas, un comportement intentionnel adopté par les agents [Ferber, 1999]. Certains auteurs [Durfee et Lesser, 1989] ont une vue différente et considèrent des critères sociaux pour qualifier la coopération. L'interprétation se faisant du point de vue d'un observateur extérieur qui interprète à posteriori les comportements des agents. Les actions des agents sont alors qualifiées de coopératives ou non à ce niveau. Les critères sociaux pris en considération sont l'interdépendance des actions ou le nombre de communications effectuées.

Ainsi, la coopération permet à un agent d'avoir recours à d'autres agents pour accomplir des tâches qu'il ne pourrait pas réaliser tout seul ainsi que pour avoir accès à des ressources partagées. De ce fait, la productivité de chaque agent et par conséquent celle du système se voit augmentée. De même, le nombre de taches est augmenté et les temps de réalisation d'une tache sont diminués. Ceci améliore de façon notable les performances de tout le système. [Ferber, 1999].

- **La coordination**

La coordination est généralement définie en termes d'actions individuelles accomplies par chacun des agents dans le but que l'ensemble du système aboutisse à un objectif cohérent et performant [Ferber, 1999]. Plusieurs cas nécessitent la coordination des actions des agents [Jennings, 1996] :

a- Lorsqu'il y'a dépendance entre les actions des agents.
b- Lorsqu'aucun agent n'a suffisamment de compétence, de ressources et d'information pour atteindre tout seul le but du système complet.
c- Lorsqu'il faut éviter les redondances dans la résolution de problèmes.

L'action du système multi-agents se trouve ainsi améliorée et les conflits diminués ce qui engendre une augmentation des performances. Les agents du système bénéficient des capacités des autres agents et la communauté d'agents peut s'auto-organiser en un ensemble cohérent d'individus et non en en une collection chaotique d'individus [Nwana *et al.,* 1996]. L'efficacité de la coordination est assurée en évitant les interactions négatives entre agents.

- **La négociation**

La négociation [Durfee et Lesser, 1989] concerne le processus de passer des accords entre les agents pour réduire les inconsistances et les incertitudes sur des points de vue communs ou des plans d'action grâce à l'échange d'informations. Cette dimension est une des bases de l'interaction surtout étant donnés que les agents sont autonomes [Jennings *et al.,* 1998].

Smith [Smith, 1988] définit la négociation comme suit :

> *"By negotiation, we mean a discussion in which the interested parties exchange information and come to an agreement. For our purposes negotiation has three important components (a) there is a two-way exchange of information, (b) each party to the negotiation evaluates the information from its own perspective, (c) final agreement is achieved by mutual selection".*

Il en ressort que la négociation repose sur trois grands axes : a) l'échange d'information (la communication), b) l'évaluation des informations envoyées d'une part et de l'autre c) un agrément (ou accord) final et une prise de décision.

2.3.5 Organisation multi-agents

L'organisation des systèmes multi-agents détermine les éléments atomiques et couplés du système ainsi que les liaisons et interactions qui existent entre eux. E. Morin [Morin, 1977] donne la définition suivante :

> « *Une organisation peut être définie comme un agencement de relations entre composants ou individus qui produit une unité, ou système, dotée de qualités inconnues au niveau des composants ou individus. L'organisation lie de façon relationnelle des éléments ou événements ou individus divers qui dès lors deviennent les composants d'un tout. Elle assure solidarité et solidité relative, donc assure au système une certaine possibilité de durée en dépit de perturbations aléatoires*».

Selon Fox [Fox, 1981], une organisation peut être définie comme une structure décrivant la façon dont les membres du système sont en relation et comment ils interagissent afin d'atteindre un but commun.

D'autres travaux [Cossentino *et al.,* 2009 ; Ferber *et al.,* 2003 ; Hübner *et al.,* 2007] placent l'organisation au centre des préoccupations de l'ingénierie des SMA. Dans ce cas, il est fait référence aux théories organisationnelles ainsi qu'aux rôles sociaux des SMA.

Dans cette perspective organisationnelle l'analyse et la conception des SMA se fait en utilisant les concepts de rôles ou comportements et attitudes abstraits au sein d'une organisation et/ou de norme sociale qui impose des contraintes et des règles au sein d'une organisation. La figure 2.18 [Zambonelli *et al.,* 2000] décrit une perspective organisationnelle conduisant à une caractérisation générale d'un SMA.

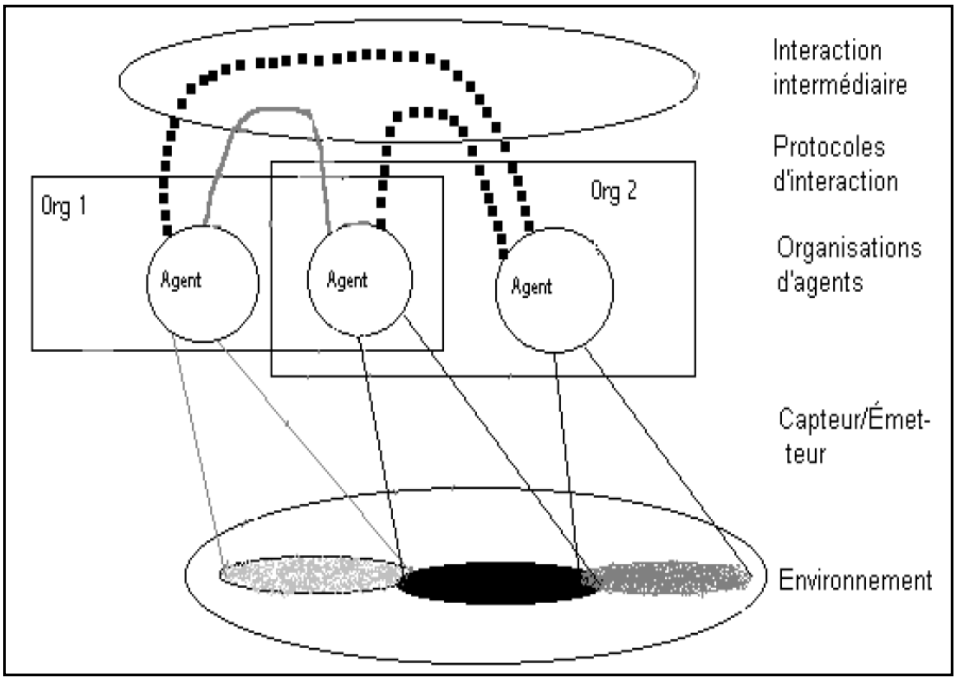

FIG. 2.18 – Caractérisation d'un SMA [Zambonelli *et al.,* 2000]

Un ensemble de sous-organisations forme le système entier. Chaque sous-organisation peut être vue comme étant composée d'un ensemble d'agents. Un agent peut jouer un ou plusieurs rôles à l'intérieur d'une sous-organisation tout en interagissant avec les autres agents de son groupe. Les sous-organisations peuvent inter-agir ensemble. Ces interactions sont assurées grâce aux capteurs (perception) et émetteurs (actions) via les portions de l'environnement auxquels peuvent accéder les agents.

2.3.6 Méthodologies de conception des systèmes multi-agents

Une phase importante dans la mise en œuvre de méthodologies de modélisation des SMA consiste en la séparation nette de l'agent des autres paradigmes logiciels (principalement l'objet). La conception orientée agent est finalement adaptée à chaque architecture d'agent. Les langages et architectures d'agents disponibles actuellement permettent l'opérationnalisation des modèles d'agents. L'analyse, quant à elle ; est indépendante des architectures d'agents, elle définit ce que fait le système et pas comment il le fait.

Plusieurs méthodologies ont été proposées pour la conception des systèmes multi-agents telles que **AAII** , **MaSE**, **DESIRE,**…Nous décrivons brièvement dans ce qui suit quelques unes.

2.3.6.1 AAII

AAII (*Australian Artificial Intelligence Institute Methodology)* [Kinny *et al,* 1996]. Cette méthodologie emploie des concepts des méthodologies orientées objet et certains concepts orientés agents. Le but étant de construire un ensemble de modèles selon les spécifications des architectures BDI (Belief, Desires, Intentions – Croyance, Desirs, Intentions) [Rao *et al.,* 1995]. La figure 2.19 montre un exemple de diagramme de classes (boîtes rectangulaires) et d'instances (boîtes arrondies) d'agents pour l'application au trafic aérien.

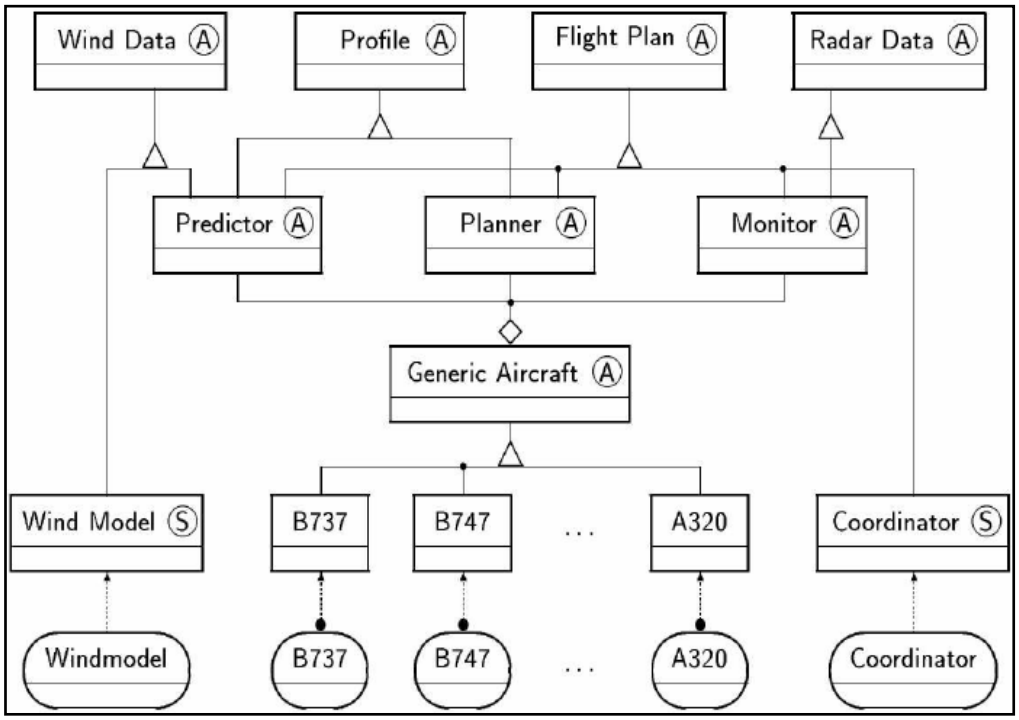

FIG. 2.19 – Un exemple de diagramme de classes et d'instances pour le problème du trafic aérien dans AAII [Extrait de Picard, 2004]

AAII exploite deux modèles :

1. Le modèle externe (point de vue externe) : il a pour rôle de définir les relations entre les classes d'agents, l'organisation, les rôles, les services et les responsabilités. Il est divisé en deux sous-modèles :

– *le modèle d'agent* qui décrit la hiérarchie entre agents (*modèle de classes d'agents*) et identifie les multiples instances possibles (*modèle d'instances d'agent*) ;
– le *modèle d'interaction* qui décrit les responsabilités, les services et les interactions entre agents.

2. Le modèle interne (point de vue interne) : il représente les comportements des agents. Il est décrit par les trois modèles suivants :

– *le modèle de croyances* (*beliefs*) qui concerne les informations à propos de l'environnement, l'état interne des agents ainsi que leurs actions possibles ;

– *le modèle de buts* (*goals*) qui décrit les *buts* que l'agent peut adopter et les évènements auxquels il peut répondre ;

– *le modèle de plans* qui décrit sous forme de *plans* les moyens pour un agent d'atteindre ses buts.

Le processus de développement d'un SMA selon AAII se déroule en quatre étapes :

1. Identification des rôles de l'application qui prend en considération les premières classes, souvent abstraites d'agents selon différentes granularités ainsi que l'organisation structurelle et fonctionnelle des agents ;

2. Identification des responsabilités et des services nécessaires à accomplir pour chaque rôle d'agent ;

3. Identification des interactions requises par la notion de responsabilités pour chaque service ainsi que la définition des formes, actes et contenus des langages de communication utilisés par les agents ;

4. Raffinement de la hiérarchie d'agents par le biais de l'introduction de nouvelles classes ou sous-classes d'agents ou en composant des classes d'agents dans le but d'établir les instances d'agents nécessaires.

Cette méthodologie reste très abordable si l'utilisateur est familiarisé avec la conception orientée objet. Cette méthode (AAII) est assez ancienne. Actuellement, peu de ressources sont disponibles pour des concepteurs désirant l'utiliser.

2.3.6.2 MaSE

MaSE (*Multiagent Software Engineering*) c'est une méthodologie de développement de systèmes multi-agents qui utilise un ensemble important de modèles et outils graphiques pour la description des types d'agents ainsi que des interactions inter-agents [Deloach, 1999 ; Deloach et Wood, 2000 ; Deloach *et al.* 2001, Deloach, 2002]. Plusieurs des diagrammes utilisés s'inspirent assez des approches orientées objet. Cependant, ils différent de cette dernière approche par le fait qu'ils apportent en plus un certain nombre de caractéristiques ainsi que quelques modifications de la sémantique du paradigme objet qui sont nécessaires à la conception des agents et de leurs comportements coopératifs.

MaSE se décompose en sept étapes réparties en deux phases (Figure 2.20). Chaque étape a pour résultat un ou plusieurs diagrammes. La phase d'analyse comporte trois étapes :

1. Identification des rôles à partir des spécifications des besoins de l'utilisateur et leur structuration dans un diagramme de hiérarchie de rôles ;

2. Identification des cas d'utilisations et création des diagrammes de séquences associés nécessaires à l'identification d'un ensemble initial de rôles et de voies de communication ;

3. Transformation des buts en ensembles de rôles :
(a) Création d'un modèle de rôle pour la saisie des rôles et des tâches associées ;
(b) Définition d'un modèle de tâches concurrentes pour chaque tâche afin de définir les comportements des rôles.

L'étape de conception comporte quatre phases :

1. Assignation des rôles à des classes d'agents spécifiques et identifications des conversations en examinant les modèles de tâches concurrentes tout en se basant sur les rôles joués par chaque classe ;

2. Construction des conversations en extrayant les messages ainsi que les états définis pour chaque voie de communication dans les modèles de tâches concurrentes et en ajoutant des messages et des états pour plus de robustesse ;

3. Définition des classes internes d'agents grâce à une architecture de chaque classe d'agent en utilisant des composants et des connecteurs. Il y'a lieu, alors, de s'assurer que chaque action apparaissant dans une conversation est bien implémentée comme une méthode au sein de l'architecture d'agent ;

4. Définition de la structure finale du système en utilisant des diagrammes de déploiement.

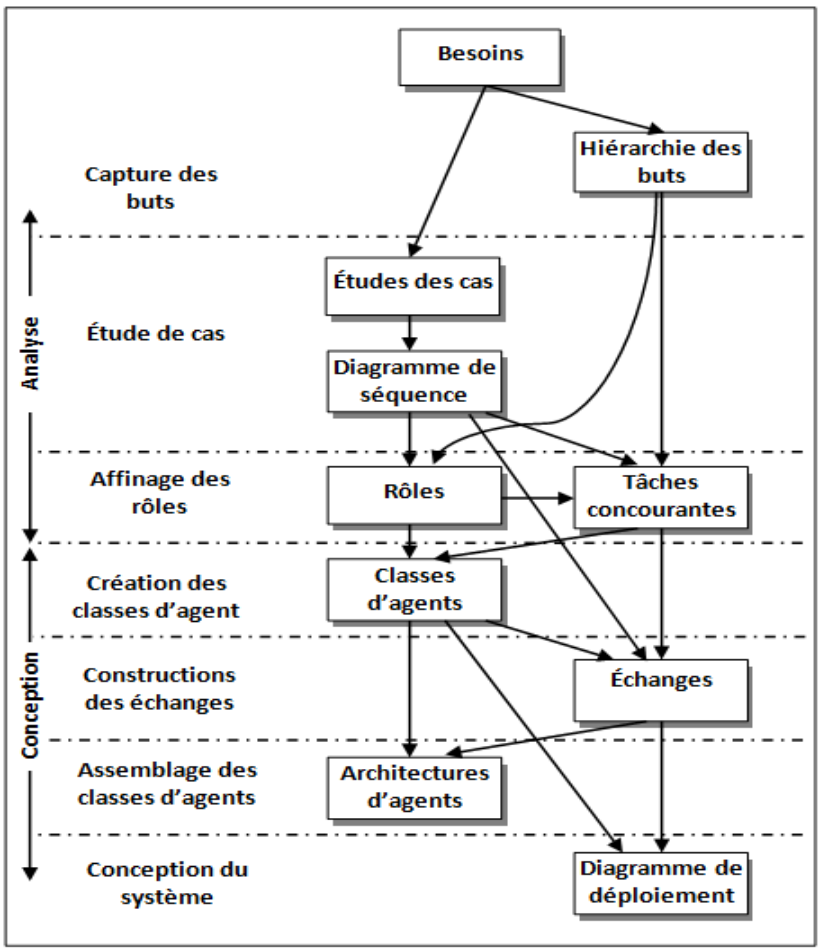

FIG. 2.20 – Les différentes étapes et les modèles de MaSE.

2.3.6.3 DESIRE

DESIRE (*Design and Spécification of Interacting REasoning framework)* [Brazier *et al.*, 1998] est l'une des premières méthodes le domaine des méthodes orientées agents. Cette méthode est directement issue de l'ingénierie des connaissances. La figure 2.21 montre comment un modèle générique d'agent peut être décrit par composition de tâches en employant la notation graphique compositionnelle de DESIRE qui permet de définir les tâches et leur relation de séquentialité et de délégation par composition.

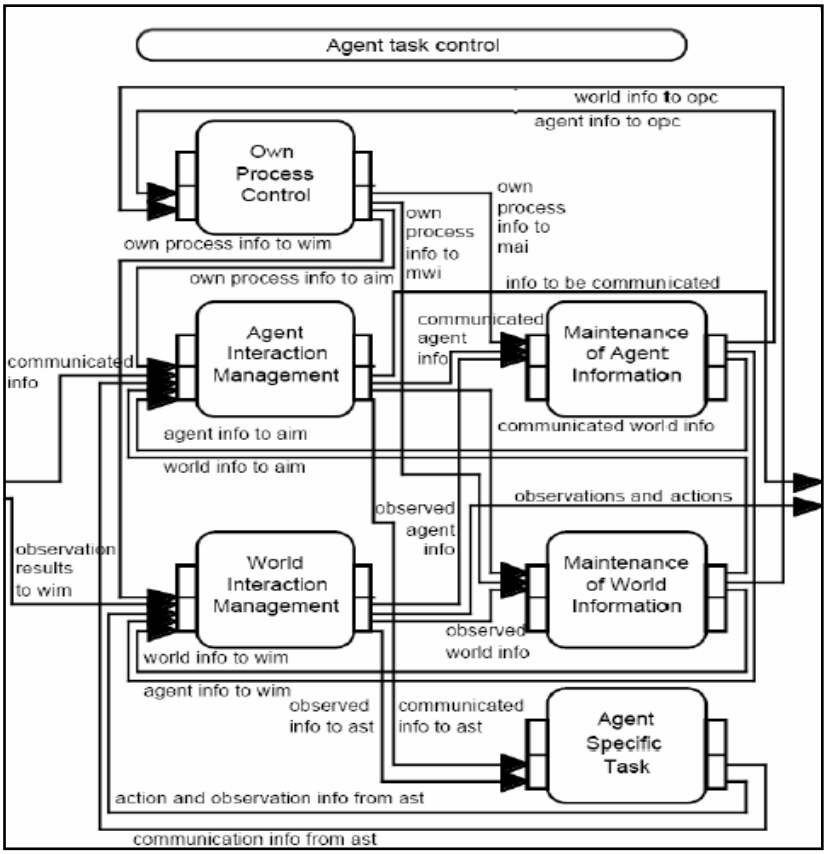

FIG. 2.21 – Le modèle d'agent générique de DESIRE

DESIRE est basée principalement sur trois concepts clés :

- ***La représentation des connaissances*** se fait à la fois de façon graphique et textuelle.
- ***Le modèle générique d'agent*** utilisé dans DESIRE est de type BDI. Il est jugé comme faible, dans le sens défini dans [Wooldridge et Jennings, 1995]. Un agent DESIRE possède au moins les quatre types de comportements suivants : autonomie (de contrôle), comportement réactif, comportement pro-actif, comportement social (communiquer, coopérer,...).
- ***Le modèle de composition de tâches***, qui permet de décrire chaque tâche des agents. Ce modèle permet de décomposer des tâches en sous-tâches et de spécifier des relations de délégation ou de séquence. Ce modèle est fonctionnel et chaque tâche est vue comme un composant ayant des entrées et des sorties d'informations [Brazier *et al.*, 1998]

Le processus de développement de DESIRE se décompose comme suit :

1. Description du problème qui inclut les besoins imposés à la conception ;

2. Principes de conception pour spécifier les choix faits lors de la conception ;

3. Conception conceptuelle qui définit un modèle conceptuel pour chaque agent, le monde extérieur et les interactions entre agents ;

4. Conception détaillée : spécification des aspects de connaissance et de comportement ;

5. Conception opérationnelle qui va spécifier les paramètres nécessaires à l'implémentation.

2.3.6.4 Gaia

Gaia est considérée comme étant une extension des approches d'ingénieries logicielles classiques [Wooldridge *et al.,* 2000]. Elle s'inspire de la méthodologie orientée objet FUSION [Coleman *et al.,* 1994]. GAIA est une méthode plus complète et bénéficie, de plus, d'une large reconnaissance dans le domaine multi-agents qu'AAII ou DESIRE. C'est une méthode de la seconde génération. Les phases de GAIA sont les suivantes (voir la figure 2.22) :

1. la phase d'analyse : pour l'établissement des modèles préliminaires visant à comprendre les buts du système. On y définit :

– la structure générale du système, en somme ; sa composition : son organisation et ses couplages.

– la modélisation de l'environnement du système, qui représente l'ensemble des ressources et des objets mises à disposition des agents qu'ils peuvent utiliser, manipuler, modifier, créer…

– le modèle préliminaire des rôles : ensemble des fonctionnalités ou compétences de base du système et les agents nécessaires afin que le système puisse atteindre son but,

– le modèle préliminaire d'interactions modélisé sous forme de dépendances et de relations entre les rôles,

– les règles organisationnelles : captures des relations générales entre les rôles et entre les protocoles au niveau de l'organisation ;

2. la phase de conception : C'est au cours de cette phase que sont opérés les choix à faire concernant les caractéristiques du système en conséquence de la première phase :

– Choix de la conception de l'architecture : Décider de la structure organisationnelle du système (repérage et application des patrons organisationnels). On y utilise des notations formelles et graphiques pour représenter l'organisation du système et on complète les modèles de rôles et les modèles d'interactions,

– la phase de conception fine : C'est la phase qui correspond à l'identification précise des rôles des agents. De même, les modèles des services que chaque agent offre aux autres agents sont explicités lors de cette phase.

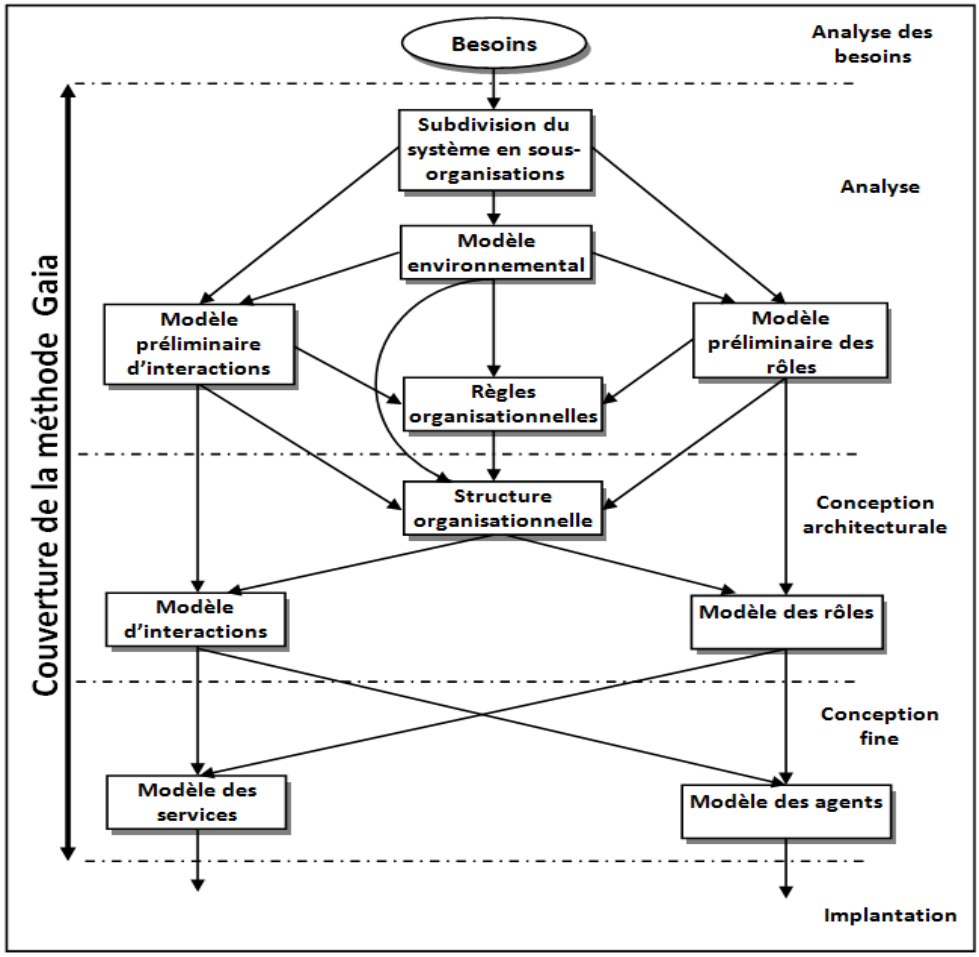

FIG. 2.22 – Le processus de développement de Gaia et les modèles manipulés.

2.3.6.5 PASSI

PASSI (*Process for Agent Societies Specification and Implementation*) [Cossentino *et al.*, 2005] : Cette méthodologie s'inspire fortement des modèles de conception et des concepts de l'ingénierie orientée objet et de l'intelligence artificielle (IA) en utilisant la méthode UML[Cossentino, 2001; Cossentino et Potts, 2002].

Le processus de PASSI est un processus pas à pas réitérable. Il est composé des cinq phases suivantes (voir figure 2.23),

1. Les besoins : Ils permettent de décrire le contexte du système.

2. La définition de la société d'agents : qui permet l'identification des interactions sociales et des dépendances entre les agents.

3. L'implémentation des agents : C'est au cours de cette phase que sont définis la structure et le comportement du SMA et des agents.

4. Le codage : Afin d'identifier les modules réutilisables pour les développements préexistants.

5. Le déploiement : Pour la spécification de la configuration du système.

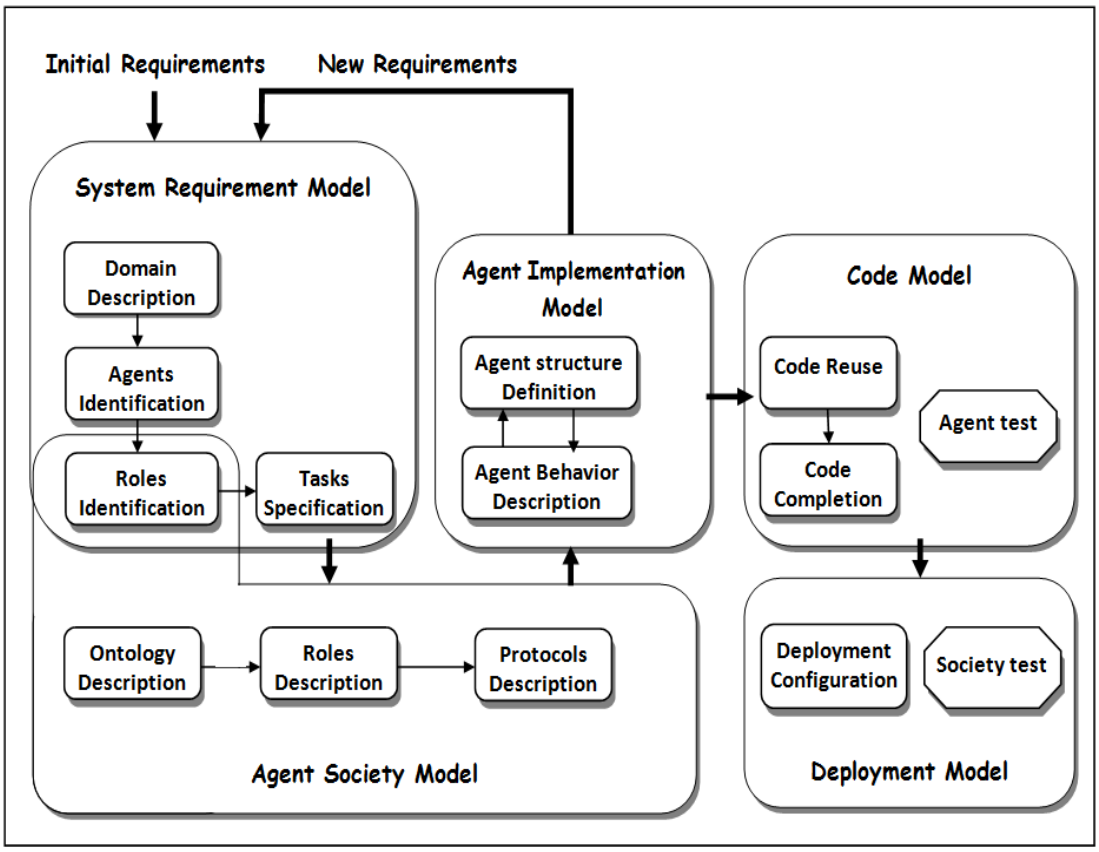

FIG. 2.23 – Les cinq modèles/phases de la méthode PASSI.

2.3.6.6 MESSAGE

MESSAGE (Methodology *for Engineering Systems of Software AGEnts*) [Caire *et al.*, 2001; Coulier *et al.*, 2004] est le fruit d'un projet réalisé par les compagnies de télécommunication européennes, porté par EURESCOM [EURESCOM, 2000]. MESSAGE étend la notation UML [Jacobson *et al.*, 1999].

Les concepts orientés agent de MESSAGE (voir figure 2.24) ajoutés à UML sont :

– agent : Un agent est considéré comme étant une entité atomique autonome capable de réaliser une fonction ;

– organisation : C'est un groupe d'agents opérant ensemble pour réaliser un but commun ;

– rôle : la distinction entre rôle et agent est similaire à celle entre une interface et une classe d'objet, le rôle décrivant les caractéristiques externes de l'agent dans un contexte donné ;

– ressource : C'est toute entité non autonome ;

– tâche : activité définie par des ensembles de pré-conditions et de post-conditions. Si les pré-conditions sont satisfaites et que la tâche est activée, on peut s'attendre à ce que les post-conditions deviennent vraies. Cette notion permet de créer des chaînes de causalités.

– interaction et protocole : qui spécifie la façon pour les agents d'interagir lorsqu'ils tentent de coopérer pour atteindre un but commun ;

– but : associe un agent à une situation ;

– entité informationnelle : c'est une donnée, une information (entité passive) ;

– message : un message est une information passée d'un agent à un autre agent et non pas au sens UML (appel de méthode d'un objet)

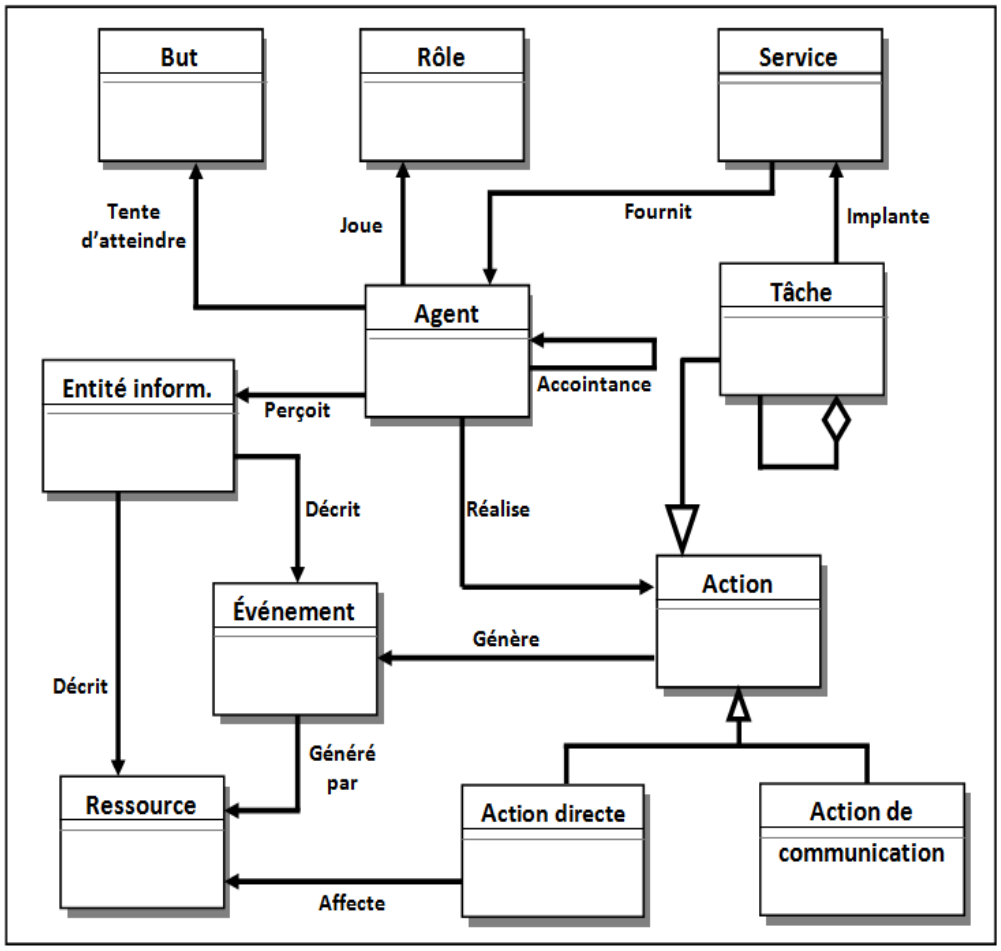

FIG. 2.24 – La méthode MESSAGE

2.3.6.7 Prometheus

Padgham et Winikoff [Padgham et Winikoff, 2002, 2003, RMIa, RMIb] ont développé la méthode Prometheus en tant que méthode complète (méthode, notations et outils) pour des agents BDI. Cette méthode a considérablement évolué aujourd'hui et elle est indépendante de toute architecture. C'est une méthode de seconde génération, qui profite des avancées des méthodes précédentes, et s'inspire notamment de Gaia ou MaSE.

Les modèles produits et le processus de Prometheus sont décrits dans la figure 2.25. Trois phases majeures composent ce processus :

1. la spécification du système :

– Identification des buts et sous-buts,

– Réalisation de cas d'études,

– Spécification de l'interface système de l'agent avec l'environnement par rapport aux actions et percepts,

– Identification des fonctionnalités,

– Identification des types de données lues et écrites par ces fonctions ;

2. Conception de l'architecture :

– Regroupement des fonctionnalités pour déterminer les types d'agents en se basant sur le couplage des données,

– Conception des protocoles d'interactions selon les cas d'études par le biais de diagrammes d'interactions ;

3. Conception fine :

– Conception des diagrammes de processus,

– Définition de la structure interne des agents,

– Définition des détails des événements, des plans et des types de données.

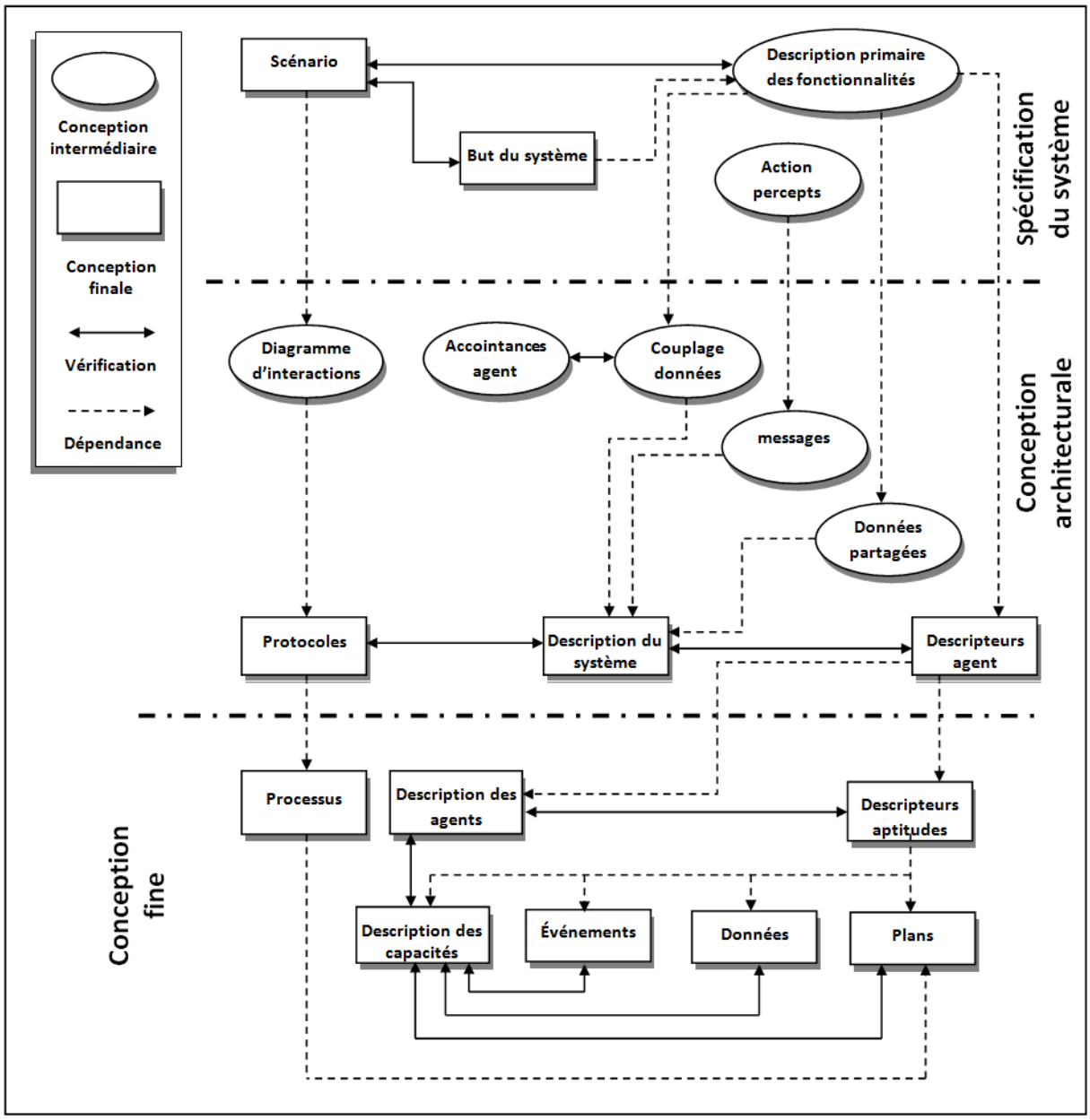

FIG. 2.25 – Méthode Prometheus

2.3.6.8 La méthode Voyelles

L'approche Voyelles ou méthode AIEO (Agent, Evenement, Interaction, Organisation) [Demazeau, 2001a] est une méthode d'agentification développée au sein de l'équipe Magma. C'est une méthode de haut niveau (peu de directives techniques sont données) reposant sur des principes purement multi-agents [Demazeau, 1995, 2001a, b, 2003]. La conception d'un système multi-agents passe par sa décomposition selon les quatre briques : AIEO, [Boissier et Demazeau, 1996 ; Ricordel et Demazeau, 2000, 2002 ; da Silva et Demazeau, 2002].

Voyelles suit un processus de développement logiciel classique, comme le montre la figure 2.26 :

– L'analyse : Cette phase considère le domaine d'application ainsi que le type du problème posé. Le problème est décomposé selon les différentes composantes du système. À un plus haut niveau d'abstraction, les entités identifiées devront répondre aux principes de récurions ou d'émergence.

À la fin de l'analyse, le problème est donc « voyellisé » ;

– La conception : correspond à l'identification des modèles à utiliser pour chacune des voyelles [Occello *et al.*, 2001].

– L'implantation : c'est au cours de cette phase que les modèles sont instanciés en utilisant des plates-formes et des langages adéquats. Le système implémenté, est alors exécuté, évalué (test et validation) et corrigé si ses résultats sont en inadéquation avec les spécifications exprimés par le type de problème et le domaine d'application.

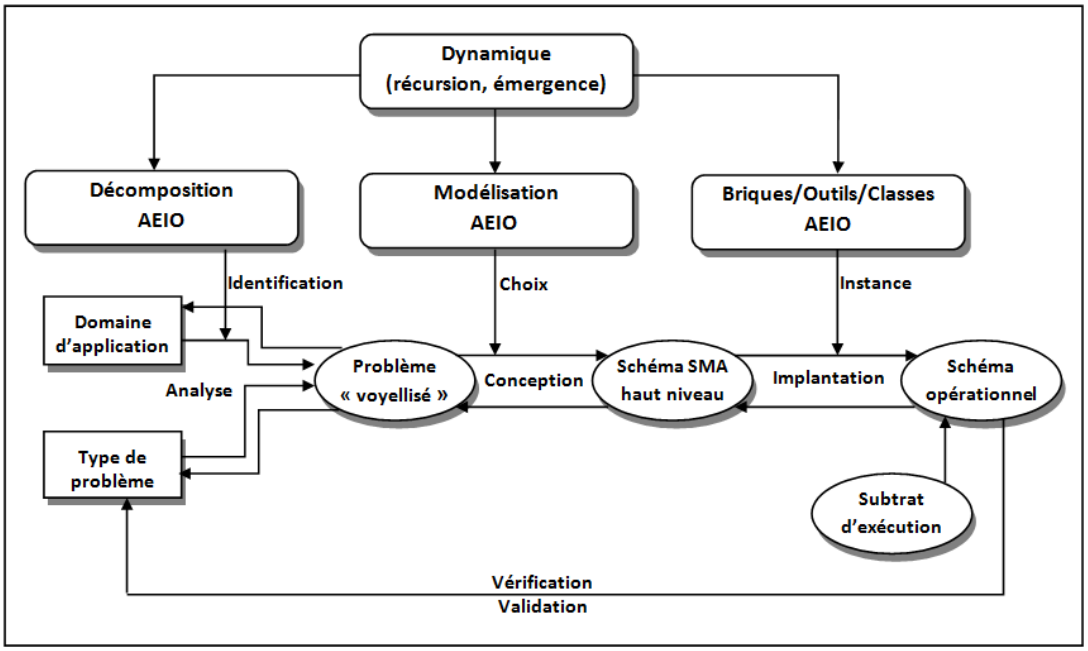

FIG. 2.26 – La méthode Voyelles.

2.3.6.9 ADELFE

ADELFE (*Atelier pour le DÉveloppement de Logiciels à Fonctionnalité Émergente)* [Bernon *et al.*, 2005] est une méthodologie applicable aux systèmes multi-agents auto organisateurs par coopération [Picard et Gleizes, 2004]. La méthode ADELFE a été développée dans le but

de fournir un outil applicatif de la théorie des AMAS (Adaptive Multi-Agent Systems) [Jean-Pierre, 2003].

Le processus de développement d'ADELFE est basé sur le *RUP (Rational Unified Process)* [Jacobson *et al.,* 1999]. Le processus de développement ADELFE comprend quatre phases : la collecte des besoins préliminaires, la collecte des besoins finaux, l'analyse et la conception. La méthode ADELFE propose une plateforme comprenant un outil de modélisation graphique ; une bibliothèque de composants permettant des simulations et un prototypage rapide et une technique d'auto-assemblage des composants logiciels que sont les agents adaptatifs.

Le processus ADELFE s'articule autour des phases suivantes (voir figure 2.27) :

1. La phase d'analyse.

– Vérification de la pertinence d'une approche multi-agents qui concerne l'étude de critères tels que :

- distribution physique ou fonctionnelle,
- environnement évolutif,
- besoin d'interactions–coopération entre entités.
- L'analyse (similaire à celle de UML) ;
- complexité nécessitant l'approche distribuée,
- absence d'autres méthodes,
- frontières mal définies,

2. La phase de conception.

- Pour la conception d'un agent, ADELFE propose une description d'agent générique composée de sept modules :

* Communication avec l'environnement,
* Croyances sur lui-même,
* Croyances sur les autres agents,
* Croyances sur son environnement,
* Communication avec les autres agents,
* Compétences,
* Attitude sociale et coopérative.

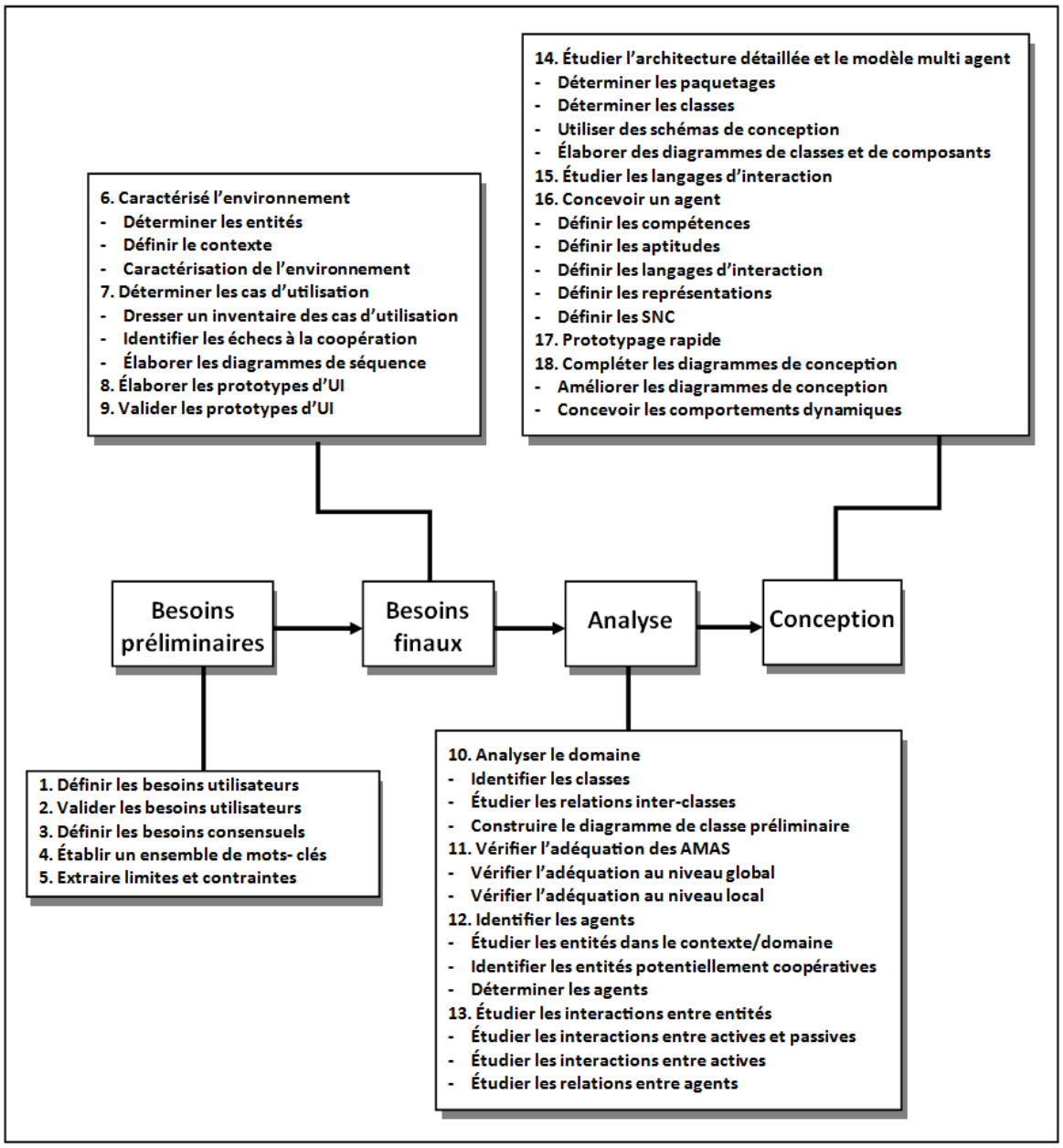

FIG. 2.27 – Le processus ADELFE.

2.3.6.10 AALAADIN

AALAADIN [Ferber et Gutknecht, 1998] Figure 2.28 est l'une des premières méthodologies à établir les principes de développement des SMA fondée sur le concept AGR (Agent, Groupes et Roles) [Ferber *et al.*, 2003]. Dans AALAADIN, un agent est considéré comme une entité communicante qui joue un ou plusieurs rôles dans un ou plusieurs groupes. Ces derniers peuvent se chevaucher et se former dynamiquement. Chaque agent peut appartenir à plusieurs groupes. Un rôle est une représentation abstraite de la fonction, de service ou l'identificateur d'un agent.

Dans AALAADIN, un agent peut jouer un ou plusieurs rôles. Chaque rôle est local à un groupe. Les agents ne peuvent établir des communications entre-deux qu'aux travers des rôles qu'ils jouent. De ce fait, le contrôle sur les communications est effectué par le groupe.

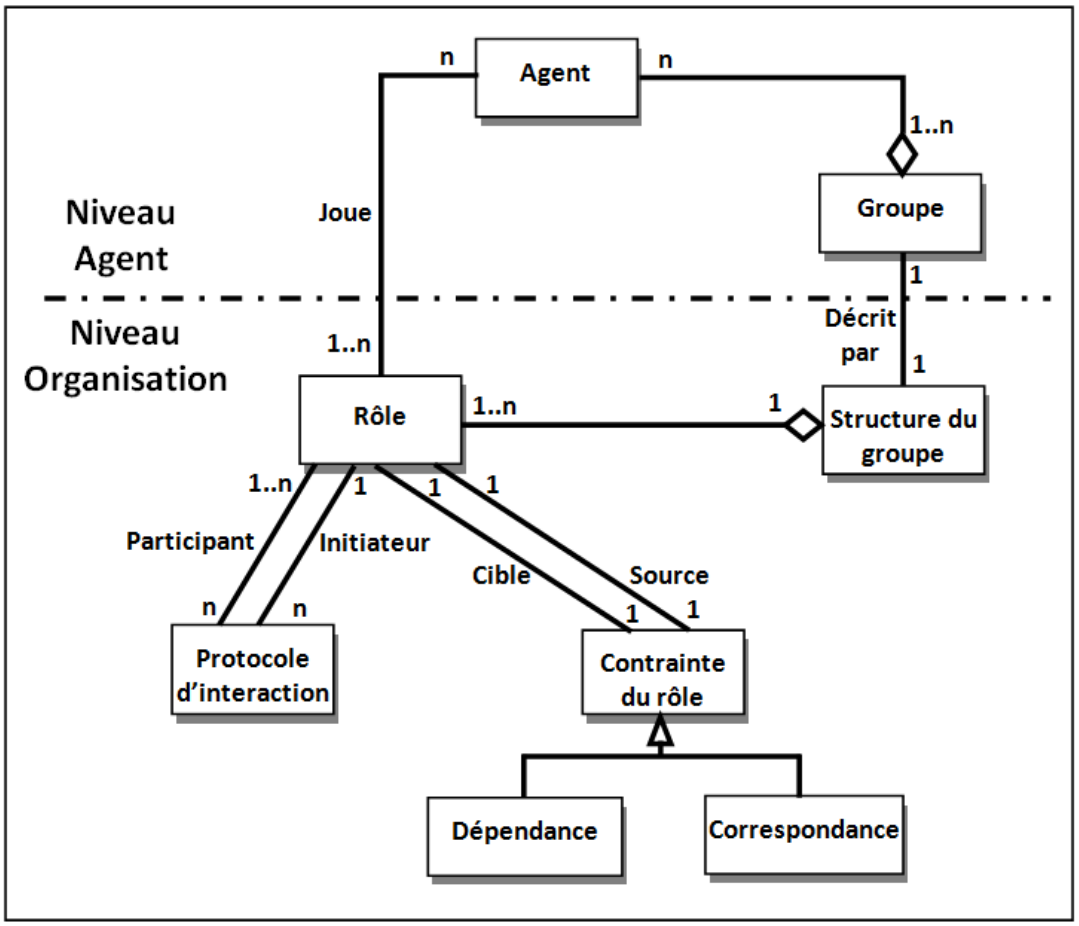

FIG. 2.28 – Le modèle organisationnel AALAADIN

AALAADIN est une méthodologie focalisée sur l'organisation et non sur l'agent. Ceci à des fins de sécurité, de normes et de responsabilités [Ferber *et al.*, 2003]. Au sens d'AALAADIN, Une organisation est un framework assurant les activités et les interactions en prenant en considération les groupes, les rôles et leurs relations. Ainsi, une organisation est déterminée comme un étant un ensemble de relations structurelles entre les rôles définis au sein des groupes. La notion d'interaction n'est pas explicitement représentée ni manipulable dans le système (réifiée) et la méthode ne fournit pas de principes permettant de faciliter la décomposition d'un système complexe. Une organisation est découpée selon deux niveaux :

– *La structure organisationnelle* qui représente les relations qui font d'une agrégation d'individus un tout identifiable (un invariant organisationnel) ;

– *l'organisation concrète* qui est une instance de structure organisationnelle avec des agents, des rôles et des groupes particuliers.

Trois types de diagrammes sont utilisés dans AALAADIN :

Les diagrammes de structures organisationnelles : Ce diagramme représente les structures des groupes, les rôles, les contraintes et les interactions entre rôles ;

Les diagrammes d'organisations concrètes permettent de représenter l'appartenance d'agents (quilles) qui jouent des rôles dans des groupes (ovales).(voir figure 2.29) ;

Les diagrammes organisationnels d'interaction spécifient les interactions entre groupes et/ou rôles. Ces diagrammes sont analogues aux diagrammes de séquences d'UML. A la différence que les lignes de vies représentent des rôles et non à des instances de classes (comme dans A-UML). Ces diagrammes permettent notamment de représenter les créations ou destructions de groupes.

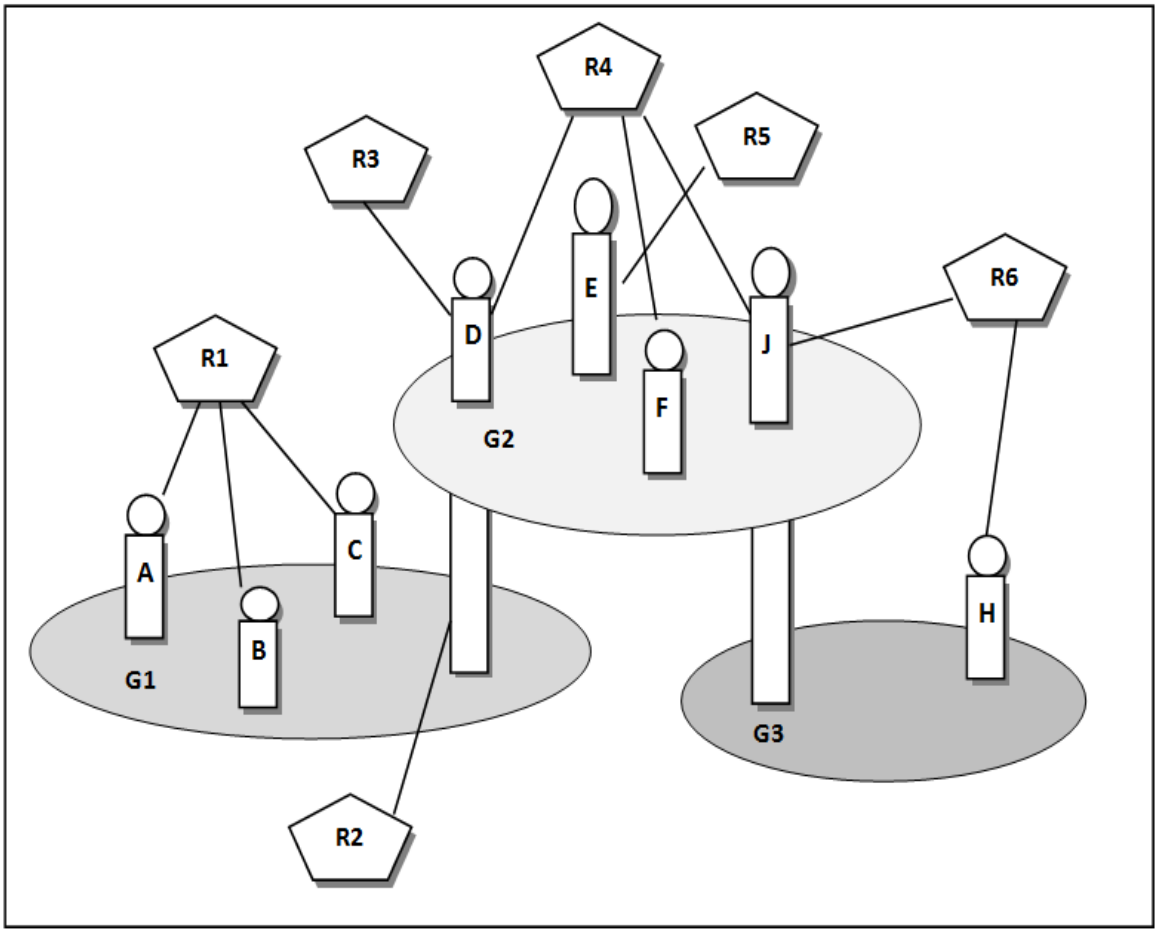

FIG. 2.29 – Les diagrammes d'organisation concrète dans Aalaadin.

Le processus de développement d'AALAADIN suit trois phases [Gutknecht et Ferber, 1999]:

1. L'analyse : Pour l'identification des fonctions du système et des dépendances au sein des communautés identifiées. Il convient de définir quels sont les mécanismes de coordination et d'interaction entre les entités d'analyse.

2. La conception qui est la phase d'identification des groupes et des rôles dans des diagrammes de structures organisationnelles. Les protocoles d'interactions alors identifiés entre rôles sont décrits dans des diagrammes organisationnels de séquences. L'identification des objets et des actions sur ces objets permet de commencer à décrire les attributs des agents.

3. La réalisation commence par le choix de l'architecture d'agent. Ensuite, la gestion des entités du domaine permet d'implanter le système à partir d'organisations concrètes.

La plate-forme MadKit est un SMA construit autour des concepts AALAADIN est qui est utilisée comme plate-forme de prototypage et de simulation [Campagne, 2005].

2.3.6.11 Autres méthodologies

Un nombre important de méthodologies furent développées, telles que : MASSIVE [Lind, 2001] , Tropos [Bresciani *et al.,* 2004], Ingenias [Pavon et Gomez-Sanz, 2003], OAA (open agent architecture) [Martin *et al.,* 1999] Kaos [Bradshaw *et al.,* 1996], JAFMAS [Chauhan, 1997] , dMARS [d'Iverno *et al.,* 1997], Cassiopée [Collinot et Drogoul, 1996, 1998; Drogoul, 2000], RETSINA [Sycara *et al.,* 2001], Role Modeling [Collis et Ndumu, 1999a,b], SCALA [Degirmenciyan & Marc, 2002]…

2.3.6.12 Quelques éléments de comparaison entre différentes méthodologies de conception des SMA

Il existe de nombreuses méthodes de développement des SMA. Ces méthodes ne cessent d'évoluer de façon très rapide. Il est ainsi, intéressant de faire une comparaison de l'existant afin de pouvoir bien cerner, d'une part les apports de ces méthodologies et d'autre part leurs lacunes éventuelles. Le tableau 2.30 tiré de [Bernon *et al.,* 2009] résume des éléments de comparaison de quelques méthodologies de conception des SMA.

	ADELFE	Gaia	MaSE	INGENIAS	PASSI	Prometheus	Tropos	Voyelles
Adaptation	++	±	+	±	±	-	-	+
Autonomie	+	+	+	+	+	+	+	±
Buts	+	±	+ +	+ +	±	+ +	+ +	±
Croyances	+	±	+	-	+	+ +	+ +	+
Désirs	-	±	+ +	+	+ +	+ +	+	+
Environnement	+	±	-	-	±	+	±	+
Groupe	±	±	-	±	+	+	±	+
Intention	-	±	±	±	±	+ +	+ +	+
Interaction/Message	++	+	+ +	+ +	+ +	+ +	+	+ +
Norme	±	+	+ +	±	-	±	+	+
Organisation	+	+ +	±	+ +	+	+	+	+ +
Ouverture	++	- -	±	±	±	-	- -	+
Proactivité	+	+	+	+	+	+	+	±
Réactivité	++	+	-	+	+	±	+	±
Rôle	+	+ +	+ +	+ +	+ +	-	+ +	+
Tâche	±	+	+ +	+ +	+ +	+	±	+
Accessibilité	+	±	±	+	+	+	+	?
Analysabilité	+	- -	+	+	+	+ +	+ +	?
Complexité	+	- -	±	+	+	+	-	±
Consistance	+	+	±	±	+	+ +	+ +	?
Exécution	++	- -	+ +	+	+ +	+	+ +	+
Expressivité	+	±	±	±	±	+	±	-
Modularité	++	+ +	±	+ +	+ +	+	+	?
Portabilité	±	±	±	±	+	±	-	+
Précision	-	+ +	+ +	±	±	+	+ +	- -
Raffinage	+	+	+ +	+	+	+	+ +	?
Traçabilité	+	+	-	+	+	+	±	?
Contexte	Tous	Tous	Tous	Tous	Tous	Tous	Tous	Tous
Besoins	+	-	- -	+	+	-	+ +	-
Analyse	++	+ +	+ +	+ +	+ +	+ +	+ +	+ +
Conception	++	+ +	+ +	+ +	+ +	+ +	+	+ +
Implémentation	+	- -	+	+	+	+	+	+
Test	+	-	±	+	±	-	±	+
Déploiement	±	- -	- -	±	+ +	- -	- -	+
Maintenance	±	- -	- -	±	±	- -	- -	- -
Délivrables	++	+ +	+ +	+ +	+ +	+	-	- -
Gestion de la qualité	+	- -	- -	+	- -	- -	- -	- -
Gestion de projet	++	- -	- -	+	- -	- -	- -	- -
Ressources	++	±	+	+ +	+ +	+	±	-
Expertise requise	Plutôt Oui	Oui	Non	Non	Non	Oui	Oui	Non
Langage spécifique	Non	Non	Non	Non	Non	Non (Java)	Non	Non
Domaine d'application	Non	Gros Grains	Non	Non	FIPA	Utilisateur	Utilisateur	Non
Scalabilité	Oui	Oui	Non	Oui	Oui	Oui	Non	Oui

Légende : (++) pour les propriétés pleinement et explicitement prises en charge ; (+) pour les propriétés prises en charge de manière indirecte ; (±) pour des propriétés potentiellement prises en charge ; (−) pour des propriétés non prises en charge ; (−−) pour des propriétés explicitement non prises en charge.

FIG. 2.30 – Comparatif des différentes méthodes de conception des SMA [Bernon *et al.*, 2009].

Il apparait sur ce tableau quelques caractéristiques communes :

- ✓ On observe que c'est pratiquement le même découpage en phases de la tâche de modélisation : la spécification des besoins, l'analyse et la conception.
- ✓ Les modèles d'agent et dans les services définissent l'architecture interne de l'agent ainsi que ses fonctionnalités.
- ✓ Un modèle d'interface d'environnement définit l'environnement dans lequel évoluent les agents.
- ✓ Les interactions qu'entretiennent les agents entre eux sont représentées à travers des modèles d'interaction et de rôles.

On remarque, cependant que peu de méthodologies proposent une dynamique aux systèmes applicatifs. Ceci diminue grandement l'action du modélisateur quand il s'agit de SMA. La notion de dynamique étant au centre même de l'approche SMA.

Le modèle AGR proposé dans AALAADIN assure la généricité. Le point fort de ce modèle réside dans la plateforme MadKit qui reprend les principes même d'AALAADIN et qui a été écrite et développée par la même équipe. Nous présentons cette plateforme dans la section suivante.

2.3.7 Outils et environnements pour les systèmes Multi-Agents

Il existe aujourd'hui plusieurs plates-formes multi-agents dédiées conçues et réalisées selon plusieurs approches d'agents. Ces plates-formes ont l'avantage de fournir des couches d'abstraction permettant d'implémenter facilement les concepts des systèmes multi-agents. Une plate-forme SMA permet aussi le déploiement, l'exécution et l'évolution des agents de ces systèmes. Elle offre un environnement permettant de gérer le cycle de vie des agents dans lesquels ces derniers ont accès à certains services.

2.3.7.1 AgentBuilder

AgentBuilder [AgentBuilder R.M., 2000; AgentBuilder U.G., 2000]. Cet environnement de développement a été mis au point par Reticular Systems Inc en JAVA. C'est un outil

commercial permettant de construire des agents intelligents selon les concepts du modèle BDI. Les agents sont décrits à l'aide du langage RADL (*Reticular Agent Definition Language*). Ce langage fournit des outils qui permettent de définir les règles du comportement des agents. Certaines conditions associées à des actions déclenchent les règles. Les conditions portent sur les messages reçus par l'agent tandis que les actions correspondent à l'invocation de méthodes Java. De même, Il est aussi possible de décrire des protocoles définissant les messages acceptés et émis par l'agent. La manipulation d'AgentBuilder nécessite de bonnes connaissances dans le domaine des SMA. Cet outil fournit un ensemble assez complet du point de vue des composants mais reste limité aux niveaux de l'extensibilité, du déploiement et de la réutilisabilité. Durant la phase d'analyse, AgentBuilder propose l'utilisation de la méthode OMT pour la spécification des objets du domaine et des opérations qu'ils peuvent effectuer. Cette spécification conduit à construire une ontologie du domaine. Un ensemble d'outils graphiques est disponible pour la réalisation de ces tâches. C'est au niveau de la phase de conception, que les fonctionnalités, les rôles et les caractéristiques ainsi que les protocoles d'interaction sont assignées aux agents après leurs identifications. Le comportement de l'agent est défini ainsi que ses croyances, intentions et capacités durant la phase de développement. Durant le déploiement, le code de l'agent est interprété par le *Run-Time Agent Engine*.

2.3.7.2 JADE

JADE (Java Agent DEvelopment framework) [Bellefemine *et al.,* 2004] est un environnement de développement développé selon les normes FIPA [Bellifemine *et al.,* 2002a, 2002b ; Pitt et Bellifemine, 1997 ; Poggi *et al.,* 2001 ; Bellifemine *et al.,* 1999 ; Rimassa *et al.,* 2000]. Cet plateforme facilite le développement d'agent. Elle s'articule autour de trois modules conformes aux normes FIPA (Figure 2.31) :

- ACC (Agent Communication Channel) gère la communication entre les agents.
- AMS (Agent Management System) supervise l'enregistrement des agents, leur authentification, leur utilisation ainsi que l'accès aux ressources du système.
- DF (Directory Facilitator) fournit un service de pages jaunes à la plate-forme

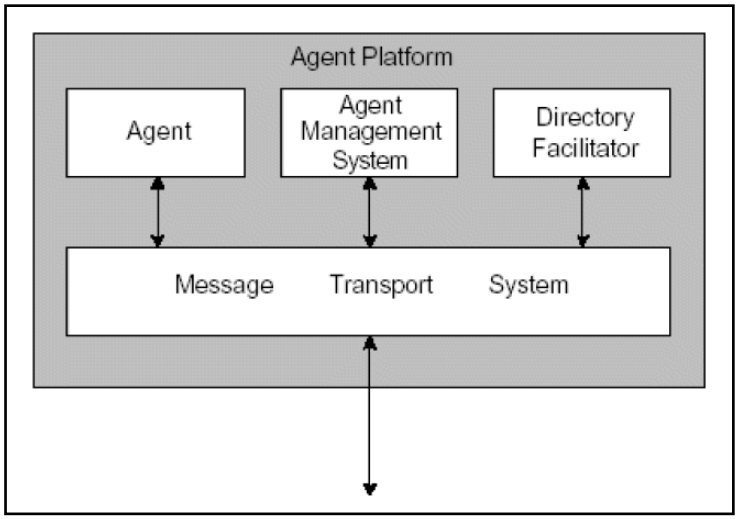

FIG. 2.31 – Architecture d'un agent FIPA [Bellifemine *et al.*, 2002b]

Les agents dans JADE possèdent les six propriétés suivantes :

– Autonomie : les agents contrôlent leurs actions, leurs propres prises de décisions afin de réaliser leurs buts mais aussi leurs cycles de vie.

– Réactivité : les agents sont munis de perceptions des évènements de leur environnement et réagissent en fonction de ces évènements.

– Aspects sociaux : les agents sont dotés d'aspects sociaux qui leur permettent de communiquer et d'interagir entre eux. La communication se fait grâce au passage de messages asynchrones. La communication est considérée comme un type d'actions et peut, de ce fait, intégrer un plan d'actions. Les messages sont formulés selon une sémantique et une structure conformes au standard FIPA.

– Dynamicité : les agents ont la possibilité de découvrir dynamiquement d'autres agents et de communiquer avec eux.

– Offre de service : Un ensemble de services est offert par l'agent. Il peut enregistrer ses services et les modifier. De même, l'agent a aussi la possibilité de chercher d'autres agents qui offrent les services dont il a besoin.

– Mobilité : Dans JADE, les agents sont implémentés dans des conteneurs et peuvent se déplacer.

JADE est implémentée sous le langage Java et profite ainsi des fonctionnalités de ce langage. La plate-forme JADE peut être répartie sur un ensemble de machines et configurée à distance. La possibilité de migration des agents au sein de la même plateforme peut altérer la configuration du système dynamiquement (voir figure 2.32).

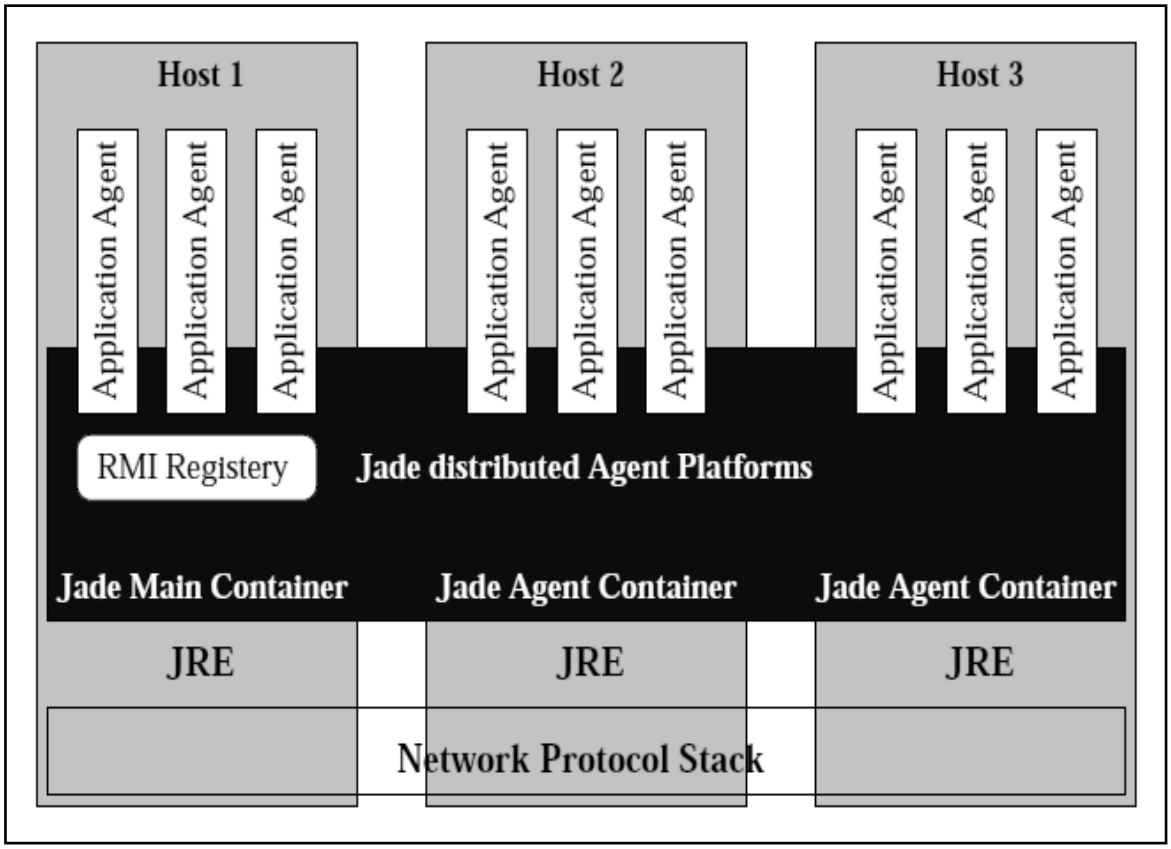

FIG. 2.32 – Architecture d'une application Jade [Bellifemine *et al.,* 2002b]

2.3.7.3 ZEUS

ZEUS [Ndumu *et al.*, 1998 ; Azarmi et Thompson, 2000] est un environnement SMA complet qui utilise la méthodologie *Role Modeling* pour le développement de systèmes collaboratifs [Collis et Ndumu, 1999a, b, c, d]. ZEUS couvre toutes les étapes du processus de développement. Ces étapes se font à l'intérieur de plusieurs éditeurs [Collis et Lee, 1998 ; Collis *et al.,* 1998, 1999]. Dans ZEUS, un agent (figure 2.33) est composé de trois couches :

- la couche de définition, qui contient les capacités de raisonnement et les algorithmes d'apprentissage,
- la couche organisationnelle, qui contient la base de connaissances des accointances de l'agent,
- et la couche de coordination, pour les interactions avec les autres agents.

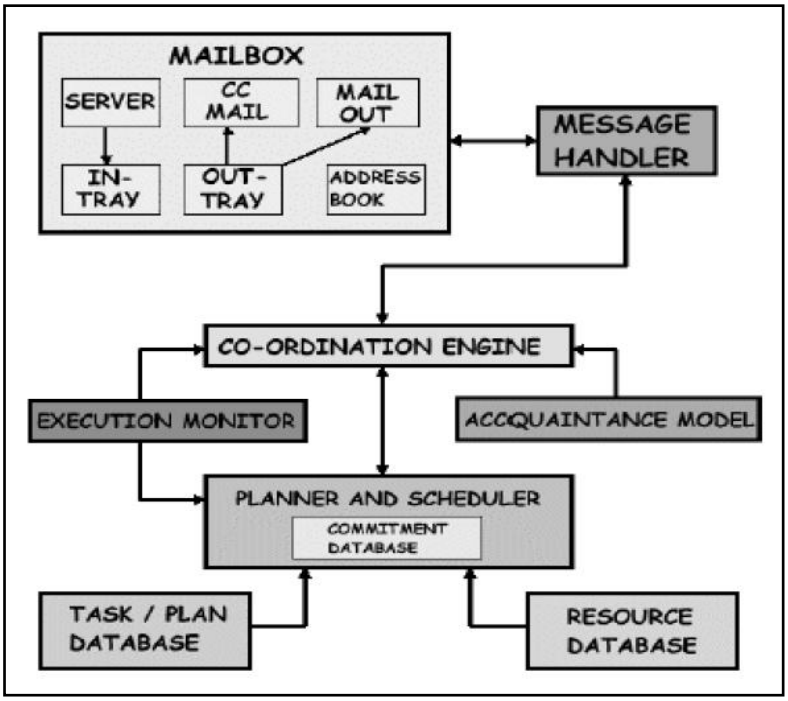

FIG. 2.33 – Architecture d'un agent de ZEUS [Collis et Ndumu, 1999d]

La méthode ZEUS utilise une décomposition en quatre étapes pour la construction d'agents. On retrouve cette forme de décomposition dans beaucoup d'autres méthodes d'analyse :

– l'analyse du domaine : Cette étape consiste à modéliser les rôles. A ce stade, le modèle UML est utilisé et aucun outil logiciel n'est fourni.

– L'étape de conception : Elle consiste à spécifier les buts et les tâches qui permettent de les réaliser. Cette spécification se fait via le langage naturel et n'est supportée par aucun outil.

– L'étape de le développement : qui comporte quatre phases : création de l'ontologie, création des agents, configuration de l'agent utilitaire, configuration de l'agent tâche et implémentation des agents. Un ensemble d'outils graphiques est fourni afin d'aider à l'élaboration de ces différentes phases.

– le déploiement : Le système multi-agents est lancé au niveau de cette phase. Pour suivre le déroulement de l'exécution, un outil de visualisation est fourni par ZEUS. Cet outil permet de visualiser l'organisation, les interactions ayant lieu dans la société d'agents, la décomposition des tâches et l'état interne des agents.

L'architecture de ZEUS s'adapte bien aux applications d'ordonnancement, ce qui la rend intéressante pour la supervision. Mais, chaque agent fonctionne grâce à une machine virtuelle, ce qui devient difficile à gérer si le nombre d'agents devient considérable.

2.3.7.4 SWARM

SWARM [Minar *et al.,* 1996] est une plate-forme relativement ancienne de simulation pour les systèmes complexes adaptatifs. Une simulation dans SWARM est constituée d'un ensemble d'agents et d'un calendrier qui indique ce qu'un agent peut envoyer comme message et à quel moment. La dynamique des échanges qui constitue un modèle dans SWARM est définie par l'association des agents au calendrier. L'environnement SWARM contient des méthodes et objets pour la simulation.

Parmi les caractéristiques principales de SWARM, on observe l'absence d'une classe agent à la base de cet outil. Ainsi, l'utilisateur dispose de la liberté de créer ses propres agents avec la sémantique qu'il souhaite. Dans SWARM, l'agent est considéré comme une extension de la notion de processus. L'originalité de ce système provient de la notion d'activité associée à un SWARM ou essaim qui permet de contrôler le déroulement des actions effectuées par les agents. Un SWARM est le composant de base du système.

Ainsi ; dans une application SWARM (voir figure 2.34), l'utilisateur peut spécifier le comportement d'un ensemble d'agents sous la forme d'un groupe d'actions déclenchées suivant un calendrier.

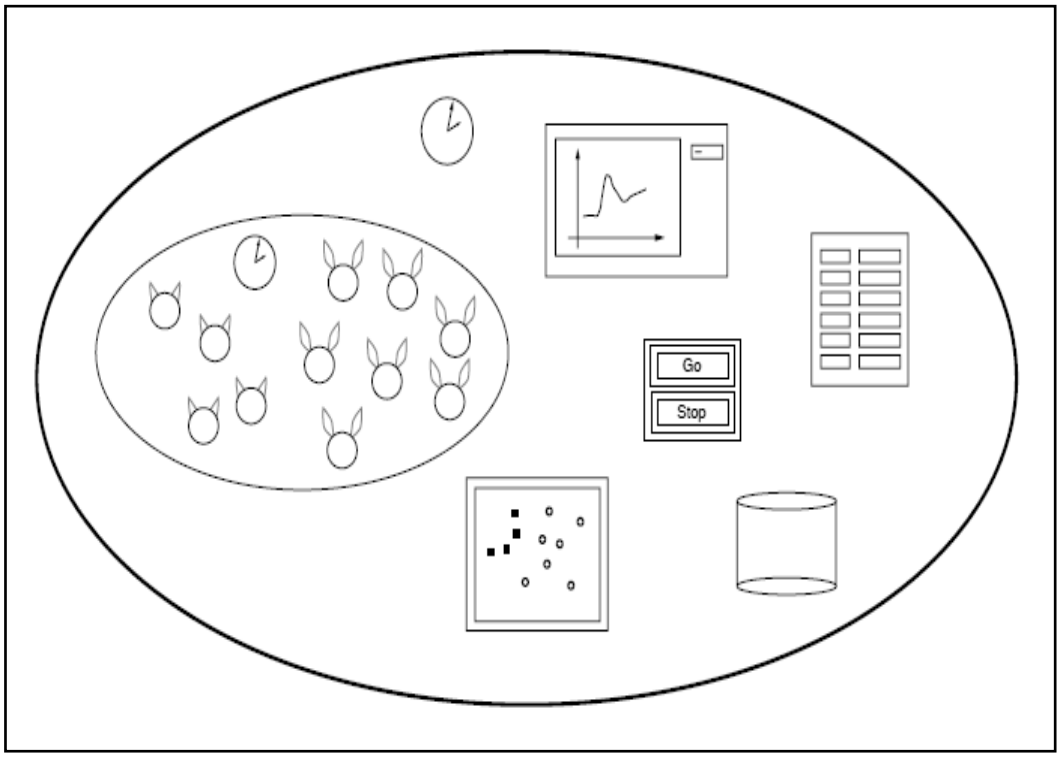

FIG. 2.34 – Architecture de SWARM [Minar *et al.,* 1996].

Sur la figure 2.34, on peut voir que la réalisation d'une application SWARM repose sur la spécification de trois types d'éléments :

– les *swarm objects* (les ellipses et les agents de la figure) qui modélisent les objets de la simulation. Un *swarm object* pourra être encapsulé dans un autre *swarm object* pour créer les hiérarchies souhaitées dans l'approche de la plate-forme.

– les *activities* (les horloges de la figure). Ces éléments sont fondamentaux dans SWARM, il s'agit d'ordonnanceurs. Chaque SWARM dispose de son propre composant *activity* auprès duquel chaque agent du SWARM enregistre ses actions et les *times tamps* associés. Les *activities* transmettent ensuite les contraintes temporelles à leur SWARM de plus haut niveau jusqu'à obtenir l'ordonnancement complet de la simulation.

– les *simtools* (les objets restant de la figure). Ces outils servent à observer et contrôler la simulation. Ils offrent aussi la possibilité de lancer la simulation selon deux modes : un mode graphique et interactif et un mode console pour lancer plusieurs simulations sur des lots de données.

2.3.7.5 JACK

JACK Intelligent Agent [Busetta *et al.*, 1999] repose sur le concept BDI Cette plate-forme est une évolution du modèle dMars [d'Iverno *et al.*, 1997]. Les agents sont dotés d'autonomie et se fixent des objectifs explicites [Busetta et Ramamohanarao, 1998]. Chaque agent dispose de :

- Un ensemble de données sur l'environnement,

- Un ensemble d'évènements auxquels il pourra répondre,

- Un ensemble d'objectifs (goals) qu'il peut souhaiter atteindre,

- Un ensemble de plans qui décrivent la façon d'atteindre ces différents objectifs.

JACK comprend un éditeur gestionnaire de projet, un langage de programmation JAL (Jack agent language) et un compilateur transformant le code JAL en Java pur [Jack, 2001a, b]. Il n'existe aucun éditeur disponible pour le développement ou le déploiement des systèmes JACK. Un inconvénient de JACK réside dans le fait qu'il soit assez long à maîtriser surtout qu'il faut bien maitriser le langage JAL et idéalement connaître le modèle BDI de dMars. JACK repose sur cinq grandes classes :

- La classe Agent : Le comportement de l'agent logiciel est défini par l'utilisation de l'agent construit. Ainsi, le type de messages, les évènements qui correspondent, et les plans à utiliser pour accomplir ses objectifs sont déterminés par cette construction.

- La classe Capacité : Elle permet de définir les éléments réutilisables par l'agent (plans, des données…).

- Les bases de données : JACK manipule des bases de données relationnelles.

- Les évènements : Leurs occurrences sont prises en compte par les agents concernés lors de leurs actions.

- Les plans : Les agents se fixent des plans d'actions à exécuter (certains y voient des analogies avec les fonctions pouvant être déclenchées à des moments déterminés).

2.3.7.6 DECAF

DECAF [McGeaty, 2001] est un environnement basé sur le développement de plans. Cet outil fournit quelques utilitaires dont un ensemble de classes servant à l'élaboration de plans et la coordination des tâches. Un planificateur applique des heuristiques pour l'exécution des tâches. Ces taches sont construites grâce à une interface. De plus, un éditeur d'agents fourni par DECAF est utilisé pour le « débuggage » [Mcgeary et Decker, 2001]. Cet environnement permet le développement d'applications Java par l'intermédiaire de classes déjà implémentées facilitant, ainsi, le développement des agents. Les limites de ce système tiennent surtout au manque de documentation (On dispose de juste une petite introduction à la programmation avec DECAF) et à l'absence de méthodologie.

2.3.7.7 CORMAS

CORMAS [Bousquet *et al.,* 1998] une plateforme SMA qui offre un environnement de développement de SMA pour des problèmes de dynamique et d'usage de ressources. Cette plateforme fournit à l'utilisateur un ensemble d'outils lui permettant de construire son modèle (en s'appuyant éventuellement sur des modèles préexistants) pour le traiter par simulation. CORMAS a été développée pour fournir un cadre multi-agent qui peut être utilisé pour simuler les interactions entre un groupe d'agents et d'un environnement partagé pour l'exploitation des ressources naturelles. Il vise à simplifier la tâche de la simulation de la gestion des ressources. La figure 2.35 présente la hiérarchie des classes génériques de CORMAS.

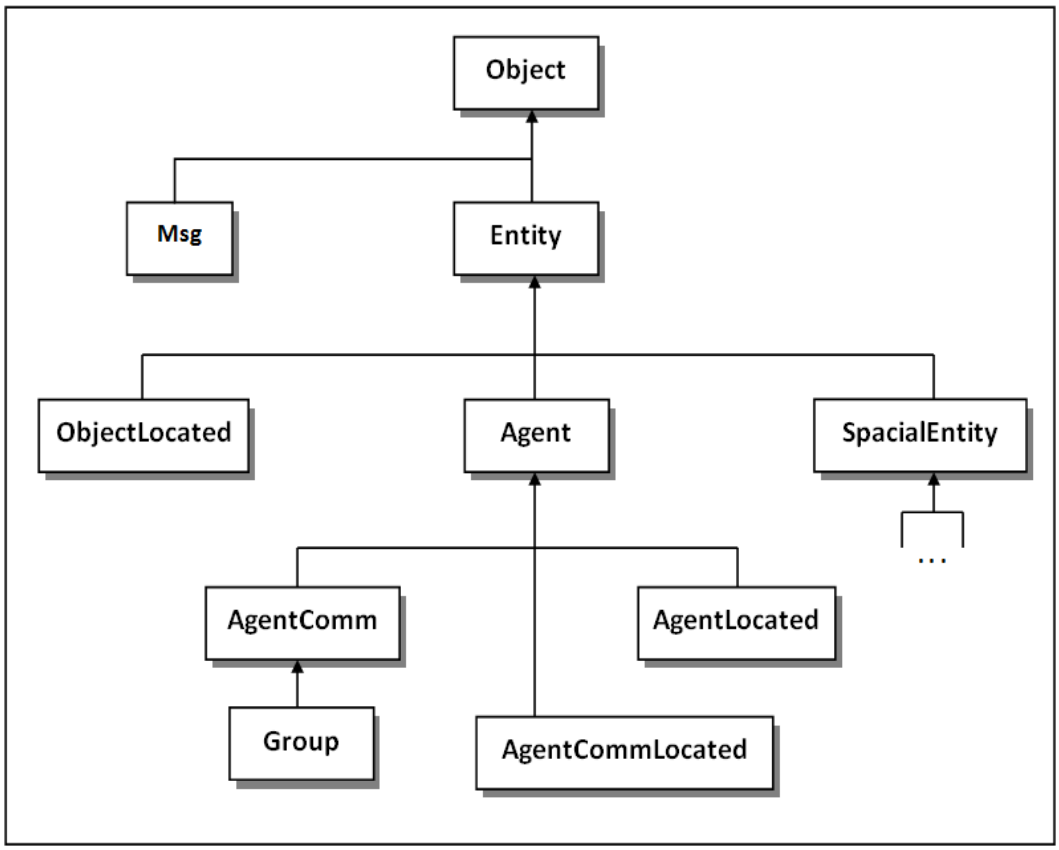

FIG. 2.35 – Hiérarchie des class génériques à CORMAS

Le cadre CORMAS se structure selon trois modules :

- le premier module qui définit les agents informatiques qui sont les entités du système à modéliser ainsi que leurs interactions. Ces dernières s'expriment par des procédures d'envois de messages, et/ou par le fait de partage du même support spatial.
- Le second module concerne le contrôle de la dynamique globale (ordonnancement des différents événements d'un pas de temps du modèle).
- Le troisième module permet de définir une observation de la simulation selon des points de vue. Cette fonctionnalité autorise l'intégration des modes de représentation dans le processus de modélisation.

CORMAS propose la facilitation du travail de construction du modèle en insérant au sein de ces trois modules des éléments prédéfinis. Parmi ces éléments, on trouve les entités-type (classes SMALLTALK génériques) à partir desquelles, par spécialisation et affinage, l'utilisateur définit d'autres entités particulières spécifiques aux besoins de ses applications [CORMAS, 2001].

2.3.7.8 MADKIT

MADKIT [Gutknecht et Ferber, 1998a, b, 2000 ; Gutknecht, 2000] est une plate-forme multi-agents implémentée en Java et basée sur le modèle organisationnel AALAADIN. Son développement fut initié par Olivier Gutknecht et Jacques Ferber au LIRMM (Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier). MADKIT est une plate-forme d'exécution de systèmes multi-agents qui utilise un micronoyau agent. Le modèle organisationnel sous-jacent est le modèle AALAADIN. Cette plate-forme a été écrite dans l'esprit même du modèle AALAADIN et en recopie tous les principes. Ici, l'organisation n'est pas le résultat émergent de l'activité du système, mais un moyen pour lui d'atteindre son but. La plate-forme MADKIT est basée sur les concepts agent, groupe et rôle (AGR). Chaque agent peut tenir différents rôles au sein de différents groupes. Les agents sont lancés par le noyau de MADKIT, qui propose notamment les services de gestion des groupes et de communication entre ces groupes. Il est ainsi possible d'échanger des messages directement entre agents ou à l'ensemble d'un groupe d'agents. Le travail de cette thèse repose en grande partie sur la plate-forme MADKIT qui sera détaillée dans les prochains chapitres.

2.3.7.9 Autres outils et environnements

Plusieurs autres outils et environnements de différents types ont été développés pour la programmation orientée-agent. En voici quelques-unes: MACE [Gasser *et al.,* 1987], NetLogo [Wilensky, 2004], Ascape [North, 2006], JAFMAS [Chauhan, 1997], GEAMAS [Marcenac, 1997; Marcenac et Giroux, 1998; Soulie *et al.,* 1998], AgentTool [DeLoach et Wood, 2001], MAST [Boissier *et al.,* 1998; Vercouter, 2004], OSACA [OSA, 2001], RMIT [Kendall *et al., 2000*], MoCAH [MOC, 2001], Brainstorm/J [Zunino et Amandi, 2000], GAMA [Taillandier *et al.,* 2010], MASON [Luke *et al.,* 2005], RePAST Symphony [North *et al.,* 2007], JEDI [Kubera *et al.,* 2008], Mobidyc [Ginot et Le Page, 1998; Ginot *et al.,* 2002]…

2.3.7.10 Évaluation de différents outils SMA

Pour Garneau et Delisle [Garneau et Delisle, 2002], les critères de l'évaluation de différents outils SMA sont liés aux objectifs qui doivent être atteints pour considérer les outils comme étant des environnements de développement de systèmes multi-agents. La figure 2.36 représente la grille d'évaluation tirée de [Garneau et Delisle, 2002] qui prend en considération les objectifs essentiels d'un environnement de développement de SMA :

– Rapidité de développement et simplification de la programmation ;

– Formalisation et abstraction des outils et mécanismes de communication, d'interaction et de coordination ;

– Faciliter la représentation et la mise en œuvre de systèmes relativement complexes ;

– Assurer la possibilité d'extension de code ;

– Garantir un ensemble support de déploiement, d'exécution, de développement, d'implémentation et de debuggage des systèmes.

Critères	JADE	DECAF	Agent/Builder	Zeus	Jack	AgenTool	MadKit
Méthodologie	0	0	4	4	0	3	3
Facilité d'apprentissage	0	3	1	1	0	3	2
Transition entre étapes	0	0	3	2	0	3	2
Souplesse de l'outil	3	0	1	1	3	0	3
Communication inter-agents	4	2	4	4	3	2	3
Outil de 'débuggage'	3	2	4	4	0	2	4
Support développement	0	2	4	4	1	4	2
Support implémentation	0	0	4	4	1	2	1
Gestion SMA	4	0	3	3	0	1	4
Effort et simplicité	2	3	2	2	2	3	1
Bases de données	0	0	1	2	3	0	0
Génèration de code	0	0	1	3	0	1	0
Extensibilité de code	4	1	1	2	4	0	3
Déploiement	4	1	2	2	2	1	3
Disponibilité de documentation	3	1	4	4	3	1	3
Total (60)	27	15	39	42	22	26	34

FIG. 2.36– Évaluation de différents outils SMA [Extrait de Garneau et Delisle, 2002]

Le barème choisi est un nombre compris entre 0 et 4 qui sont interprété selon [Garneau et Delisle, 2002] selon que la plate-forme ne réponde pas du tout au critère (0), qu'elle réponde peu au critère (1), moyennement (2), bien (3) et très bien 4. On obtient ainsi un total de points sur 60.

Il en ressort de cette grille d'évaluation les tendances suivantes :

- Les plateformes qui fournissent des GUI (Graphical User Interface) pour le développement et l'implémentation manquent de souplesse et d'extensibilité.
- Les plateformes qui fournissent des GUI pour l'implémentation diminuent l'effort de programmation.

- Ce sont les outils qui offrent une méthodologie pour le développement qui sont les plus complets.
- La plupart de ces plateformes ne supportent pas de mécanismes de sauvegarde des données.

2.4 Conclusion

Dans ce chapitre, nous avons présenté les concepts d'agent et de système multi-agents ainsi que leurs caractéristiques. De plus, nous avons étudié les méthodologies de conception des SMA et les plateformes permettant de les mettre en œuvre. Il faut noter que l'on dénote un besoin insistant de la standardisation et de l'uniformisation de ces méthodologies et plateformes. De plus, les équipes de chercheurs ont axé leurs recherches sur des sujets précis et pointus, ce qui s'est reflété sur les outils et environnements qu'ils ont développés.

CHAPITRE 3

LE FORMALISME DEVS

3.1 Introduction

DEVS (Discrete Event system Specification) est un formalisme proposée par le professeur B.P Zeigler en 1976. Ce formalisme repose sur un ensemble de bases mathématiques pour la modélisation et la simulation des systèmes à événements discrets [Zeigler *et al.,* 2000b].

Aujourd'hui, une grande communauté de chercheurs s'intéresse à ce formalisme. Aussi, ses spécifications ont évolué, de nouvelles considérations ont ainsi été ajoutées pour l'améliorer et l'adapter à la modélisation de phénomènes et de systèmes particuliers tels que les éco-systèmes, les systèmes naturels, les systèmes industriels,…La version originale de DEVS est appelée actuellement DEVS classique. D'autres versions plus récentes de DEVS telles que Parallel DEVS [Zeigler *et al.*, 2000b; Chow et Zeigler, 1994; Vangheluwe, 2004] ont été introduites pour prendre en charge un certain nombre de concepts qui n'existent pas de façon explicite dans DEVS classique.

Ainsi, dans Parallel DEVS, le cas de conflit occasionné par l'occurrence de transitions simultanées est pris en considération, chose qui n'existe pas dans DEVS classique. Ce cas est traité dans Parallel DEVS grâce à la fonction de transition-collision appelée λ (*confluent transition function)* qui agit au niveau des modèles atomiques.

Il existe plusieurs autres extensions en plus de DEVS classique et de Parallel DEVS. Parmi les plus significatives, Cell-DEVS [Wainer et Giambiasi, 2001a ; Wainer et Giambiasi, 2001b] et Dynamic Structure DEVS (DSDEVS) [Barros, 1997; Uhrmacher, 2001]. Cell-DEVS améliore DEVS en le rendant plus efficace dans la modélisation des phénomènes utilisant un partitionnement des états dans l'espace. L'espace est décomposé en cellules, le fonctionnement d'un tel système est bien adapté aux automates cellulaires. DSDEVS adapte DEVS pour la modélisation de situations qui incluent des structures à changement dynamique.

L'un des avantages de DEVS est qu'il procure non seulement un cadre modulaire et hiérarchique, mais aussi le fait qu'il détermine le concept de simulateur abstrait (c.à.d. une sémantique opérationnelle) grâce à laquelle le comportement du model peut être régénéré.

Il existe plusieurs implémentations différentes de ces formalismes qui reposent sur DEVS et qui en reprennent les fondements. Ainsi plusieurs outils DEVS pour la modélisation et la simulation sont disponibles actuellement [Zeigler et Sarjoughian, 2005; Bolduc et Vangheluwe, 2002; Nutaro, 2003], chacun ayant ses propres caractéristiques et fonctionnalités. Nous en présentons, dans ce qui suit, quelques-uns en les commentant de façon succincte.

3.2 Le formalisme DEVS

Le formalisme DEVS [Fishwick 1995 ; Vangheluwe, 2001] est une approche de modélisation basée sur la théorie générale des systèmes. Il s'agit, plus précisément, d'un formalisme modulaire et hiérarchique pour la modélisation basée sur le concept d'état.

DEVS a été élaboré et adopté depuis plus de quarante ans par une communauté internationale de chercheurs [Vangheluwe, 2001; Sarjoughian et Zeigler, 1999; Kofman *et al.*, 2000; Hild, 2000; Anglani *et al.*, 2000a; Filippi *et al.*, 2002a; Jacques and Wainer, 2002; Hamri *et al.*, 2006]. Ces travaux sont basés sur le développement d'architectures logicielles permettant d'une part de faciliter les étapes de modélisation, de simulation et de validation et d'utiliser, d'autre part, le même environnement de multi modélisation pour analyser les systèmes résultant des différents champs afin de générer automatiquement les algorithmes de simulation.

DEVS peut être considéré comme un environnement de multi-modélisation qui permet de rassembler (voire de fédérer) de manière cohérente d'autres formalismes de modélisation qui se basent eux-mêmes sur la théorie générale des systèmes. C'est, en effet, un formalisme adapté à un grand nombre de champs d'application [Barros, 1995; Uhrmarcher, 2001; Ntaimo et Zeigler, 2004; Troccoli et Wainer, 2003].

3.2.1 La modélisation DEVS

La multitude d'outils de modélisation et le désir de réutiliser des modèles existants ont motivé les chercheurs à orienter leurs travaux vers un objectif de normalisation de ces outils. Le formalisme DEVS semble être bien adapté à cette mission. La force du formalisme DEVS pourrait se résumer dans sa capacité à exprimer, grâce à la notion d'abstraction appliquée à chaque niveau, et ce, à partir des modèles atomiques ou couplés, la collaboration d'un ensemble de modèles où chaque entité interagit avec les autres. Bien qu'indépendante de l'implémentation, DEVS assure une vision modulaire et hiérarchique des modèles dynamiques. Les événements générés par un modèle peuvent prendre des valeurs dans différents domaines. Ainsi, et d'après B.P. Zeigler [Zeigler *et al.*, 2000a], nous pouvons prouver qu'il existe un modèle DEVS pour l'ensemble des systèmes à événements discrets. Mais nous pouvons aller plus loin. En effet, DEVS peut être «universel» [Touraille *et al.*, 2010], ce qui permet le couplage des modèles et formalismes décrits comme étant des paradigmes hétérogènes [Redjimi et Boukelkoul, 2013]. L'idée cruciale est que les modèles

sont considérés comme des boîtes noires qui n'ont de liens avec le monde extérieur qu'à travers les ports d'entrée et de sortie. Ces ports leur permettant d'échanger des événements et des valeurs. Grâce à cette fonctionnalité d'abstraction, plusieurs modèles peuvent être couplés tout en tirant profit de la réutilisation des modèles existants. Il est également possible de mener une vérification formelle des modèles DEVS, qui est une assistance précieuse dans la conception des systèmes [Freigassner *et al.*, 2000].

B.P. Zeigler propose dans [Zeigler *et al.*, 2000b] principalement deux formes de DEVS : classique ou avec ports. Nous emploierons dans ce qui suit, uniquement la définition de DEVS avec ports ce qui permet de mieux découper les modèles. Un modèle DEVS atomique peut, ainsi, être représenté de façon graphique par une boîte noire contenant la structure et le comportement du système. La fonction δ_{con} qui est responsable de la gestion des conflits peut être utilisée pour compléter la liste des fonctions DEVS lorsque deux événements surviennent en même temps. Cette fonction fait partie d'une extension de DEVS appelé Parallel-DEVS [Zeigler *et al.*, 2000b].
La figure 3.1 illustre un modèle DEVS en action [Zeigler et Sarjoughian, 2005].

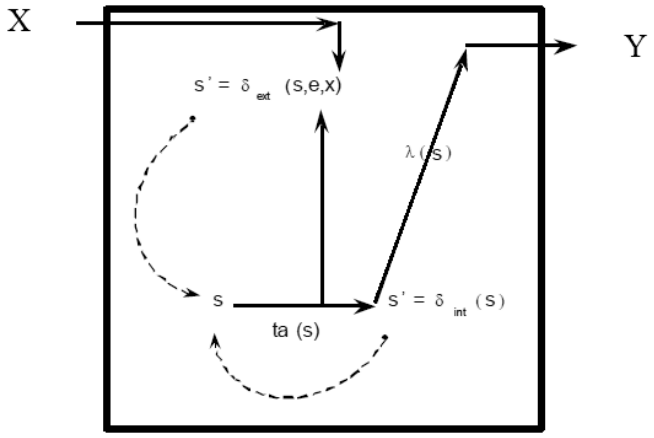

FIG. 3.1– Modèle atomique en action.

A un instant *t*, le modèle atomique est dans un état *s*. Si aucun événement externe n'intervient, le système restera dans cet état pendant durant le temps donné par la fonction ta(s). Lorsque le temps de vie du modèle expire, i.e. lorsque qu'il s'est écoulé e = ta(s), le système active sa fonction de sortie λ(s) et change d'état grâce à l'exécution de la fonction de transition interne δint (s). Si un événement externe x ϵX intervient avant que le temps ne soit expiré, i.e. quand le système est dans l'état total (s,e) avec e ≤ ta(s), le système change d'état grâce à l'exécution

de la fonction de transition externe δext (s,e,x). Dans les deux cas, le système est alors dans un nouvel état (s') avec un nouveau temps restant ta (s') et ainsi de suite.

Le temps de vie du modèle peut être quelconque entre zéro et l'infini. Dans le premier cas, on dit de s qu'il est dans un état transitoire. Dans le second cas, le système restera dans l'état s indéfiniment si aucun événement externe ne vient l'interrompre. Nous disons dans ce cas que s est un état passif.

3.3 DEVS classique

3.3.1 Spécification formelle d'un modèle atomique DEVS classique

Un modèle atomique DEVS est décrit par l'équation suivante :

$$AtomicDEVS = <X, Y, S, \delta_{int}, \delta_{ext}, t_a, \lambda> \qquad (3.1)$$

X est l'ensemble des entrées externes.

Y est l'ensemble des sorties du modèle.

S représente l'ensemble des états. Deux variables d'état sont habituellement présentes, " phase " et " sigma " (en l'absence d'événements externe le système reste dans la " phase " courante pendant le temps donné par " sigma").

δ_{int}: S → S est la fonction de transition interne qui fait évoluer le système d'un état à un autre de manière autonome.

δ_{ext}: Q × X → S est la fonction de transition externe qui se produit lorsque le modèle reçoit un événement externe. Elle renvoie le nouvel état du système basé sur l'état actuel.

Q = {(s, e) | s ∈ S, 0 ≤ e ≤ ta (s)} ensemble total des états. e : est le temps écoulé depuis la dernière transition.

λ: S → Y: fonction de sortie du modèle. Elle est activée lorsque le temps écoulé dans un état donné est égal à sa durée de vie.

t_a (s) indique la durée de vie d'un État "s" du système. C'est le temps pendant lequel le modèle demeurera dans cet état si aucun événement externe ne se produit. Quand la variable d'état de " sigma " est présente, ceci la fonction renvoie juste la valeur du " sigma ".

Un événement émis en sortie d'un modèle est interprété comme un stimulus par les modèles qui sont connectés à cette sortie. Ce stimulus est spécifié par la fonction de transition externe. La fonction de transition interne est évaluée si la durée de vie (t_a) d'un état est atteinte. La fonction de sortie (λ) est activée lors des transitions internes.

3.3.2 Spécification formelle d'un modèle DEVS couplé classique

Un modèle couplé DEVS est un modèle construit en employant d'autres modèles atomiques ou couplés. Ainsi, chaque modèle atomique DEVS peut être combiné avec un ou plusieurs modèles afin de construire un modèle Couplé. Cette opération peut être répétée afin d'obtenir une hiérarchie de modèles couplés. Ainsi, un modèle DEVS couplé peut être considéré comme un réseau hiérarchique de composants atomiques et couplés (cf. Figure 3.2). Le réseau encore appelé structure du modèle est caractérisé par les ports d'entrées et de sorties des modèles qui le constituent et les connections internes entre ces modèles.

$$CoupledDevs = <X_{self}, Y_{self}, D, \{Md/d \in D\}, EIC, EOC, IC> \quad (3.2)$$

Self: est le modèle lui-même.

X_{self}: est l'ensemble des entrées du modèle couplé.

Y_{self}: est l'ensemble des sorties du modèle couplé.

D est l'ensemble des noms associés à des éléments du modèle, *self* n'est pas en D. {Md / j ∈ D} est l'ensemble des composants du modèle couplé.

EIC, EOC et IC définissent la structure de couplage dans le modèle couplé.

EIC est l'ensemble des couplages externes en entrée. Ils relient les entrées du modèle couplé à celles de ses propres composants.

EOC est l'ensemble des couplages externes en sortie. Ils relient les sorties des composants à celles des modèles couplés.

IC définit le couplage interne. Il relie les sorties des composants aux entrées provenant d'autres composants dans le même modèle couplé. Toutefois, aucune rétroaction directe des boucles n'est autorisée. Ce qui signifie qu'un port de sortie d'un composant (modèle) ne peut pas être connecté à un port d'entrée du même composant.

La figure 3.2 présente un modèle couplé se composant de deux modèles atomiques.

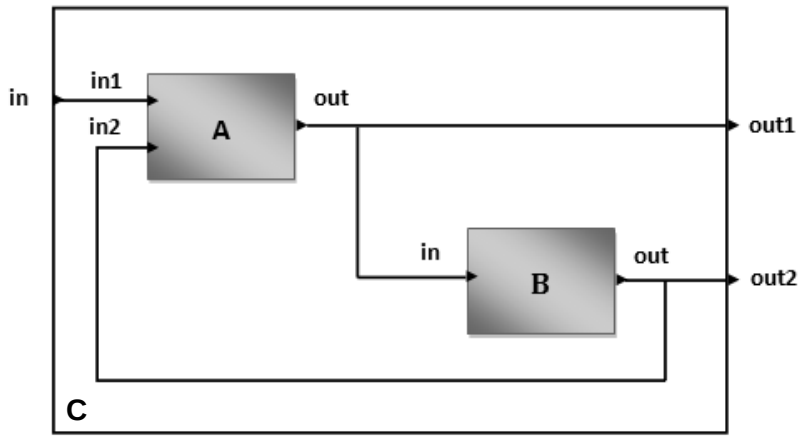

FIG. 3.2 – Représentation graphique d'un modèle couplé (C) se composant de deux modèles atomiques (A et B).

3.3.3 Simulation DEVS

Un des points forts de la modèlisation/simulation DEVS réside dans la notion de simulation abstraite. B.P. Zeigler a défini un simulateur abstrait [Zeigler *et al.*, 2000a] indépendant d'une quelconque implèmentation et apte à simuler tout modèle dècrit selon le formalisme DEVS. Il est ainsi nécessaire, pour établir une simulation, que le comportement du système à simuler soit déterminé de façon précise. De même, les différentes interactions qui existent entre les différents entités le composant doivent, elles aussi, être déterminées. Une fois ce travail fait, le principe consiste en l'élaboration automatique des algorithmes de simulation correspondants. Dans DEVS, la simulation a pour objectif de generer un ensemble d'évènements de sortie à partir d'un ensemble d'évènements d'entrée issus d'un système déterminé. L'architecture du simulateur abstrait, developpé par B.P. Zeigler repose sur le formalisme DEVS et met en œuvre un ensemble d'algorithmes qui permettent d'implémenter les instructions implicites du modèle considéré pour en élaborer le comportement [Zeigler, 1984, 1990]. C'est là un interet fondamental étant donné que la construction du simulateur est indépendante du modèle. De plus, l'architecture du simulateur abstrait telle qu'elle est proposée sépare la partie modélisation de la partie simulation au niveau de la réalisation. A chaque composant du modèle correspond un composant du simulateur.

Dans le simulateur abstrait, on trouve deux sortes d'éléments de simulation : les coordonnateurs et les simulateurs. Ces deux éléments sont appelés : les processeurs. Les simulateurs sont responsables des modèles couplés qui leurs sont liés, tandis que les coordonnateurs sont, quant à eux, responsables du contrôle des modèles atomiques qui leurs sont assujettis.

Le coordonnateur Root (racine) est un coordonnateur particulier qui contrôle tout le procédé de simulation et est lié au coordinateur du modèle couplé, de plus haut niveau. La structure de l'ensemble des processeurs représente un graphe qui est nommé 'arbre de simulation' (figure 3.3). Cet arbre a en charge l'exploitation du modèle.

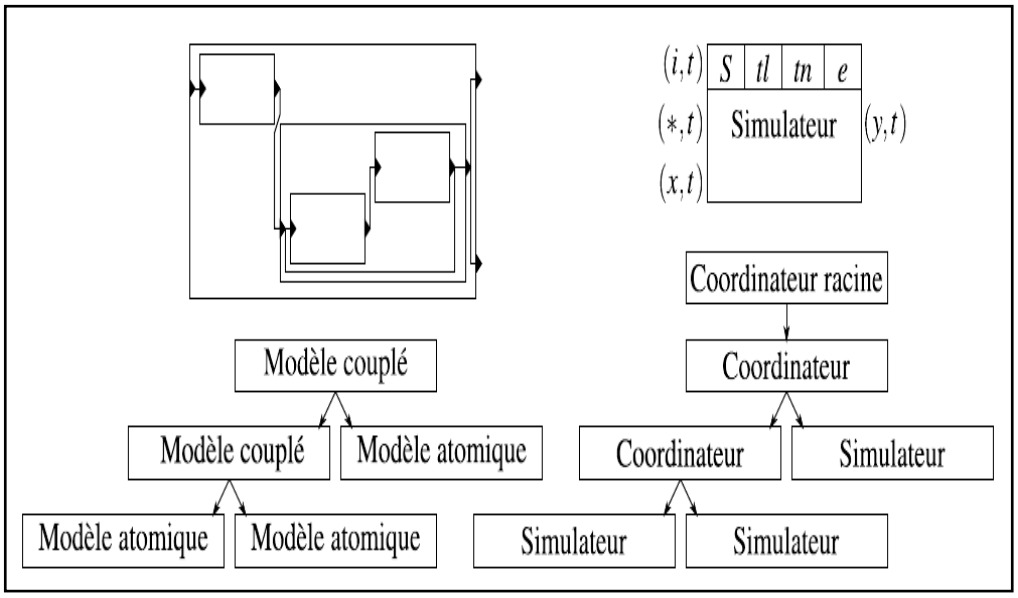

FIG. 3.3– Arbre de simulation

Chaque élément communique grâce à plusieurs types de messages : les "i"-messages initialisent les variables des processeurs, les "x"-messages symbolisent l'arrivée d'un évènement d'entrée externe sur le module, les "y"-messages symbolisent la réponse d'un module par un évènement de sortie, les "*"-messages permettent de prendre en compte les évènements internes qui vont modifier l'état du module associé au processeur concerné par le message, les "d"-messages signalent qu'une transition d'état du module concerné vient de s'achever ; le temps véhiculé par ce type de message permet la synchronisation de la simulation. On trouvera dans [Zeigler, 1976 ; Vangheluwe, 2001] leur enchainement. Un échéancier, qui est une structure de données stocke les événements générés par les messages et les classe par ordre chronologique. La tête de l'échéancier représente le futur immédiat, et la queue le futur plus éloigné. La simulation à pour rôle de faire évoluer le temps et à provoquer les changements d'états en fonction des évènements.

3.3.3.1 Algorithme d'un composant simulateur

Nous considérons ci-dessous un composant simulateur DEVS qui utilise deux variables temporelles t_l *(last time)* et t_n *(next time).* *(last time)* et t_n *(next time).* t_l *(last time)* correspond au temps de simulation du dernier événement et t_n *(next time)* au temps d'apparition du

prochain événement. En considérant la définition du temps d'avancement *ta ;* on a : $t_n = t_l + t_a$. Lorsque, de plus, le temps de simulation t est connu, le composant simulateur peut calculer le temps écoulé depuis le dernier événement : $e = t - t_l$ et le temps qui reste avant l'apparition du prochain événement : $s = t_n - t = t_a - e$. Le temps t_n est envoyé au composant coordinateur parent afin de lui permettre d'effectuer une bonne synchronisation des événements.

Algorithme 3.1 : Algorithme du simulateur DEVS.

Variables :

parent // *coordinateur parent*

t_l // *temps du dernier événement*

t_n // *temps du prochain événement interne*

DEVS =< *X,Y,S,* δ_{int} , δ_{ext} , λ, t_a>// *modèle associé*

y // *sortie courante du modèle*

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

$t_l = t - e$

$t_n = t_l + t_a(s)$

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

si ($t \neq t_n$) alors

Erreur : mauvaise synchronisation

$y = \lambda(s)$ *//stockage de la sortie avant changement d'état*

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

$s = \delta_{int}(s)$

$t_l = t$

$t_n = t_l + ta(s)$

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

si !($tl \cdot t \cdot tn$) alors

Erreur : mauvaise synchronisation

$e = t - t_l$

$s = \delta_{ext}(s,e,x)$

$t_l = t$

$t_n = t_l + ta(s)$

3.3.3.2 Algorithme d'un composant coordinateur

Un composant coordinateur a pour tâche d'assurer le bon fonctionnement et la synchronisation des ses subordonnés *(simulateurs et/ou coordinateurs qui le composent)*. Dans un but de synchronisation, ce composant utilise une liste d'événements $liste_{event}= \{(d, t_{nd}) \mid d \in D, t_{nd} \in R_+\}$. Le premier élément de la liste détermine, alors, le prochain événement du coordinateur. Le temps minimum, $t_n= min \{t_{nd} \mid d \in D\}$ est envoyé aux parents du coordinateur comme étant le temps du prochain événement. De manière similaire, le temps de l'événement précédent du coordinateur est calculé par : $t_l = max \{t_{nd} \mid d \in D\}$.

La fonction $Z_{i,j}$ permet la transmission des valeurs entre les ports des modèles *i* et *j*. Par exemple, si la valeur y_i est issue d'un port du modèle *i* et que ce port est relié à un port du modèle *j* qui attend une valeur x_j, on a alors : $Zi,j(y_i) = x_j$. L'ensemble I_i est composé des modèles qui influencent le modèle *i*. Si un modèle *i* a deux ports d'entrées reliés à deux modèles *j* et *k*, on a : $I_j= \{ j,k\}$.

Algorithme 3.2 : Algorithme du coordinateur DEVS.

Variables :

parent // *coordinateur parent*

t_l // *temps du dernier événement*

t_n //*temps du prochain événement interne*

$MC =< X,Y,D,\{M_i\},\{I_i\},\{Z_{i,j}\} >$

$Liste_{event}$ *//liste des événements* (d, t_{nd}) *(triée par* t_n *croissant)*

d //fils imminent sélectionné*

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

pour chaque modèle d dans D faire :

envoi d' un i-message (i, t) au fils d

$t_l = max \{t_{ld} \mid d \in D\}$

$t_n = min \{t_{nd} \mid d \in D\}$

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

si $(t \neq t_n)$ alors :

Erreur : mauvaise synchronisation

$d^* = premier (liste_{event})$

envoie *-message $(*, t)$ à d^*

$t_l = t$

$t_n = min\{t_{nd} \mid d \in D\}$

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

si !($t_l \leq t \leq t_n$) alors :

Erreur : mauvaise synchronisation

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

receveur = $\{r \mid r \in D, MC \in I_r , Z_{MC,r}(x) \neq 0\}$

pour chaque r dans *receveur* :

envoie x-message (x_r, t) avec $x_r = Z_{MC,r}(x)$ à r

$t_l = t$

$t_n = min\{t_{nd} | d \in D\}$

Réception y-message (y_{d*}, t) **au temps t de la part de** d*:
si $d^* \in I_{MC}$et $Z_{d*,MC}(y_{d*}) \neq 0$ alors :
envoie y-message (y_{MC}, t) avec $y_{MC} = Z_{d*},MC(y_{d*})$ au parent
receveur = $\{r \mid r \in D, d^* \in I_r, Z_{d*,r}(y_{d*}) \neq 0\}$
pour chaque r dans *receveur* :
envoie x-message (x_r, t) avec $x_r = Z_{d*,r}(y_{d*})$ à r

Algorithme 3.3 : Algorithme du coordinateur *"Root"*.

Variables :

t //*temps de simulation*

fils //*subordonné directe*

t_n // 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

$t = t_n$

3.2.4 Limites de DEVS classique

Dans un schéma général comportant un ensemble de modèles DEVS, une transition interne pourrait survenir à une même date dans plusieurs de ces modèles. Cependant, dans DEVS classique, en cas d'*événements simultanés* le lien de causalité entre modèles peut être brisé. Ce cas de figure peut s'expliquer aisément de la manière suivante : Si un modèle est influencé par plusieurs autres, la fonction 'select' qui interdit la simultanéité des évènements aura pour effet que les événements de sortie des influenceurs ne seront pas reçus par l'influencé à la même date. De même, un événement externe peut parvenir aux ports d'entrées en même

temps que celle prévue pour la transition interne. Pour pallier ces limites, le formalisme DEVS parallèle [Chow et Zeigler, 1994].

3.4 DEVS parallèle

Le formalisme DEVS parallèle (*Parallel Discrete EVent system specification* PDEVS), [Chow et Zeigler, 1994] est un formalisme qui reprend les principes de base de DEVS classique. Ce formalisme étend DEVS classique à la prise en compte de conflits entre les fonctions de transitions internes et externes. PDEVS donne la possibilité au modélisateur de spécifier une fonction de conflit pour traiter les transitions internes et externes simultanées. De plus, afin de pallier les problèmes liés aux événements simultanés dans DEVS classique, on dispose dans PDEVS de *bags* d'événements destinés à la fonction de transition externe. Ces *bags* permettent de collecter les événements émis à une même date. Donc, avec formalisme PDEVS, plusieurs événements externes peuvent être réalisés en même temps. Ces événements sont stockés comme des ensembles d'événements survenus en même temps. Les sorties réalisées par les modèles concernés sont collectées dans des *bags* d'entrées (I^b). Chaque événement du *bag* est identifié par sa date d'occurrence. Ainsi, aucune relation d'ordre n'est préconisée pour les événements appartenant à un même *bag*. Donc, on peut autoriser les évènements simultanés. La prise en compte des événements du *bag* n'est autorisée qu'après les transitions internes.

3.4.1 Spécification formelle d'un modèle atomique DEVS parallèle

Le modèle DEVS atomique parallèle M est décrit par un tuple

$$M= <X,Y, S, \tau, \delta_{int} , \delta_{ext} , \delta_{con}, \lambda> \qquad (3.3)$$

X est l'ensemble des ports x et des valeurs d'entrées,
Y est l'ensemble des ports y et des valeurs de sorties,
S est l'ensemble des états partiels du système,
$\tau : S \rightarrow R_0^+$ est la fonction d'avancement du temps,
$\delta_{int} : S \times S$ est la fonction de transition interne,
$\delta_{ext} : Q \times X^b \rightarrow S$ est la fonction de transition externe,
X^b est l'ensemble des *bags* des entrées X,
Q est l'ensemble des états totaux,
$Q = \{(s, e) | s \in S, (0 \leq e \leq \tau (s)\}$,
e est le temps écoulé depuis la dernière transition,

δ_{con} : S ×X^b → S est la fonction de conflit,

λ : S → Y^b est la fonction de sortie.

En l'absence d'événements sur les ports d'entrées, le système conserve un état passif jusqu'à la prochaine transition interne. Une sortie est alors réalisée suivie de la transition interne. Si un événement externe apparait sur l'un des ports d'entrée avant la date prévue pour la transition interne, l'état du système passe à δ_{ext} (s, e, x^b). Dans PDEVS, à la différence de DEVS classique, la transition externe se base sur les *bags* d'événements provenant d'un ou de plusieurs modèles. Si un événement apparait sur X^b à $e = \tau$ (s), le système passe dans l'état δ_{con} (s, e, x^b). Le comportement de la fonction de conflit est défini par le modélisateur. Par défaut $\delta_{con} = \delta_{ext}(\delta_{int}(s, e), 0, x^b))$, donnant ainsi la priorité à la transition interne lors d'un conflit.

3.4.2 Spécification formelle d'un modèle couplé PDEVS

Un modèle couplé dans PDEVS parallèle a une structure quasiment identique à celle de la version classique. Les modèles atomiques sont tous parallèles.

Un modèle DEVS couplé M est décrit par un tuple

$$M = \langle X, Y, D, \{M_n\}, \{I_n\}, \{Z_{n,n'}\} \rangle \qquad (3.4)$$

Où

X est l'ensemble des ports et des valeurs d'entrées,

Y est l'ensemble des ports et des valeurs de sorties,

D est l'ensemble des identifiants des modèles constituant le modèle couplé,

M, y compris l'identifiant « self » de M lui même,

M_n est un modèle DEVS (atomique/couplé) constituant du réseau hiérarchique et indexé par n ∈ D,

I_n est l'ensemble des modèles qui influencent le modèle n,

$Z_{n,n'}$ est une famille de fonctions de transfert telles que :

$Z_{n,n'}$: $Y_n \rightarrow X_{n'}$ si n, n' ∈ D et n ∈ $I_{n'}$,

$Z_{self,k}$: $X \rightarrow X_k$ si k ∈ D et self ∈ I_k,

$Z_{k,self}$: $Y_k \rightarrow Y$ si k ∈ D et k ∈ I_{self} .

Une conséquence directe de la suppression de la fonction de sélection est la prise en compte des modèles concernés. En effet, tous ces modèles sont autorisés à effectuer une sortie [Akplogan, 2013].

3.5 Cell-DEVS

Une autre extension de DEVS classique est Cell-DEVS qui est née de la constatation que de nombreux modèles font intervenir des espaces discrets et utilisent des formalismes tels que les automates cellulaires. Cette extension a été développée par Wainer et Giambiasi dans [Wainer et Giambiasi, 2001a]. Le but de cette extension est de pouvoir simuler des automates cellulaires multidimensionnels. Cell-DEVS fournit un mécanisme simple de définition de la synchronisation des cellules. La dynamique de ces dernières est dans le sens où l'état d'une cellule d'un automate cellulaire sera modifié selon l'état de son voisinage mais cet état ne sera connu des cellules voisines qu'après un certain délai.

Une extension Cell-DEVS est basée sur le modèle DEVS classique. On retrouve les modèles atomiques, les éléments de base d'un réseau et les modèles couplés eux-mêmes. Néanmoins, la structure proposée par Zeigler [Zeigler *et al.*, 2000b] se voit augmentée d'attributs.
Le modèle DEVS d'une cellule est défini par la structure suivante :

$$TDC = <X, Y , I, S, N, delai, d, \delta_{int}, \delta_{ext}, \tau, \lambda, t_a> \qquad (3.5)$$

où :

– *X* est l'ensemble des ports et des valeurs d'entrée ;

– *X* est l'ensemble des ports et des valeurs de sortie) ;

– *I* l'interface de la cellule ;

– *S* l'ensemble des états de la cellule ;

– *N* l'ensemble des états des cellules voisines ;

– *delai* le type de délai utilisé ;

– *d* la durée du délai ;

– δ_{ext} et δ_{int} la fonction de transition externe et interne ;

– τ la fonction locale de calcul ;

– λ la fonction de sortie ;

– t_a la fonction d'avancement du temps.

Le voisinage d'une cellule ainsi que ses connexions en termes de ports sont déterminés par son interface *I*. Il y a autant de ports d'entrée que de voisins. La fonction τ modélise la fonction de calcul de l'état de la cellule en fonction de l'état de son voisinage. Cet état d'une cellule n'est effectif (stable) pour les cellules voisines qu'au bout du délai d'attente *d*. La figure 3.4 représente ce fonctionnement.

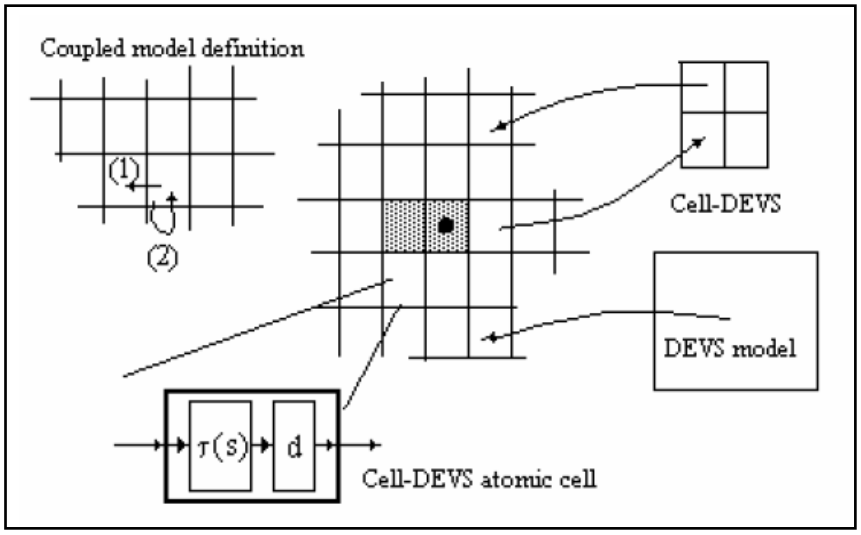

FIG. 3.4 – Description d'un modèle Cell-DEVS [Wainer, 2000]

La définition d'un modèle couplé complète la définition des cellules. Ainsi, Le modèle d'une cellule détermine son interface vue de l'extérieur mais ne spécifie pas la forme des connexions. C'est le modèle couplé qui définit les connexions entre les cellules :

$$GCC = <X_{list}, Y_{list}, I, X, Y, n, \{t_1, ..., t_n\}, N, C, B, Z> \qquad (3.6)$$

Dans cette équation, on a X_{list} et Y_{list} qui déterminent les listes des modèles (cellules) du réseau qui possèdent des ports d'entrée et des ports de sortie non connectées en interne et, par conséquent, disponibles pour une connexion avec un autre modèle. Dans DEVS, c'est cet ensemble qui détermine toutes les connexions entre les ports d'entrée et les ports de sortie avec les ports du modèle couplé.

I est l'interface *I* du réseau qui réunit au niveau d'une même structure les éléments de définition de l'interface du réseau vers l'extérieur.

Z met en relation les ports de sortie qui sont des éléments de Y_{list} et les ports d'entrée de X_{list} d'un autre réseau.

X et *Y* : ensemble des événements d'entrée et de sortie.

Pour simplifier la définition des connexions entre les cellules du réseau, un «*pattern*» de voisinage, noté *N*, est défini. Ce «*pattern*» spécifie pour toute cellule n'appartenant pas à la bordure *B* la position relative de ses voisins.

Cell-DEVS est un outil puissant utilisant DEVS dans le domaine des automates cellulaires. Cell-DEVS offre aussi des simulateurs abstraits afin de préciser le comportement d'un modèle [Versmisse , 2008].

3.6 Formalismes DEVS pour la modélisation approximative

3.6.1 Fuzzy-DEVS

Le formalisme fuzzy-DEVS [Kwon *et al.,* 1996] est basé sur la logique Floue et sur la règle "Max-Min". Il dérive du formalisme DEVS classique et en conserve la sémantique, les concepts et la modularité. Ce formalisme permet la modélisation et l'analyse de systèmes complexes à évènement discret avec une structure et un comportement partiellement inconnus.

Dans Fuzzy-DEVS, indique pour chaque état où le système à analyser se trouve, l'ensemble des évolutions possibles du système (le modèle garde la trace des évolutions possibles) [Anglani *et al.,* 2000a]. La structure du modèle de base "modèle atomique flou" est :

$$A\tilde{M}F = <X, Y, S, \tilde{\delta}_{int}, \tilde{\delta}_{ext}, \tilde{\lambda}, \tilde{t}_a> \qquad (3.7)$$

Avec :

- X, Y : l'ensemble des évènements d'entrées sorties ;
- S : l'ensemble des états séquentiels ;
- $\tilde{\delta}_{int} : S \mathbf{x} S \rightarrow [0, 1]$: fonction floue de transition interne ;
- $\tilde{\delta}_{ext}: Q \mathbf{x} X \mathbf{x} S \rightarrow [0, 1]$: fonction floue de transition externe, avec $Q = \{ (s, e) \mid s \in S, 0 \leq e \leq t_a(s)\}$, ou $t_a(s)$ est la valeur de defuzzyfication de $\tilde{t}_a(s)$;
- $\tilde{\lambda} : S \mathbf{x} Y \rightarrow [0, 1]$: fonction floue de sortie ;
- $\tilde{t}_a: S \mathbf{x} \tilde{N} \rightarrow [0, 1]$: fonction floue d'avancement du temps, où . N est l'ensemble des nombres flous appartenant à $R_{[0, \infty[}$.

fuzzy-DEVS étend les fonctions caractéristiques de DEVS aux ensembles flous.

(1) Les fonctions floues de transition interne et externe ($\tilde{\delta}_{int}$, $\tilde{\delta}_{ext}$) représentent les possibilités de transition entre chaque état (passage de s_t à s_t+1), elles sont exécutées pour $\tilde{\delta}_{int}$ lorsque le temps ($\tilde{t}_a$) est écoulé et pour $\tilde{\delta}_{ext}$ lorsque un évènement externe arrive avant que le temps ($\tilde{t}_a$) ne soit écoulé.

(2) La fonction floue de sortie $\tilde{\lambda}$ génère les valeurs possibles de sortie Y_t, d'un état s_t.

(3) La fonction floue d'avancement du temps peut être représentée par des variables linguistiques (on introduit un flou au niveau de la date de l'évènement).

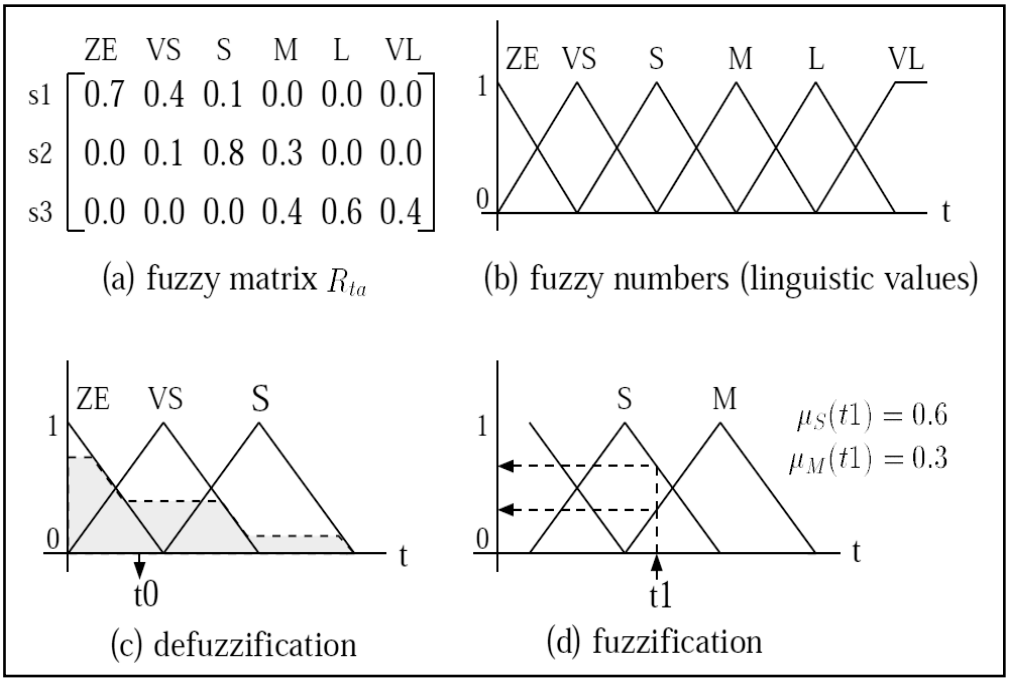

FIG. 3.5– Exemple de données fuzzy-DEVS [Kwon *et al.*, 1996]

L'examen de la figure (3.5, a) fait ressortir les valeurs d'avancement dans le temps des états s1, s2 et s3 sous forme matricielle. On associe à chaque valeur, dans le temps, un terme linguistique (figure 3.5, b) (ZE : Zero, VS : Very Small, S : Small, M : Medium, L : Large, VL : Very Large). Si l'on prend l'exemple de la probabilité qu'à l'état s3, on ait la valeur ' L' est 0,6. Afin d'obtenir des valeurs réelles telles que représentées sur la figure 3.5, c ; on utilise la méthode de "defuzzication" et inversement la méthode de "fuzzification" nous permet d'obtenir valeurs floues (figure 3.5, d).

Contrairement au modèle atomique DEVS, Le modèle *A ˜MF*, est non déterministe, en ce sens qu'il ne répond pas aux deux conditions suivantes :

(1) A l'expiration du temps, la fonction de transition interne est exécutée ($\delta_{int}(s_t) = s_t+1$). De même ; lorsqu'un évènement externe arrive avant que n'a pas expiré ; la fonction de transition externe ($\delta_{ext}(s_t,X_t) = s_t+1$) est exécutée.

(2) Quand la durée de vie d'un état est finie ; la fonction de sortie ($\lambda(s_t) = Y_t$) est exécutée.

La règle "Max-Min" détermine, dans fuzzy-DEVS l'état suivant s_t+1 et non avec δ_{int} et δ_{ext} [Kwon *et al.*, 1996].

On distingue deux types de modèles dans Fuzzy-DEVS tout comme dans DEVS-classique, un modèle atomique et un modèle couplé, ce dernier ayant la même structure qu'un modèle couplé DEVS :

$$CM = <X, Y, M, EIC, EOC, IC, SELECT> \qquad (3.8)$$

Les différentes possibilités d'entrées, de sorties et changements d'état dans fuzzy-DEVS sont représentées par des matrices. Des arbres de probabilités, représentent les évolutions des modèles [Kwon *et al.,* 1996 ; Anglani *et al.,* 2000a]. Les arbres de probabilités génèrent toutes les trajectoires possibles, les algorithmes qui les mettent en œuvre ne sont pas toujours très performants et constituent des domaines de recherche très actifs. Ce formalisme est appliqué dans plusieurs domaines tels que : analyses de marché (prévision de prix ou de cours), aide à la décision [Anglani *et al.*, 2000a, Anglani *et al.*, 2000b]. Cependant, cette approche qui utilise la règle "Max-Min", les fonctions "fuzzyfication" et "defuzzification", ne semble pas, pour beaucoup de chercheurs totalement cohérents avec le formalisme DEVS. D'autant plus qu'un modèle fuzzy-DEVS ne possède pas la propriété de fermeture sous composition du formalisme DEVS classique. Pour exécuter une simulation, les paramètres flous doivent être remplacés par des paramètres réels (defuzzyfication). Pour effectuer la procédure inverse, on procède par la méthode de fuzzyfication.

3.6.2 Min-Max-DEVS

Le formalisme Min-Max-DEVS [Giambiasi et Ghosh, 2001 ; Hamri *et al.,* 2006] est une suite des travaux concernant la modélisation et la simulation de circuits logiques à retard flou [Smaili, 1994]. Le but poursuivi par ce formalisme est la modélisation et la simulation des systèmes réels pour lesquels les valeurs des retards ne sont pas connues avec exactitude. Min-Max-DEVS est applicable dans les systèmes pour lesquels la durée de vie des états transitoires est représentée par des intervalles de temps.

Les modèles atomiques Min-Max-DEVS et DEVS classique sont quasiment identiques, la seule différence réside dans la représentation de la fonction d'avancement du temps t_a. Dans Min-Max-DEVS elle est définie comme suit :

– $t_a(s_i) : S \rightarrow R^+ \times R^+$;
– $t_a(s_i) = (d_{min}, d_{max})$;
Où

- d_{min} est le temps minimum pendant lequel le modèle reste dans le même état s_i, d_{min} représente l'évolution la plus rapide du système ;
- d_{max} est le temps maximum pendant lequel le modèle reste dans le même état s_i, d_{max} représente l'évolution la plus lente du système.

On distingue deux modifications importantes relativement au formalisme DEVS classique.
Le premier concerne l'évolution du modèle ; dans Min-Max-DEVS, les évènements externes sont choisis en fonction du temps minimum ; en l'absence d'évènement externe, un évènement interne est déclenché au temps maximum. Le second changement concerne les algorithmes de simulation, la fonction t_a ayant été modifiée, il y'a lieu d'en tenir compte au niveau de la simulation. La fonction t_a fixe tous les temps correspondants aux évènements internes. Un évènement est déclenché en fonction du temps. Ainsi, deux nouvelles variables de temps ont été définies : *ts* représente le temps maximum avant le prochain évènement et *tf* représente le temps minimum avant le prochain évènement. Les algorithmes de simulation ont été modifiés pour prendre en compte ces évolutions [Giambiasi et Ghosh, 2001].

Min-Max-DEVS présente des pistes intéressantes pour la représentation des imprécisions au niveau du temps de vie de l'état $t_{a.}$. Cependant, ceci est applicable principalement dans un nombre restreint de domaines tels que la détection de fautes ou la prise en compte de retards flous dans les circuits numériques [Bisgambiglia, 2008].

3.7 Real-Time DEVS (RTDEVS)

Le formalisme en temps réel de DEVS (RTDEVS) est une extension de formalisme classique de DEVS seulement dans les modèles atomiques de DEVS. Le formalisme RTDEVS pour les modèles couplés reste le même comme l'original sauf qu'un modèle couplé de RTDEVS n'a aucune spécification. C'est parce qu'une horloge de simulation dans RTDEVS n'est plus une horloge virtuelle mais une horloge en temps réel, qui n'est pas commandée par un algorithme de simulation. Un modèle atomique de RTDEVS, RTAM, est défini comme suit :
La structure d'un RTDEVS atomique est donnée par :

$$RTAM = <X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta, ti, \psi, A> \qquad (3.9)$$

X est l'ensemble d'événements d'entrée
Y est l'ensemble d'événements de sortie
S est l'ensemble d'états séquentiels
$\delta_{ext} : Q \times X \rightarrow S$ est la fonction de transition externe d'état
$\delta_{int} : S \rightarrow S$ est la fonction de transition interne d'état
$\lambda: S \rightarrow Y$ est la fonction de sortie
$ta : S \rightarrow R_0^+ \cup \infty$ est la fonction d'avancement de temps
$ti : S \rightarrow R^+_{0,\infty} \cup R^+_{0,\infty}$ est la fonction d'avancement d'intervalle du temps
$\psi : S \rightarrow A$ est la fonction de mappage de l'activité

A : est l'ensemble d'activités avec des contraintes

Dans le formalisme DEVS classique, le temps de simulation avance seulement lorsqu'un simulateur appelle la fonction d'avancement de temps *'ta'* du modèle associé. La fonction d'avancement de temps *'ta'* dans le formalisme de RTDEVS se comporte la même que cela dans le formalisme DEVS classique sauf que la fonction calcule l'événement suivant des nombres entiers, qui est un nombre réel dans le formalisme DEVS classique. Le temps calculé par la fonction d'avancement de temps aussi synchronisée avec le temps de l'horloge murale.

3.8 Formalismes DEVS à structures dynamiques

3.8.1 DS-DEVS

Le formalisme DS-DEVS [Barros, 1995] (voir figure 3.6) est basé sur le formalisme DEVS classique et fournit tous les mécanismes pour le changement dynamique de la structure d'un modèle DEVS. Dynamic Structure DEVS introduit une nouvelle spécification pour les modèles couplés mais cette spécification n'effectue aucune modification de spécification des modèles atomiques. Un modèle couplé particulier appelé 'exécutive' est prévu dans le formalisme DS-DEVS. A chaque état du modèle couplé 'exécutive' est associé à une structure du modèle couplé dont la dynamique dépend des transitions internes et externes de l'exécutive [Barros, 1996, 1997, 2003].

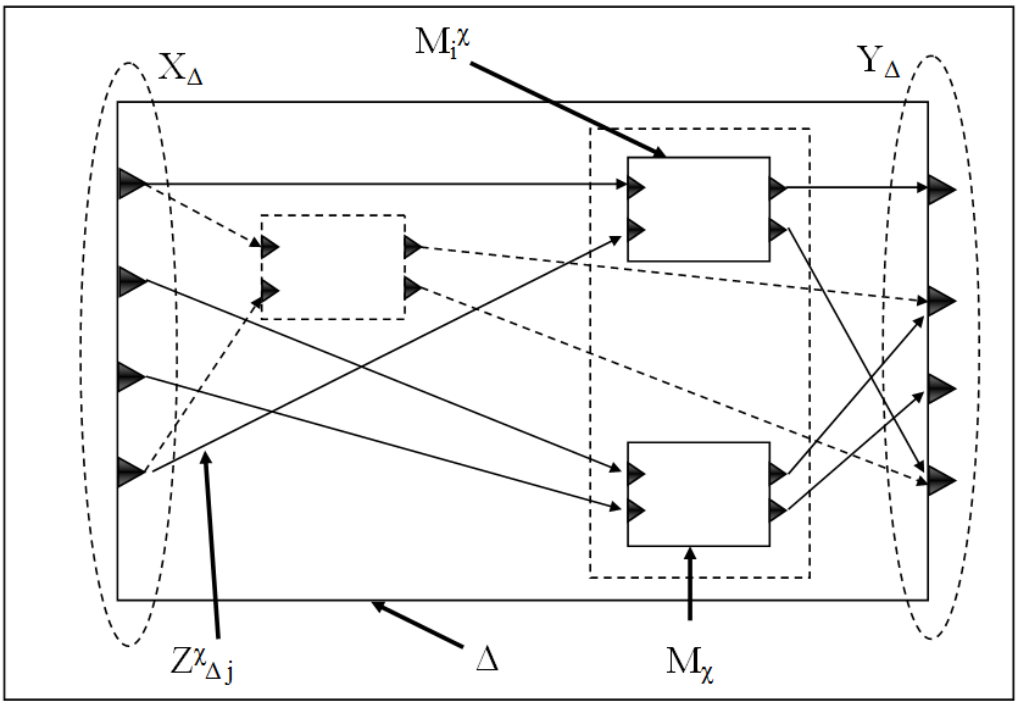

FIG. 3.6 – Représentation graphique d'un modèle DS-DEVS.

Le changement dynamique de structures est défini à l'aide d'un réseau de modèles DEVS atomiques. Ainsi, à la différence des modèles couplés DEVS, les listes de connexions et de modèles du réseau peuvent changer au cours du temps. Ce réseau est représenté par la structure suivante :

$$DSDEVN = <X_{\Delta}, Y_{\Delta},\chi,M_{\chi}> \qquad (3.10)$$

Où

Δ est l'ensemble des modèles atomiques activables, χ est le nom de l'exécutive et $M\chi$ le modèle associé à l'exécutive χ. Ce couple détermine la manière dont la structure change au cours du temps. Le modèle M_{χ} est défini par un tuple

$$M_{\chi} = <X_{\chi},Y_{\chi}, S_{\chi}, s^{0}_{\chi}, \tau_{\chi}, \gamma, \Sigma^*, \delta_{int,\chi}, \delta_{ext,\chi}, \lambda_{\chi}> \qquad (3.11)$$

Où

X_{χ} est l'ensemble des ports et des valeurs d'entrées,

Y_{χ} est l'ensemble des ports et des valeurs de sorties,

S_{χ} est l'ensemble des états partiels du système,

s^{0}_{χ} est l'état partiel initial du système. Cet état se compose de :

- $D\chi$: l'ensemble des composants (nom des modèles composants le modèle dynamique) actifs.
- M^{χ}_{i} : modèle du ième composant actif.
- I^{χ}_{i} : l'ensemble des composants sous l'influence du ième composant actif.
- $Z^{\chi}_{i,j}$: fonction de traduction entre les ports du modèle du composant i et ceux du modèle du composant j.
- Select : fonction de gestion des conflits entre composants
- θ^{χ} : ensemble de variables d'état du modèle global

$\tau_{\chi} : S_{\chi} \rightarrow R^{+}_{0}$ est la fonction d'avancement du temps,

$\gamma : Q_{\chi} \rightarrow \Sigma^*$ est la fonction de structure,

Σ^* est l'ensemble des structures,

$\delta_{int,\chi} : S_\chi \times S_\chi$ est la fonction de transition interne,

$\delta_{ext,\chi} : Q_\chi \times X_\chi \rightarrow S_\chi$ est la fonction de transition externe où :

Q_χ est l'ensemble des états totaux,

$Q_\chi = \{(s_{\alpha,\chi}, e) | s_{\alpha,\chi} \; 2 \; S_\chi, \; 0 \leq e \leq \tau(s_{\alpha,\chi})\}$

e est le temps écoulé depuis la dernière transition,

$\lambda_\chi : S_\chi \rightarrow Y_\chi$ est la fonction de sortie.

Nous remarquons qu'à l'exception de Σ^* et γ, cette structure est assez proche de celle des modèles atomiques DEVS classiques. L'état $s^{\alpha}_{\chi} \in S_\chi$ contient des informations relatives à la structure du réseau de modèles. Si l'on considère un état, s^{α}_{χ}, la structure du réseau de modèles $\Sigma^{\alpha} \in \Sigma^*$ est définie par $\Sigma^{\alpha} = \gamma\ (s^{\alpha}_{\chi}) = (X^{\alpha}_{self}, Y^{\alpha}_{self}, D^{\alpha}, \{M^{\alpha}_d / d \in D\}, (EIC)^{\alpha}, (EOC)^{\alpha}, (IC)^{\alpha})$, ces paramètres sont les mêmes que ceux décrits dans la définition du modèle DEVS couplé . Le réseau est défini par une composition hiérarchique de modèles DEVS atomiques et couplés. Tout changement provoqué par l'appel de $\gamma(s^{\alpha}_{\chi})$ se traduit par l'ajout ou la suppression de modèles et le changement de connexions entre modèles.

3.8.2 DSDE

Le formalisme DSDE [Barros, 1997, 1998] est une version parallèle de DS-DEVS. Comme dans PDEVS, à chaque modèle est associée une fonction de conflit capable de prendre en compte des transitions interne et externe simultanées. DSDE intègre aussi la notion de *bags* permettant ainsi la gestion et la collecte des événements qui sont émis en même temps. Ce formalisme peut aussi piloter des modèles atomiques PDEVS [Akplogan, 2013].

3.9 Plateformes et environnements basée DEVS

3.9.1 JDEVS

JDEVS [Filippi *et al.,* 2002b ; Filippi, 2003 ; Filippi et Bisgambiglia, 2004] est issu d'un travail de thèse portant sur le développement d'une plateforme basée sur un environnement de modélisation et de simulation de systèmes naturels complexes. JDEVS propose un ensemble de solutions pour la communication mettant en jeu des données fournies par des composants externes. Les modèles intégrés par JDEVS sont les réseaux de neurones et les automates cellulaires. Cet environnement permet aussi le couplage de plusieurs types de modélisation grâce à l'utilisation d'interfaces d'entrées/sorties bien définies.

3.9.2 ADEVS

ADEVS [Nutaro, 2003] a été développé par J. Nutaro à l'Université d'Arizona. C'est une bibliothèque dédiée à la construction de simulations basées sur PDEVS et *Dynamic* DEVS (dynDEVS) [Uhrmacher, 2001]. ADEVS constitue une extension pour le support de la gestion de structures dynamiques de modèles basée sur *Dynamic* DEVS.

3.9.3 DEVS/C++

Cette bibliothèque a été développée à l'Université d'Arizona [Zeigler *et al.,* 1996] avec la possibilité d'utiliser les extensions de structures dynamiques et d'automates cellulaires en laissant la possibilité d'être distribuable sur plusieurs calculateurs. DEVS/C++ constitue l'une des premières plateformes basées sur le formalisme DEVS. Elle est considérée comme une implémentation efficace des simulateurs abstraits et propose différents mécanismes d'optimisation. Plusieurs versions existent de cette plateforme.

3.9.4 CD++

CD++ [Wainer, 2002] est un ensemble de bibliothèques qui permet la définition de modèles DEVS et Cell-DEVS, par l'utilisation d'un langage de spécification de haut niveau par la manipulation de graphes d'états. Certaines versions de CD++ incorporent les extensions PDEVS et RTDEVS [Glinsky et Wainer, 2002]. Dans CD++, les modèles peuvent se baser sur les automates cellulaires, les graphes d'états DEVS, ou les réseaux de Petri.

3.9.5 ATOM^3

Atom^3 [de Lara et Vangheluwe, 2002] est une plate-forme de modélisation et de simulation développée par Juan De Lara à l'Université Autonome de Madrid (UAM) et Hans Vangheluwe à l'Université Mc. Gill au Canada. Dans cette plateforme, les modèles et les formalismes, sont décrits sous forme de graphes. Atom^3 propose des outils de manipulation des modèles décrits dans ces formalismes. Les méta-modèles fournis sont, entre autres, les automates à états finis, les réseaux de Petri, les diagrammes de flux… Les modèles peuvent alors être automatiquement traduits vers le formalisme DEVS par des routines de traduction [Vangheluwe, 2000]. Dans ce cas, DEVS est utilisé de la même manière qu'un formalisme unificateur en transformant les modèles sous forme DEVS.

3.9.6 DEVS/JAVA

DEVS/JAVA [Sarjoughian et Zeigler, 1998] supporte plusieurs extensions de DEVS et permet l'extension de la modification de la structure des modèles atomiques et couplés tels que l'ajout d'autres modèles et leur suppression. Les modifications des ports de ces modèles peuvent aussi être modifiés. D'autres extensions sont aussi incluses comme l'intégration d'automates cellulaires et des systèmes d'équations différentielles. DEVS/JAVA permet aussi le développement collaboratif par composition de manière synchrone ou asynchrone [Quesnel, 2006].

3.9.7 MOOSE

MOOSE [Fishwick, 1998] est un autre environnement de modélisation et de simulation qui a été développée à l'université de Floride. MOOSE comprend une interface graphique, une bibliothèque de modèles et un moteur de modélisation et de simulation. Un aspect important de l'environnement MOOSE concerne le fait qu'il 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.

3.9.8 ECLPSS

ECLPSS [Woodbury *et al.*, 2002] est un environnement basé sur le langage Java. Cet environnement est orienté vers la simulation d'écosystèmes prenant en considération de nombreuses échelles de temps et d'espace. Trois entités composent les modèles Eclpss: un ensemble de composants avec état modifiable, un ensemble de variables et un ensemble de simulateurs pour ces composants. Les composants Eclpss sont développés en java et stockés dans une bibliothèque pour réutilisation [Filippi, 2003].

3.9.9 Small DEVS

SmallDEVS [SmallDEVS , 2003] est un environnement expérimental de simulation basé sur le formalisme DEVS développé par Vladimir Janousek et ElodcKironsky à l'université de Brno de Technologie (République de Tchèque). Il est orienté objet basé sur les prototypes et tient compte de l'aspect classe d'objet. Cet outil permet la modélisation et la simulation interactive. Il supporte aussi bien la multi simulation et simulation réflective.

3.10 Conclusion

Nous avons présenté dans ce chapitre le formalisme DEVS développé par B.P. Zeigler ainsi que des extensions DEVS qui permettent de prendre en compte parallélisme, imprécision et incertitude sur les événements ou sur les états d'une part. Où si l'incertitude intervient au niveau des changements d'états, alors que l'imprécision intervient sur le temps ou les variables des événements. Dans les deux cas, ces changements modifient les algorithmes de simulation et la dynamique de structures des modèles en cours de simulation d'autre part, ensuite nous avons brossé une vue d'ensemble de plateformes de ces formalismes.

CHAPITRE 4

APPROCHE PROPOSÉE POUR LA TRANSFORMATION DEVS/AGR

4.1 Introduction

Nous présentons, dans ce chapitre, l'approche Ingénierie Dirigée par les Modèle (IDM) et plus particulièrement l'Architecture Dirigée par les Modèles (ADM). Puis nous traitons d'un ensemble de travaux proposant des transformations entre DEVS et les SMA, qui sont nos sources d'inspiration. L'approche et les algorithmes de transformation proposés sont présentés dans la dernière section de ce chapitre.

4.2 Définition de transformation de modèles

Hubert Kadima [Hubert, 2005] donne les définitions suivantes pour le concept de transformations de modèles :

Définition 4.1. *Une transformation de modèles est la génération d'un ou de plusieurs modèles cibles à partir d'un ou de plusieurs modèles sources conformément à une définition de transformation.*

Définition 4.2. *Une définition de transformation est un ensemble de règles de transformation qui décrivent globalement comment un modèle décrit dans un langage source peut être transformé en un modèle décrit dans un langage cible.*

Définition 4.3. *Une règle de transformation est une description de la manière dont une ou plusieurs constructions dans un modèle source peuvent être transformées en une ou plusieurs constructions dans un modèle cible.*

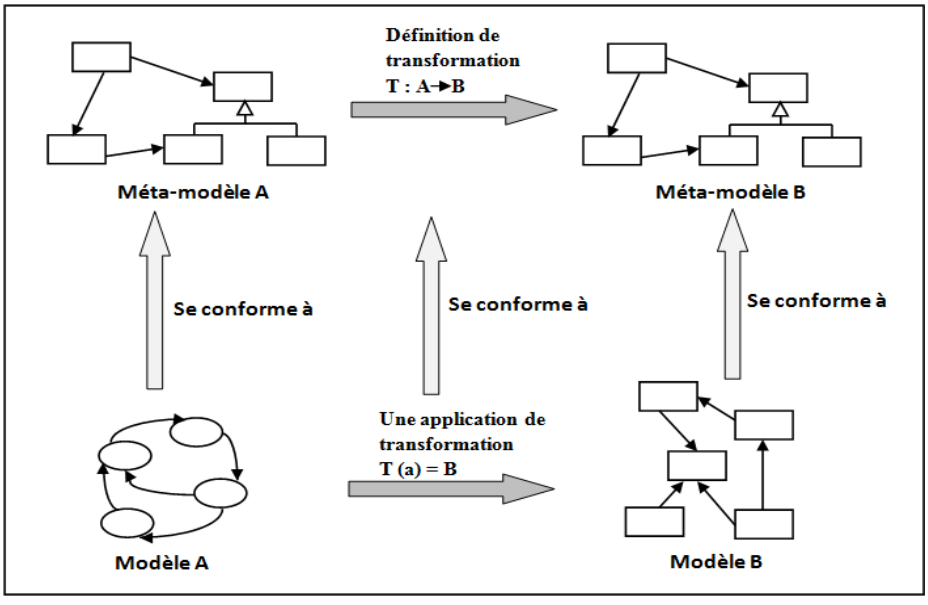

FIG. 4.1 – Transformation de modèles

Un large consensus existe autour de ces définitions. En règle générale, pour réaliser une transformation entre modèles, ces derniers (source et cible) doivent être représentés en adoptant un langage de modélisation adéquat (par exemple : UML pour des modèles de conception, Java pour des modèles de code source). Ce langage de modélisation étant lui-même défini par un méta-modèle. Les règles de transformations sont basées sur les langages de transformations, tels que le standard QVT, ATL, Kermeta,… Le moteur de transformation traite en entrée le modèle source, exécute les règles de transformation, et produit en sortie le modèle cible [Jouault, 2006 ; OMG, 2008].

4.3 Ingénierie Dirigée par les Modèles (IDM)

L'ingénierie dirigée par les modèles (IDM) est une voie vers laquelle s'oriente l'ingénierie du logiciel (IDM) en adoptant le principe du « tout est modèle » [Combemale *et al.,* 2008]. Cette approche peut être considérée à la fois en *continuité* et en *rupture* avec les principales tendances et travaux précédents [Bézivin, 2004, 2005]. Ainsi l'IDM offre un cadre où l'application est produite à partir de modèles. L'idée de l'IDM est d'exploiter la production des modèles par la machine lors du processus de développement. Il est nécessaire, dans la pratique, d'utiliser des spécifications formelles. Le terme 'transformations de modèles' regroupe les algorithmes et programmes qui permettent de traiter ces modèles.

Différentes approches existent pour manipuler les modèles dans leur processus de développement des systèmes. L'approche dite 'Model Driven Architecture' (MDA) est une approche assez connue et utilisée, c'est l'approche vision IDM de l'Object Management Group (OMG)[1].

L'approche MDA utilise sur trois types de modèles : Le modèle des besoins CIM (*Computation Independent Model*. Le modèle d'analyse et de conception PIM (*Platform Independent Model*) et le modèle de code PSM (Plate- form Specific Model)[Slimane [Hammoudi, 2010].

4.3.1 Concepts généraux

Nous introduisons, dans cette section, quelques concepts de base de l'IDM ainsi que les relations qui existent entre ces concepts. La notion de modèle peut être définie comme suit :

[1]http ://www.omg.org/

> ***Définition 4.4.*** *Un modèle est une image simplifiée d'un système, autrement dit, une abstraction d'un aspect particulier d'un système. L'aspect système capturé dépend de l'objectif du modèle. Bézivin et Gerbé définissent le modèle comme une simplification d'un système élaboré pour répondre à un objectif précis [Bézivin et Gerbé, 2001].*

Afin de rendre un modèle utilisable il est nécessaire de préciser le langage de modélisation dans lequel il est exprimé. On utilise pour cela un méta-modèle qui peut être défini comme suit :

> ***Définition 4.5.*** *Un méta-modèle est un modèle spécifique qui définit la syntaxe abstraite d'un langage de modélisation. Bézivin et Gerbé définissent le méta-modèle comme la spécification explicite d'une abstraction (simplification). Un méta-modèle utilise un langage spécifique pour exprimer cette abstraction [Bézivin et Gerbé, 2001].*

Un méta-modèle permet d'interpréter et de faciliter la compréhension et l'utilisation d'un modèle. De même, pour pouvoir interpréter un méta-modèle il faut disposer d'une description de sa syntaxe pour pouvoir l'instancier et ainsi, on utilise un méta-modèle pour ce méta-modèle. C'est naturellement que l'on désigne ce méta-modèle particulier par le terme de méta-méta-modèle :

> ***Définition 4.6.*** *Un méta-méta modèle est un modèle qui définit la syntaxe d'expression d'un méta-modèle. Un méta-méta modèle doit être auto-descriptif, c'est-à-dire que l'on peut utiliser les concepts qu'il définit pour le décrire et le comprendre.*

4.3.2 L'approche MDA

Le MDA a été proposé par l'OMG (Object Management Group) [OMG, 2004]. C'est à la fois la proposition d'une approche de développement et d'une architecture. Cette approche a pour but principal la séparation des spécifications fonctionnelles d'un système des spécifications de son implémentation. Assurant, ainsi, son interopérabilité des modèles fonctionnels pour différents choix de mise en œuvre. L'approche MDA est entièrement basée sur la transformation de modèles.

4.3.2.1 Architecture à quatre niveaux

L'architecture MDA est distribuée selon 4 niveaux d'abstraction (Figure 4.2) :

- Niveau M0 : Ce niveau concerne la modélisation des données réelles que l'on souhaite modéliser. Ces données présentent une instance du modèle du niveau suivant M1.
- Niveau M1 : Composé de modèles d'informations (PIM, PSM), c'est le niveau du modèle permettant la description des données du niveau M0. Typiquement, un modèle UML appartient à ce niveau. Un modèle appartenant à ce niveau est décrit par un méta-modèle du niveau M2.
- Niveau M2 : Ce niveau est composé de langages de définition des modèles d'informations. Les méta-modèles de description de langages, typiquement, le méta-modèle UML sont décrits à ce niveau.
- Niveau M3 : Ce niveau comprend entité unique : langage unique de définition des méta-modèle appelé le Meta-Object-Facility (MOF) [OMG, 2011a].

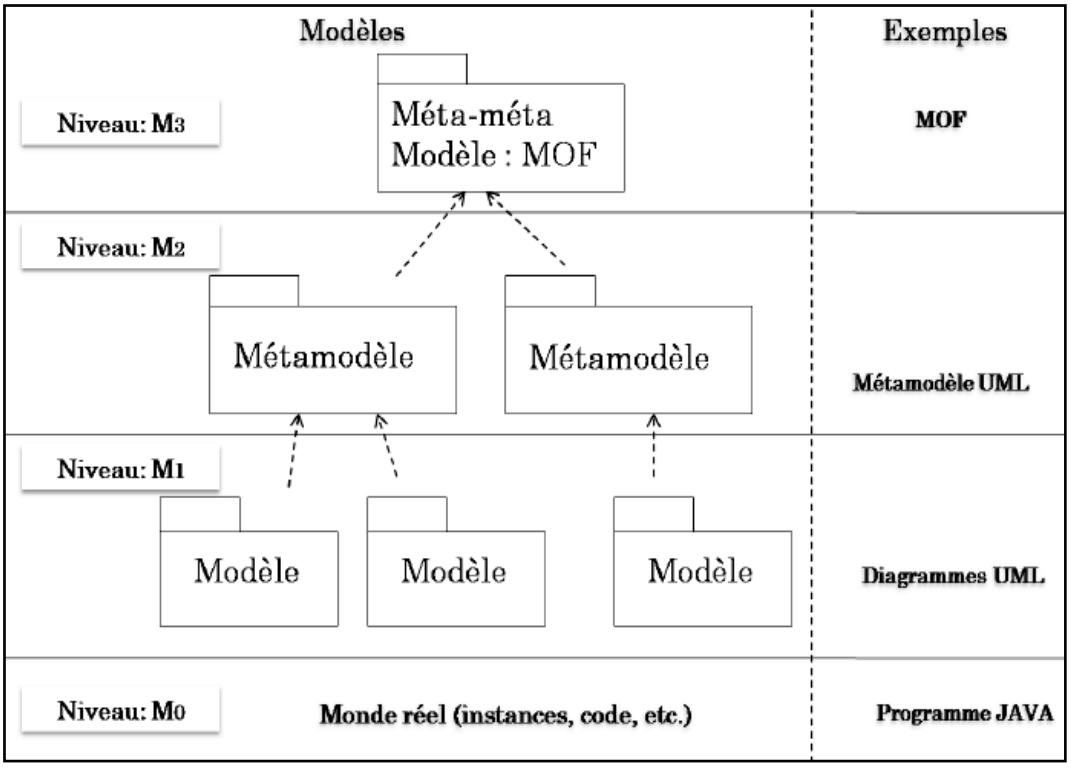

FIG. 4.2 – Architecture de modélisation pour MDA [OMG, 2003]

4.3.2.2 Modèles MDA

L'approche MDA définit trois niveaux de modèles représentant les niveaux d'abstraction de l'application : le CIM, le PIM et le PSM. Ces trois niveaux de modèles sont utilisés pour la génération du code par transformations successives. La figure 4.3 montre les différents types de ces modèles que nous décrivons comme suit :

- CIM (Computational Independant Model) : présente une vue système indépendante de tout calcul. A ce niveau sont décrits les besoins fonctionnels de l'application indépendamment des détails liés à son implémentation. Cette indépendance technique de ce modèle lui permet de garder tout son intérêt au cours du temps. Il n'est modifié que si les connaissances ou les besoins métier changent.

- PIM (Platform-Independent Model) : Ce niveau est, quant à lui, indépendant de toute plateforme technique. Il donne une vision structurelle et dynamique de l'application de façon indépendante de toute conception technique. Ainsi, la prise en compte tardive des détails d'implémentation permet de maximiser la séparation des préoccupations entre la logique de l'application et les techniques d'implémentation. Par ailleurs, ce modèle doit contenir toutes les informations nécessaires pour qu'une génération automatique de code soit envisageable.

- PSM (Platform-Specific Model) : A ce niveau, le modèle est spécifique à une plateforme technique particulière et il permet de faciliter la génération de code. Dans le modèle MDA, l'obtention du code d'une application est obtenue à partir d'un modèle qui comporte en même temps la spécification fonctionnelle et les informations de la plate-forme d'exécution [Atigui, 2013].

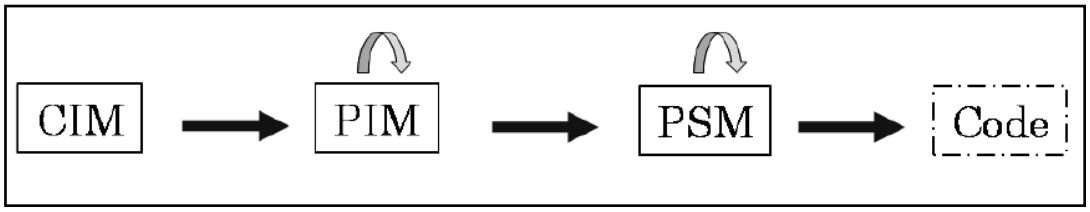

FIG. 4.3 – Les modèles et les transformations dans l'approche MDA [OMG, 2003]

En résumé, on constate que l'implémentation du MDA est entièrement basée sur les modèles et leurs transformations. L'étape la plus critique dans un processus MDA est la transformation d'un PIM en un ou plusieurs PSM, i.e. le passage d'un niveau indépendant de toute implémentation à celui qui utilise une plate-forme réelle [Kleppe *et al.*, 2003].

4.3.2.3 Transformation MDA

La transformation de modèles est « le processus de conversion d'un modèle en un autre modèle du même système » [OMG, 2003]. C'est le centre névralgique de l'ingénierie dirigée par les modèles. Le but principal d'une transformation est d'automatiser les aspects fastidieux du développement logiciel. MDA permet plusieurs possibilités de transformations entre modèles.

Il s'agit, pour le processus de transformation, de générer à partir d'un ou de plusieurs modèles sources d'un système, un ou plusieurs modèles cibles du même système. Les modèles doivent dans tous les cas être conformes à leurs méta-modèles respectifs. Ainsi un moteur de transformations géré ces transformations grâce à un ensemble de règles. Le modèle source est fourni en entrée à ce moteur de transformations qui va génère le modèle cible qui lui est associé.

La figure 4.4 présente un simple scénario d'une transformation avec un modèle en entrée et un modèle en sortie. Les deux modèles sont conformes avec leurs méta-modèles.

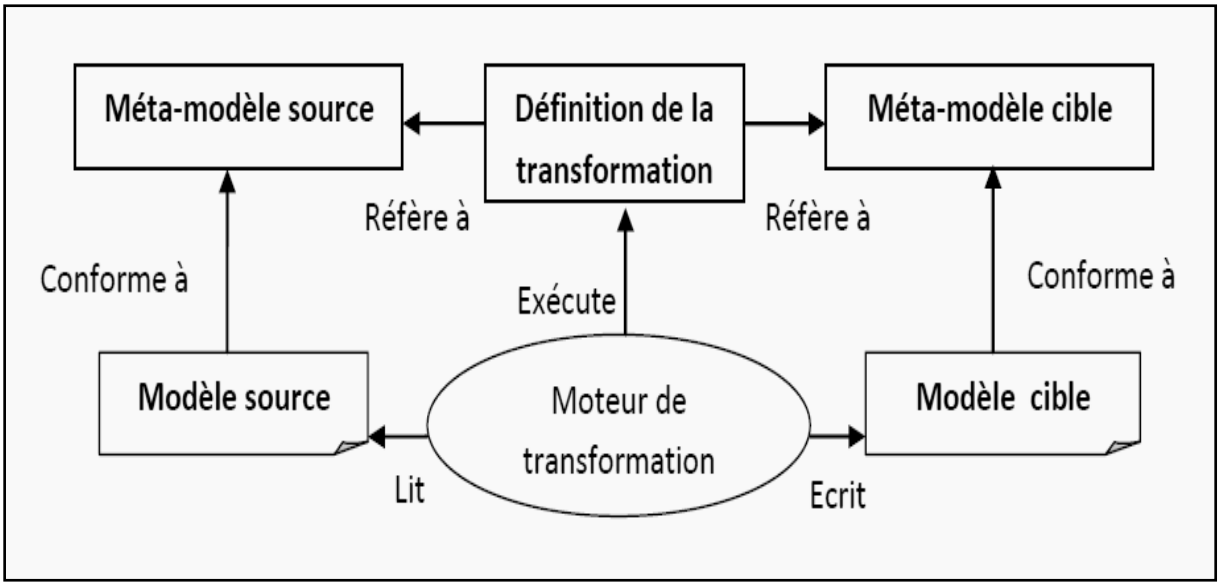

FIG. 4.4 – Scénario d'une transformation de modèle [OMG, 2004]

- **Différentes approches**

Les approches de transformation de modèles peuvent être classées selon plusieurs critères. On distingue deux approches de transformations : les transformations de « modèles vers code» et les transformations de « modèles à modèles » [Czarnecki et Helsen, 2003]. Si on considère le type de la transformation elle-même, on peut distinguer plusieurs sous-catégories comme le montre la figure 4.5.

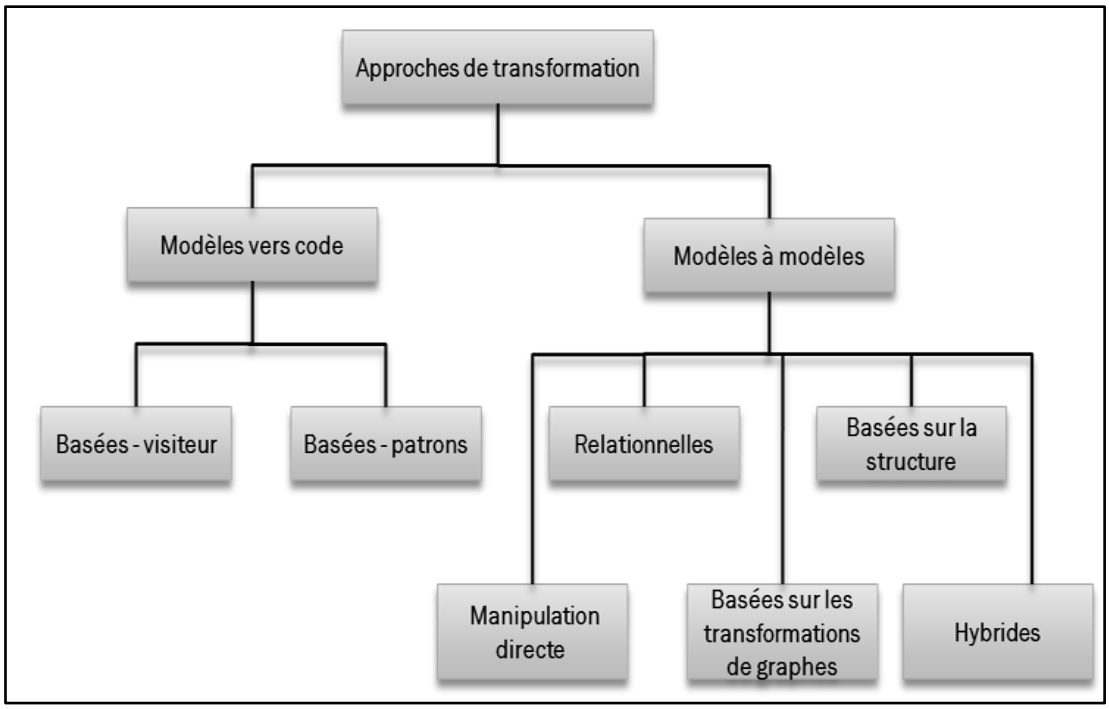

FIG. 4.5 – Approches de transformations de modèles [Czarnecki et Helsen, 2003]

- ✓ **Approches modèles vers code**

Deux approches se distinguent dans cette catégorie :

a- Les approches basées sur le principe du visiteur (*visitor-based*) dans lesquelles l'on se sert du modèle et de ses éléments qui rapprochent sa sémantique du langage de programmation pour obtenir le code Jamda [Jamda, 2004].

b- Les approches basées sur le principe des patrons (*Template-based*) qui utilisent les accès aux informations du modèle source en utilisant les fragments de texte contenant des bouts de méta-code permettant d'effectuer une sélection de code et de réaliser des expansions itératives. Codagen Architect [Codagen, 2004], ModTransf [Dumoulin, 2004], OptimalJ [OptimalJ] et JET [JET, 2004] sont des approches basées sur cette catégorie.

- ✓ **Approches modèles à modèles**

On distingue globalement cinq types d'approches : les approches qui offrent des mécanismes pour manipuler directement les modèles, les approches relationnelles, les approches basées sur les transformations de graphes, les approches dirigées par la structure et les approches hybrides.

- **Approches par manipulation directe :**

  Ces approches reposent sur la représentation interne des modèle source et cible et leurs manipulations par un ensemble d'APIs [Djeddai, 2013]. Elles sont implémentées, généralement, comme des framework orientés objet qui fournissent un ensemble minimale d'infrastructure pour organiser la transformation. Les règles de transformations et leur ordonnancement sont réalisés par les utilisateurs en utilisant un langage de programmation tel que Java.

- **Approches relationnelles**

  Les transformations de modèle, dans ces approches, sont basées sur la logique déclarative et elles sont bidirectionnelles. Le principe de base de ces transformations consiste à établir une relation entre les éléments des modèles. Ces relations seront spécifiées à l'aide de contraintes des liens entre la source et la cible d'une relation. Un exemple de mise en œuvre de ces approches est présenté par AMW [Del Fabro *et al.*, 2005], MTF [Griffin, 2004], KMTF [Akehurst *et al.*, 2005], le plugin Eclipse Tefkat [Lawley et Steel, 2005] et QVT Relations [OMG, 2011b].

- **Approches basées sur les transformations de graphes**

  Dans ce type d'approches, une représentation graphique est associée aux modèles et méta-modèles sources et cibles de la transformation apparentée à un graphe. Ainsi, les techniques utilisées en théorie des graphes (réécriture et transformations de graphes) peuvent être appliquées pour des transformations de modèles ce qui en fournit des modèles formels très appréciables. Un ensemble de règles de réécriture d'un programme de transformation va sélectionner une partie d'un graphe source identifiée par l'intermédiaire d'un langage de navigation. Ensuite, il y'a application d'un filtre sur cette partie sélectionné qui permet de le modifier avant de le recopier dans le graphe cible [Bondé, 2006]. On trouve cette catégorie d'approches dans : AToM3 [Lara et Vangheluwe, 2002], Viatra *(VIsual Automated model TRAnsformations)* [Varró *et al.*, 2002], AGG *(Attributed Graph Grammar)* [Taentzer, 2003], Moflon [Amelunxen *et al.*, 2006], UMLX [Willink, 2003] et BOTL *(Bidirectional Object-oriented Transformation Language)* [Marschall et Braun, 2003] par exemple [Garredu, 2013].

- **Approches basées sur la structure**

  Dans ces approches, Le principe de transformations consiste à construire la structure hiérarchique du modèle cible, ensuite à ajuster les valeurs des attributs et des références. Interactive Objects and Project Technology (IOPT) [IOPT] et OptimalJ [OptimalJ] sont des approches basées sur cette catégorie.

- **Approches hybrides**

  Les approches hybrides sont une combinaison des différentes techniques. La combinaison d'approches impératives et déclaratives. Donc, la stratégie d'application de règles de transformation peut être soit gérée par l'environnement de transformation soit gérée par l'utilisateur. Comme exemples d'approches de cette catégorie, on peut citer : ModTransf [Dumoulin, 2004], Kermeta [Kermeta], ATL [Jouault et Kurtev, 2006 ; OMG, 2011b].

4.3.2.4 Outils de transformations MDA

Dans le contexte de l'IDM, il est possible de compter plus d'une centaine d'outils, tant commerciaux qu'en open source disponibles pour réaliser la transformation de modèles. Quatre catégories d'outils peuvent être distinguées :

- **Les outils de transformation génériques :** On trouve notamment, dans cette catégorie les outils de la famille XML, tels que Xquery [Robie, 2007], XSLT [Kay, 2001], et les outils de transformation de graphes comme défini dans [Guerra *et al.*, 2004]. Les premiers sont largement utilisés dans le monde XML, ce qui leur a permis d'atteindre un certain niveau de maitrise. Cependant, l'expérience montre que ce type d'outils est mal adapté pour des transformations de modèles complexes parce qu'ils ne permettent pas de travailler à la sémantique des modèles manipulés mais uniquement à celui d'un arbre couvrant le graphe de la syntaxe abstraite. Ceci impose des contorsions qui rendent rapidement la transformation de modèles complexes à élaborer, à valider et à maintenir.

- **les langages de scripts intégrés dans les ateliers de génie logiciel :** La seconde catégorie, regroupe des outils de transformation de modèles proposés pour la plupart par des vendeurs d'ateliers de génie logiciel. On peut citer, sans être exhaustif, Arcstyler [OMG, 2002] d'Interactive Objects qui propose la notion de MDA-Cartridge. Cet outil encapsule une transformation de modèles écrite en JPython. Ces MDA-Cartridges peuvent être combinés avant leur exécution par un interpréteur dédié. L'on dispose aussi, dans cet outil d'une interface graphique utile pour la définition de nouvelles MDA-Cartridges. Objecteering [Softeam, 2009] d'Objecteering Software (filiale de Softeam) est un autre outil de cette catégorie. Cet outil propose un langage de script pour la transformation de modèles appelé J. Il en va de même pour le langage OptimalJ de Compuware qui utilise le langage TPL, et pour le monde de l'open source, on trouve d'autres outils comme Fujaba (From UML to Java and Back Again) [Geiger *et al.,* 2006].

  Parmi les intérêts de cette catégorie, on cite leur relative maturité, d'une part, et leur excellente intégration dans les ateliers de génie logiciel qui les héberge, d'autres part. Leur principal inconvénient vient de cette intégration poussée : Généralement, il s'agit de langages de transformation propriétaires sur lesquels il peut être risqué de miser sur le long terme. En plus, ces langages ont été conçus comme des ajouts, au début marginaux au génie logiciel. Même s'ils ont maintenant pris de l'importance, ils ne sont souvent pas vus comme les outils centraux qu'ils devraient être dans une véritable ingénierie d'IDM. Un autre point à signaler réside dans le fait que ces langages montrent une certaine lourdeur lorsque lors des transformations de modèles complexes quand il s'agit de les gérer, les faire évoluer et les maintenir sur de longues périodes [Giese *et al.,* 2009].

- **les outils spécifiques aux transformations de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standards :** On trouve dans cette catégorie, par exemple Mia-Transformation [Mia, 2009] de Mia-Software. Cet outil exécute des transformations en prenant en charge plusieurs formats d'entrée et de sortie. L'expression des transformations se fait comme des règles d'inférence semi-déclaratives (il n'y a pas de mécanisme de type programmation logique). Pour des services additionnels tels que la manipulation de chaînes, ces règles peuvent être enrichies en utilisant des scripts Java.

PathMATE de Pathfinder solutions [PathfinderSolutions, 2003] est une autre plateforme configurable de transformation de modèles qui s'intègre notamment avec les ateliers de modélisation UML. On peut trouver de nombreux projets open source dans le monde académique qui s'inscrivent dans cette approche : les outils UMT-QVT (*UML Model Transformation Tool*) [UMT-QVT, 2005], ModTransf [Dumoulin, 2004] *(XML and ruled based transformation language)*, BOTL *(Bidirectional Object-oriented Transformation Language)* [Marschall et Braun, 2003], MTL [Silaghi *et al.,* 2005], QVTEclipse [Tratt] *(une implantation préliminaire de quelques unes des idées du standard QVT dans Eclipse)*, Coral [Alanen et Porres, 2004] *(Toolkit to create/edit/transform new models/modeling languages at run-time)*, AndroMDA [Bohlen, 2007] ou encore ATLde l'INRIA [Bézivin *et al.,* 2008]. Ces langages conçus particulièrement pour la transformation de modèles présentent l'avantage d'exprimer très simplement les transformations de modèles simples (par exemple des transcodages syntaxiques). Cependant, on remarque leurs limites pour les transformations de modèles complexes du point de vue algorithmique, ou bien en cas de nombreuses variantes ainsi que lors de leur évolution sur de longues périodes.

- **les outils de méta-modélisation par exécution d'un méta-programme :** Dans cette catégorie, l'outil de méta-modélisation exécute un méta-programme orienté objet. Il est aussi aisé de construire un framework permettant la maintenance et la gestion de plusieurs variantes des transformations nécessaires au contexte des lignes de produits. Meta-Edit+ [Tolvanen *et al.,* 2003] de Meta-Case [Revault *et al.,* 1995] est sans nul doute le plus ancien et le plus connu de ces outils. Il permet de définir explicitement un méta-modèle et de programmer au même niveau tous les outils nécessaires, (éditeurs graphiques, générateurs de code, transformations de modèles,…).

  Dans le même ordre d'idées ; XMF-Mosaic [XMF Mosaic, 2007] de Xactium est un environnement complet de définition de langages. On trouve dans cet environnement un noyau exécutable de définition de langages. Tout est modélisé sur cette plateforme (les outils, les GUI, les parsers et autres analyseurs XML…). Le langage de méta-modélisation est un langage de programmation orienté objet possédant toute la force d'un langage complet ce qui en fait un outil particulièrement robuste pour la transformation de modèles. [Jézéquel, 2006 ; Menet, 2010]

4.4 Outils et approches pour la transformation DEVS/SMA

Dans le cadre de combinaisons DEVS et les SMA, plusieurs approches ont été développées La méthode décrite dans cette thèse s'inspire des approches présentées très brièvement ci-dessous.

Les plateformes JAMES (Java-based Agent Modeling Environment for Simulation) et JAMES II (Java-based multipurpose Environment for Modeling and Simulation) [Uhrmacher et Shattenberg, 1998 ; Himmelspach *et al.*, 2010 ; Himmelspach, 2012] sont des plateformes de simulation dynamique des systèmes multi agents. Un agent est caractérisé par l'ensemble des paramètres d'un modèle atomique DEVS (équation 3.1). La communication de chaque modèle avec l'environnement est déterminée par les évènements en entrée et en sortie. Un groupe d'agents est caractérisé par les paramètres d'un modèle couplé DEVS (équation 3.2).

La notion d'environnement dans lequel sont situés les agents joue un rôle très important dans un système multi-agents. Cet environnement peut être perçu comme étant un cadre partagé dans lequel s'effectuent les interactions entre les agents [Weyns *et al.,* 2005a, b ; Wooldridge, 2009; Steiniger *et al.*, 2012]

La plateforme GALATEA (*GLIDER with Autonomus, Logic-based Agents, Temporal reasoning and Abducation*) [Davilla et Uzcategui, 2000; Davilla *et al.,* 2005] offre un cadre de simulation permettant à l'utilisateur la spécification et la conception de systèmes multi-agents par le biais de concepts de hauts niveaux tels que les buts, croyances et préférences ainsi que la modélisation de rôles, de plans d'actions, l'établissement de communications, de négociations et de dialogues entre les agents ce qui permet l'émergence de solutions distribuées efficaces. La plateforme GALATEA est implémentée en tant que plateforme DEVS pour les simulations multi-agents.

GALATEA est le produit de deux lignes de recherches : les langages de simulation basés sur la théorie de Zeigler et les agents logiques (agents basés sur les connaissances disponibles concernant leur environnement et un raisonnement (logique) portant sur les actions possibles dans cet environnement). Cette plateforme permet la modélisation de plusieurs formalismes et incorpore un ensemble de langages orientés programmation logique conçu spécialement pour la modélisation des agents. L'inconvénient majeur de cette plateforme est sa difficulté de mise en œuvre [Mattei *et al.,* 2012].

La plateforme VLE (Virtual Laboratory Environment) [Ramat et Preux, 2003 ; Quesnel *et al.*, 2009 ; Mattei *et al.*, 2012] offre un cadre très développé de simulation basée DEVS. Cette plateforme permet de simuler les systèmes multi agents par le formalisme DEVS. Un agent est représenté, dans ce framework, par un modèle atomique DEVS et un stimulus externe par un message. L'environnement est représenté par CELL-DEVS [Ilachinski, 2001 ; Wainer et Giambiasi, 2001].

La plateforme DEVSimPy [Mattei *et al.*, 2012] prend en considération les points forts et les points faibles des deux dernières méthodes présentées ci-dessus et propose un cadre de simulation d'agents par des modèles DEVS. Ainsi, un agent et son environnement sont représentés par des modèles atomiques DEVS. Chaque agent est représenté par un modèle atomique DEVS. De même, un environnement est représenté lui-aussi par un modèle atomique DEVS. A un groupe d'agent est associé un modèle DEVS. L'intérêt de cette méthode réside, selon ses auteurs, surtout dans sa rapidité et dans sa flexibilité.

C'est après avoir étudié à fond toutes les approches énumérées ci-dessus, que nous proposons une nouvelle approche qui respecte les contraintes suivantes :

- Spécifique au domaine d'application : Modélisation des systèmes industriels.
- Simple et rapide à mettre en œuvre.
- Souple et extensible.
- Très fiable : La fiabilité revêt une importance capitale dans le domaine industriel.
- Exécution des simulations rapide.
- Assurant des interfaces homme-machine efficaces et adaptées au domaine d'application (pouvant représenter graphiquement les évolutions des procédés en temps réel par l'intermédiaire de différentes formes d'affichage (schémas synoptiques, courbes, …)).

L'approche mise au point et développée ici repose sur les principes suivants :

- Découpage du système en sous-systèmes.
- Utilisation du formalisme DEVS pour la détermination des modèles atomiques et couplés en fonction des niveaux d'abstractions des composants obtenus.
- Vérification et validation théorique de ces modèles (conformité avec les équations (3.1) et (3.2) des modèles atomiques et couplés DEVS).

- Implémentation du système grâce à la plateforme multi-agent MadKit en utilisant le modèle AGR. Ceci revient à écrire une extension de MadKit pour assurer les transformations nécessaires entre les modèles DEVS et MadKit d'une part, pour la modélisation des interactions entre ces modèles d'autre part et aussi assurer la coordination entre les différents constituants du système en dernier lieu.
- L'utilisation du modèle AGR est justifiée par sa facilité de mise en œuvre et son adaptation au domaine spécifié. MadKit est une plateforme multi-agent écrite en JAVA qui offre des outils hautement performants pour la gestion des agents ainsi que des interfaces simples à programmer, souples et d'une grande finesse.

4.5 Approche proposée

Dans notre approche, au lieu de représenter les composants du système multi-agent (agents, groups d'agents, environnement) par l'utilisation de modèles atomiques et couples DEVS, nous procédons de façon différente. Initialement, nous identifions de façon méthodologique et approfondie tous les sous-systèmes qui composent le système à modéliser. Chaque composant de base est ainsi modélisé par un modèle atomique DEVS. A chaque groupe (compose lui-même par un ensemble de modèles atomiques) on fait correspondre le modèle couple DEVS adéquat. Ces modèles obtenus sont vérifiés et validés séparément grâce au formalisme DEVS.

Dans un deuxième temps, ces modèles sont implémentés par l'utilisation d'un système multi-agents : les modèles atomiques sont transformés en agents, les modèles couplés en groupes d'agents et les environnements correspondent aux messages transmis entre agents. La puissance et les outils des plateformes multi-agents permettent une grande rigueur et beaucoup de souplesse dans l'implémentation de ces systèmes.

Après une étude approfondie de différentes plateformes pour l'implémentation de notre système, nous avons opté pour le système MADKIT qui est basé sur le modèle organisationnel AALADIN et qui repose sur le concept d'Agent, Groupe et Rôle (AGR) pour son adaptation avec notre approche.

4.5.1 Transformation DEVS/AGR

DEVS est un formalisme qui permet de définir mathématiquement un modèle. Autrement dit, DEVS détermine les structures et les évolutions dynamiques (comportement) des entités qui composent le système à l'aide de deux modèles atomiques et couplés. La représentation

DEVS encapsule la structure et le comportement du modèle. Ainsi, les modèles DEVS fournissent des représentations mathématiques qui peuvent être formellement vérifiées et validées.

Les approches SMA présentent des outils théoriques et pratiques solides dans le domaine de la modélisation. Un système multi-agents comprend des outils pour gérer les agents (création, destruction, interactions entre agents ...). A ce niveau, l'approche de conception globale et l'implémentation du système peuvent être facilement réalisées.

L'idée développée ici est basée principalement sur la transformation d'un modèle vérifié et validé en DEVS vers un modèle agent pour qu'il soit simulable dans des plateformes multi-agents basées sur le modèle AGR (Agent/Groupe/Rôle).

L'algorithme (CDEVS_AGR) suivant détaille le passage d'un modèle DEVS couplé (CDEVS) au modèle AGR.

4.5.2 Algorithme de transformation

L'algorithme présenté ci-dessous (Algorithme 4.1) permet de réaliser le passage du formalisme DEVS vers le modèle AGR. En premier lieu, la fonction CDEVS_G (CDEVS*i*) crée des groupes G*i*, puis les procédures CDEVS_EIC, CDEVS_IC et CDEVS_EOC définissent les différentes interconnexions entre les agents (AGA*j* *«* Agent correspondant au modèle atomique ADEVS*j »* créé par la fonction ADEVS_AGA (ADEVS*j*) , AGC*i*_R « Agent représentant les ports d'entrée du modèle couplé CDEVS*i »* , AGC*i*_E « Agent représentant les ports de sortie du modèle couplé CDEVS*i »*) et les groupes G*i* selon le type de couplage du modèle CDEVS*i* comme montré ci-dessous.

La fonction ADEVS_AGA (ADEVS*j*) : Elle permet la création des agents AGA*j* pour chaque modèle atomique ADEVS*j*. Ainsi ; les récepteurs et effecteurs de l'agent AGA*j* représentent respectivement les ports d'entrée (ADEVS*j*.Inports*i*) et de sortie (ADEVS*j*.Outports*i*) du modèle atomique. La fonction de perception de l'agent AGA*j*.Pa correspond à la fonction de transition externe ADEVS*j*.δext. La fonction de comportement de l'agent AGA*j*.Fa représente la fonction de transition interne ADEVS*j*.δint. La fonction d'action de l'agent AGA*j*.Infla correspond à la fonction de sortie ADEVS*j*.λ. L'évolution dynamique d'un agent représente la fonction d'avancement du temps ADEVS*j*.ta et l'état interne d'un agent AGA*j*. Cela correspond à l'ensemble des états ADEVS*j*.S.

La fonction CDEVS_AGC (CDEVS*j*) : Elle crée deux agents coordinateurs AGC pour chaque modèle couplé CDEVS*i*, le premier est un agent coordinateur récepteur AGC*i*_R et le deuxième est un agent coordinateur émetteur AGC*i*_E. L'objectif principal de ces deux coordinateurs est de bien conserver l'encapsulation du formalisme DEVS. Le modèle est représenté comme étant une boite noire qui n'agit avec l'environnement que via les ports d'entrée et de sortie. Cependant les agents ont une liberté d'interaction avec leur environnement. Dans cette approche ; les agents voulant communiquer avec des agents internes d'un groupe correspondant à un modèle DEVS couplé doivent impérativement passer par les agents coordinateurs de ce groupe.

La fonction CDEVS_G (CDEVS*i*) : Elle crée des groupes G*i* pour chaque modèle couplé CDEVS*i*, puis elle crée des agents AGA*j* et elle va les regrouper dans ce groupe G*i*. Ensuite elle crée les deux agents de coordination AGC*i*_R et AGC*i*_E grâce à la fonction CDEVS_AGC. Les figures 4.6, 4.7, 4.8, 4.9, 4.10 et 4.11 illustrent les transformations entre les composants DEVS et le modèle AGR

Algorithme 4.1 : Transformation CDEVS_AGR

Input CDEVS
Output AGR
1: **Pour** tous les modèles couplés CDEVS*i* **faire**
2 : Appeler CDEVS_G (CDEVS*i*) // Création des groupes et des agents
3 : Appeler CDEVS_EIC (CDEVS*i*) // Définition des interconnexions correspondant à //CDEVS*i*.EIC
4 : Appeler CDEVS_IC (CDEVS*i*) // Définition des interconnexions correspondant à //CDEVS*i*.IC
5 : Appeler CDEVS_EOC (CDEVS*i*) // Définition des interconnexions correspondant à //CDEVS*i*.EOC
6 : **Fin pour**

Fonction 4.1 : ADEVS _AGA (ADEVS : DEVS Atomique) : AGA

1 : Créer AGA*j* // Modèle agent correspondant à ADEVS*j*
2 : Créer une boite aux lettres Br de taille = ADEVS*j*.Inports.Taille
3 : Créer une boite d'envoi Be de taille = ADEVS*j*.Outports.Taille
4 : **Si** un événement externe est aperçu dans un port : ADEVS*j*.Inports **alors**

5 : Exécuter AGA*j*.Pa // Correspondant à ADEVS*j*.δext
6 : **Fin si**
7 : **Si** la durée de vie t_a de l'état S de Pa est écoulé **alors**
8 : Exécuter AGA*j*.Fa // Correspondant à ADEVS*j*.δint
9 : Exécuter AGA*j*.Infla // Correspondant à ADEVS*j*.λ
10 : **Fin si**

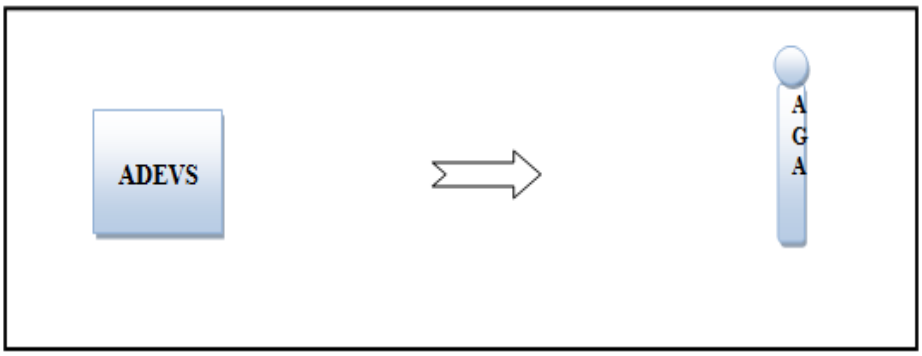

FIG. 4.6 – Représentation graphique de transformation ADEVS_AGA

Fonction 4.2 : CDEVS_G (CDEVS : DEVS Couplé) : G

1 : Créer un groupe G*i* // Correspondant à CDEVS*i*
2 : **Pour** tous les ADEVS*j* **faire**
3 : ADEVS_AGA (ADEVS*j*) // Création des agents AGA*j*
4 : Ajouter AGA*j* dans G*i* // Regroupement des agents
5 : **Fin pour**
6 : CDEVS_AGC (CDEVS*i*)
7 : Ajouter AGC*i*_E et AGC*i*_R dans G*i*

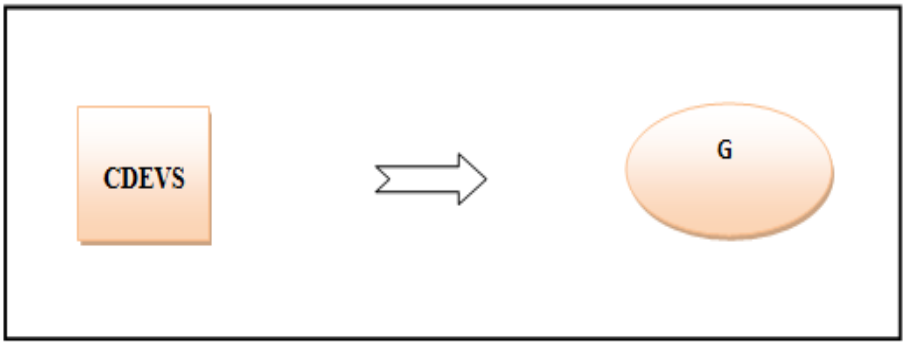

FIG. 4.7 – Représentation graphique de la transformation CDEVS_G

Fonction 4.3 : CDEVS_AGC (CDEVS : DEVS Couplé) : AGC

```
1 :  Créer AGCi_R // Agent récepteur pour CDEVSi.Inports
2 :  Créer une boite aux lettres Br de taille = CDEVSi.Inports.Taille
3 :  Créer une boite d'envoi Be de taille = CDEVSi.EIC.Taille
4 :     Si un événement externe est aperçu dans un port : CDEVSi.Inports alors
5 :        Exécuter AGCi_R.Pa
6 :     Fin si
7 :     Si la boite aux lettres Br non vide alors
8 :        Exécuter AGCi_R.Fa
9 :     Fin si
10 :    Si la boite d'envoi Be non vide alors
11 :       Exécuter AGCi_R.Infla
12 :    Fin si
13 :  Créer AGCi_E // Agent émetteur pour CDEVSi.Outports
14 :  Créer une boite aux lettres Br de taille = CDEVSi.EOC.Taille
15 :  Créer une boite d'envoi Be de taille = CDEVSi.Outports.Taille
16 :    Si un événement externe est aperçu dans un port : CDEVSi.Outports alors
17 :       Exécuter AGCi_E.Pa
18 :    Fin si
19 :    Si la boite aux lettres Br non vide alors
20 :       Exécuter AGCi_E.Fa
21 :    Fin si
22 :    Si la boite d'envoi Be non vide alors
23 :       Exécuter AGCi_E.Infla
24 :    Fin si
```

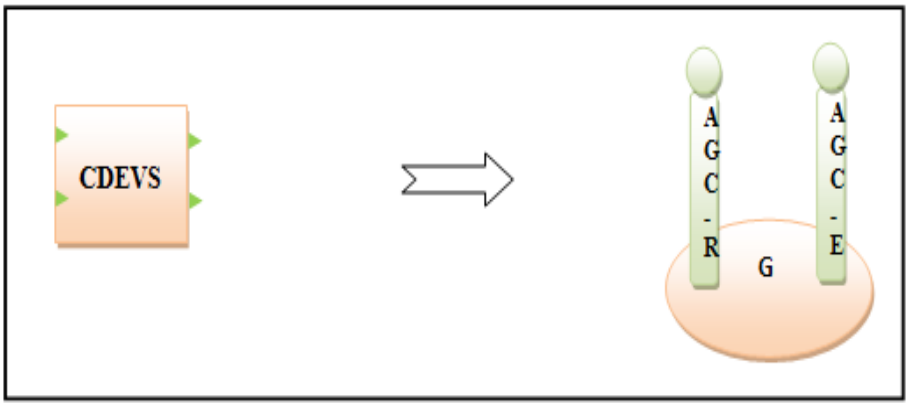

FIG. 4.8 – Représentation graphique de transformation CDEVS_AGC

Procédure 4.1 : CDEVS_EIC(CDEVS : DEVS Couplé)

```
1 : Pour m = 1 jusqu'au nombre des ports d'entrée du CDEVSi faire
2 :       Pour j = 1 jusqu'au nombre des modèles internes M faire
3 :             Pour k = 1 jusqu'au nombre des ports d'entrée du Mj faire
4 :                   Si (CDEVSi.Mj = ADEVSj) alors
5 :                     Si (CDEVSi.Xm = ADEVSj.Xk) alors // EIC entre modèle couplé
                            // et modèle atomique
6 :                         Définir l'ensemble des messages entre Gi.AGCi_R et Gi.AGAj
7 :                     Fin si
8 :                   Fin si
9 :                   Si (CDEVSi.Mj = CDEVSj) alors
10 :                    Si (CDEVSi.Xm = CDEVSj.Xk) alors // EIC entre deux modèles couplés
11 :                        Définir l'ensemble des messages entre Gi.AGCi_R et Gj.AGCj_R
12 :                    Fin si
13 :                  Fin si
14 :            Fin pour
15 :        Fin pour
16 : Fin pour
```

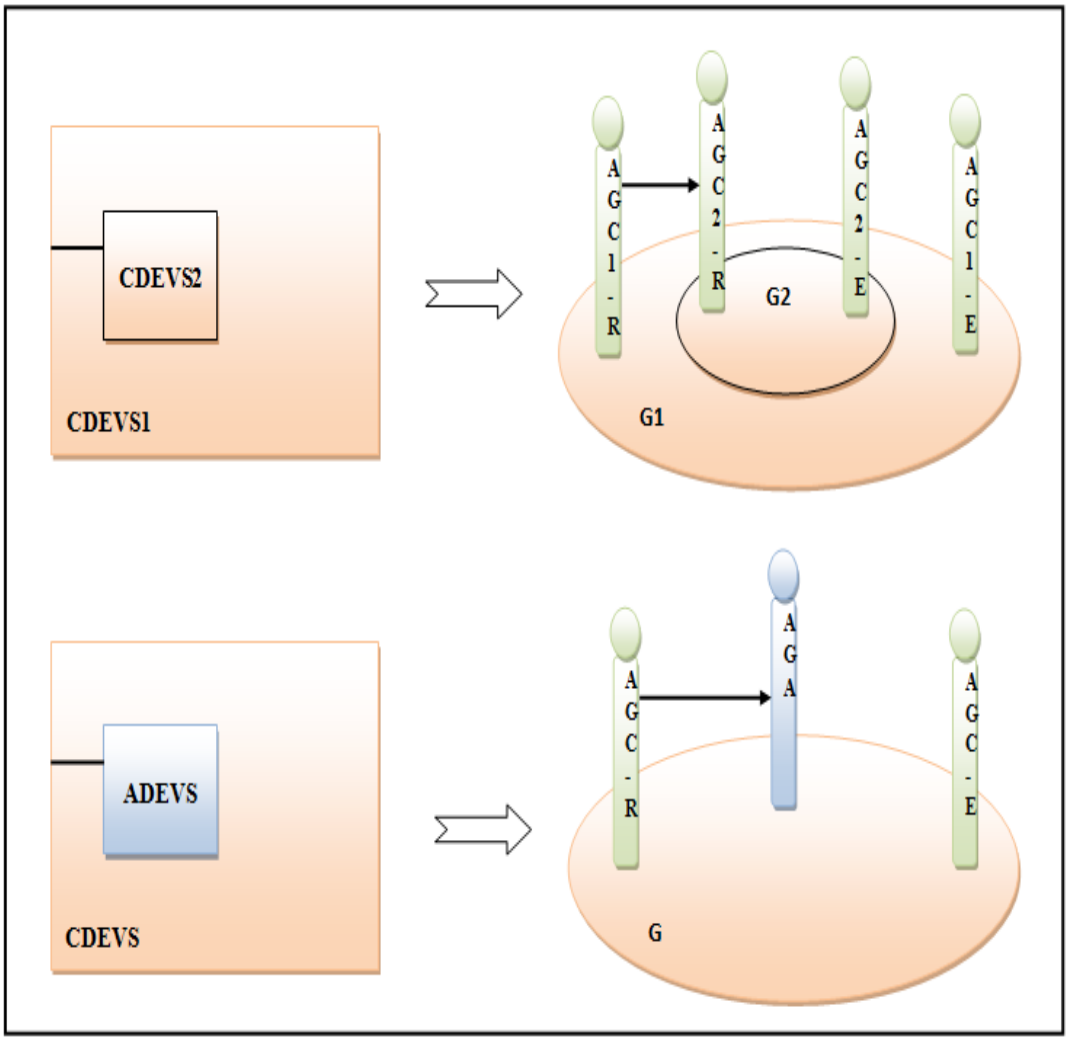

FIG. 4.9 – Représentation graphique de transformation CDEVS_EIC

Procédure 4.2 : CDEVS_IC(CDEVS : DEVS Couplé)

1 : **Pour** *j* = 1 jusqu'au nombre des modèles internes M **faire**
2 : **Pour** *k* = 1 jusqu'au nombre des modèles internes M **faire**
3 : **Si** *j* ≠ *k* **alors** // un port de sortie ne doit pas être couplé avec un port d'entré du même // modèle.
4 : **Pour** *n*= 1 jusqu'au nombre des ports de sortie du M*j* **faire**
5 : **Pour** *m*= 1 jusqu'au nombre des ports d'entrée du M*k* **faire**
6 : **Si** (CDEVS*i*.M*j* = CDEVS*j*) et (CDEVS*i*.Mk = CDEVS*k*) **alors**
7 : **Si** (CDEVS*j*.Y*n*= CDEVS*k*.X*m*) **alors** //*IC* entre deux modèles couplés
8 : Définir l'ensemble des messages entre G*j*.AGC*j*_E et G*k*.AGC*k*_R
9 : **Fin si**
10 : **Fin si**

```
11 :          Si (CDEVSi.Mj = ADEVSj) et (CDEVSi.Mk = ADEVSk) alors
12 :            Si (ADEVSj.Yn = ADEVSk.Xm) alors // IC entre deux modèles atomique
13 :              Définir l'ensemble des messages entre Gi.AGAj et Gi.AGAk
14 :            Fin si
15 :          Fin si
16 :          Si (CDEVSi.Mj = ADEVSj) et (CDEVSi.Mk = CDEVSk) alors
17 :            Si (ADEVSj.Yn = CDEVSk.Xm) alors // IC entre un modèle atomique et un
                  //modèle couplé
18 :              Définir l'ensemble des messages entre Gi.AGAj et Gk.AGCk_R
19 :            Fin si
20 :          Fin si
21 :          Si (CDEVSi.Mj = CDEVSj) et (CDEVSi.Mk = ADEVSk) alors
22 :            Si (CDEVSj.Yn = ADEVSk.Xm) alors // IC entre un modèle couplé
                  //et un modèle atomique
23 :              Définir l'ensemble des messages entre Gj.AGCj_E et Gi.AGAk
24 :            Fin si
25 :          Fin si
26 :        Fin pour
27 :      Fin pour
28 :    Fin si
29 :  Fin pour
30 : Fin pour
```

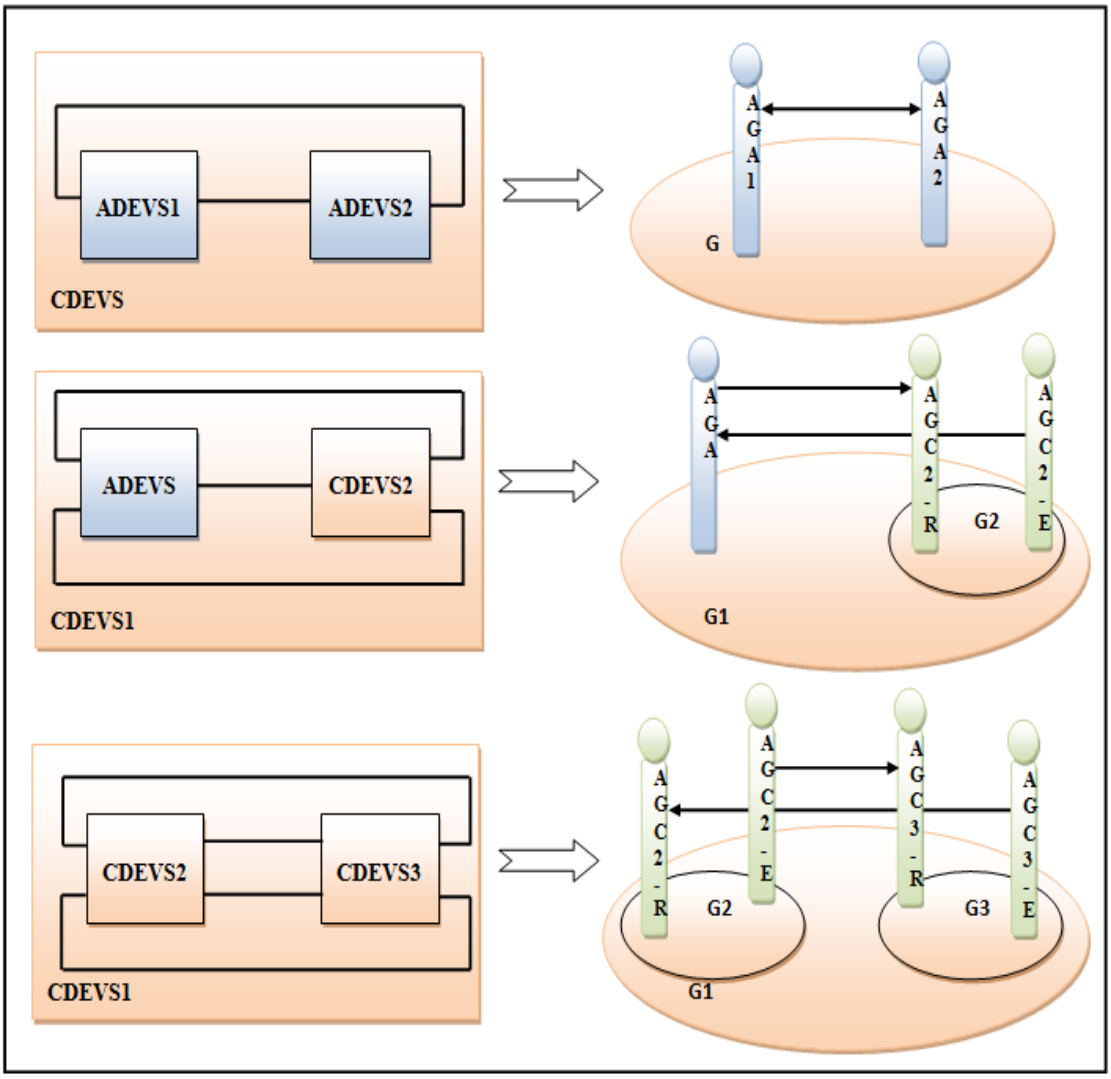

FIG. 4.10 – Représentation graphique de transformation CDEVS_IC

Procédure 4.3 : CDEVS_EOC (CDEVS : DEVS Couplé)

```
1 : Pour j = 1 jusqu'au nombre des modèles internes M faire
2 :       Pour p = 1 jusqu'au nombre des ports de sortie du Mj faire
3 :           Pour q = 1 jusqu'au nombre des ports de sortie du CDEVSi faire
4 :               Si (CDEVSi.Mj = ADEVSj) alors
5 :                   Si (ADEVSj.Yp = CDEVSi.Yq) alors // EOC entre modèle atomique
                        //et modèle couplé
6 :                       Définir l'ensemble des messages entre Gi.AGAj et Gi.AGCi_E
7 :                   Fin si
8 :               Fin si
9 :               Si (CDEVSi.Mj = CDEVSj) alors
10 :                  Si (CDEVSj.Yp = CDEVSi.Yq) alors // EOC entre deux modèles
```

```
                    // couplés
11 :                    Définir l'ensemble des messages entre G j.AGC j_E et G i.AGC i_E
12 :                Fin si
13 :            Fin si
14 :        Fin pour
15 :    Fin pour
16 : Fin pour
```

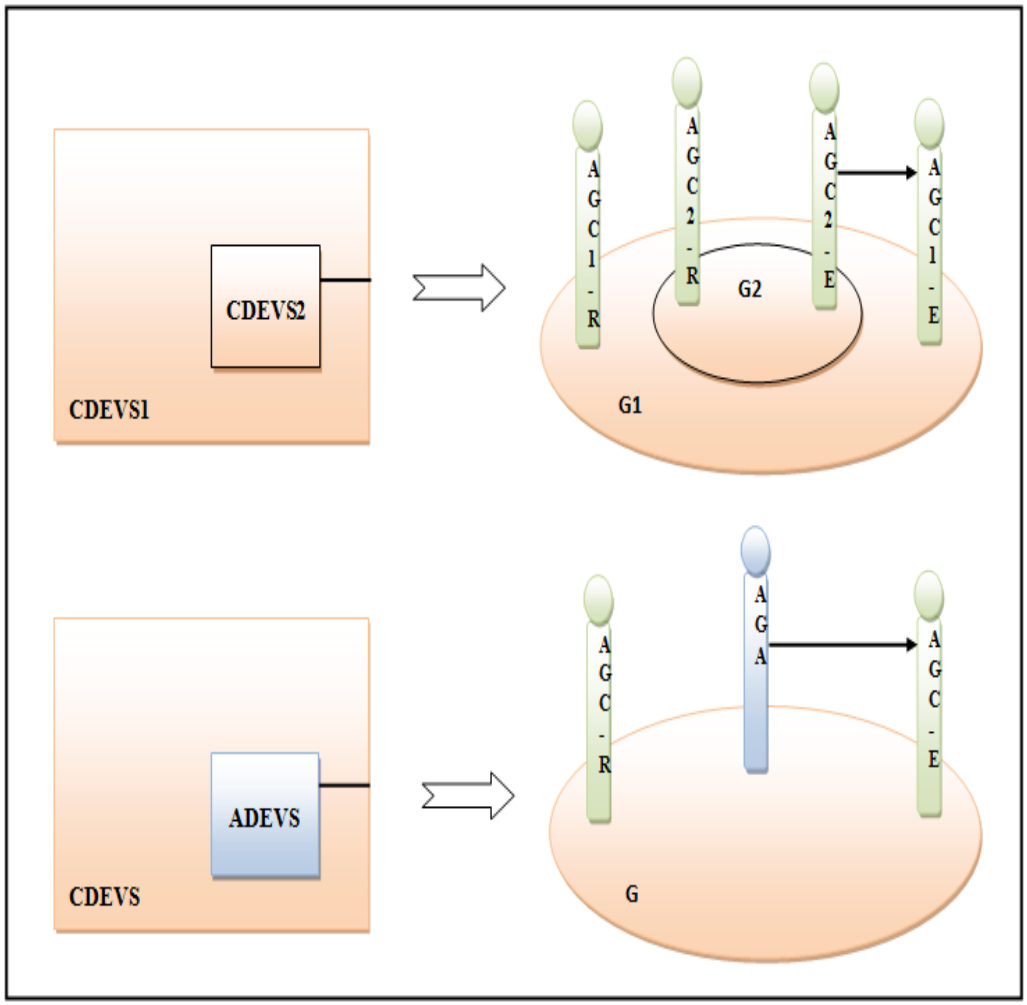

FIG. 4.11 – Représentation graphique de transformation CDEVS_EOC

Nous obtenons table 4.1 ci-dessous un ensemble de règles de passage entre DEVS et le modèle AGR conformément aux équations concernant DEVS et les équations concernant le modèle AGR ainsi que les algorithmes vus ci-dessus.

Modèle DEVS		***Modèle AGR***	
Couplé	*X*	***Groupe***	*AGC_R*
	Y		*AGC_E*
	M		*Groupe / M ∈ CDEVS* *Agent / M ∈ ADEVS*
	EIC		*Interconnexion entre AGCi_R & AGCj_R / Mj∈ CDEVS, (i,j)∈ N || AGCi_R & AGAj / Mj∈ ADEVS, (i,j)∈ N*
	EOC		*Interconnexion entre AGCj_E & AGCi_E / Mj∈ CDEVS, (i,j)∈N || AGAj & AGCi_E / Mj∈ ADEVS, (i,j)∈N*
Atomique	*X*	***Agent$_a$***	Percept$_a$
	Y		*Effecteur$_a$*
	δ_{ext}		P_a
	δ_{int}		F_a
	λ		*Infl$_a$*
	S		*Comportement de l'agent (Rôle)*
	t_a		*En fonction de l'état de l'agent*

TAB. 4.1 – Passage du formalisme DEVS vers le modèle AGR

Les interactions entre agents sont modélisées par le passage de messages entre ces agents. La figure 4.12 illustre de façon globale ces transformations.

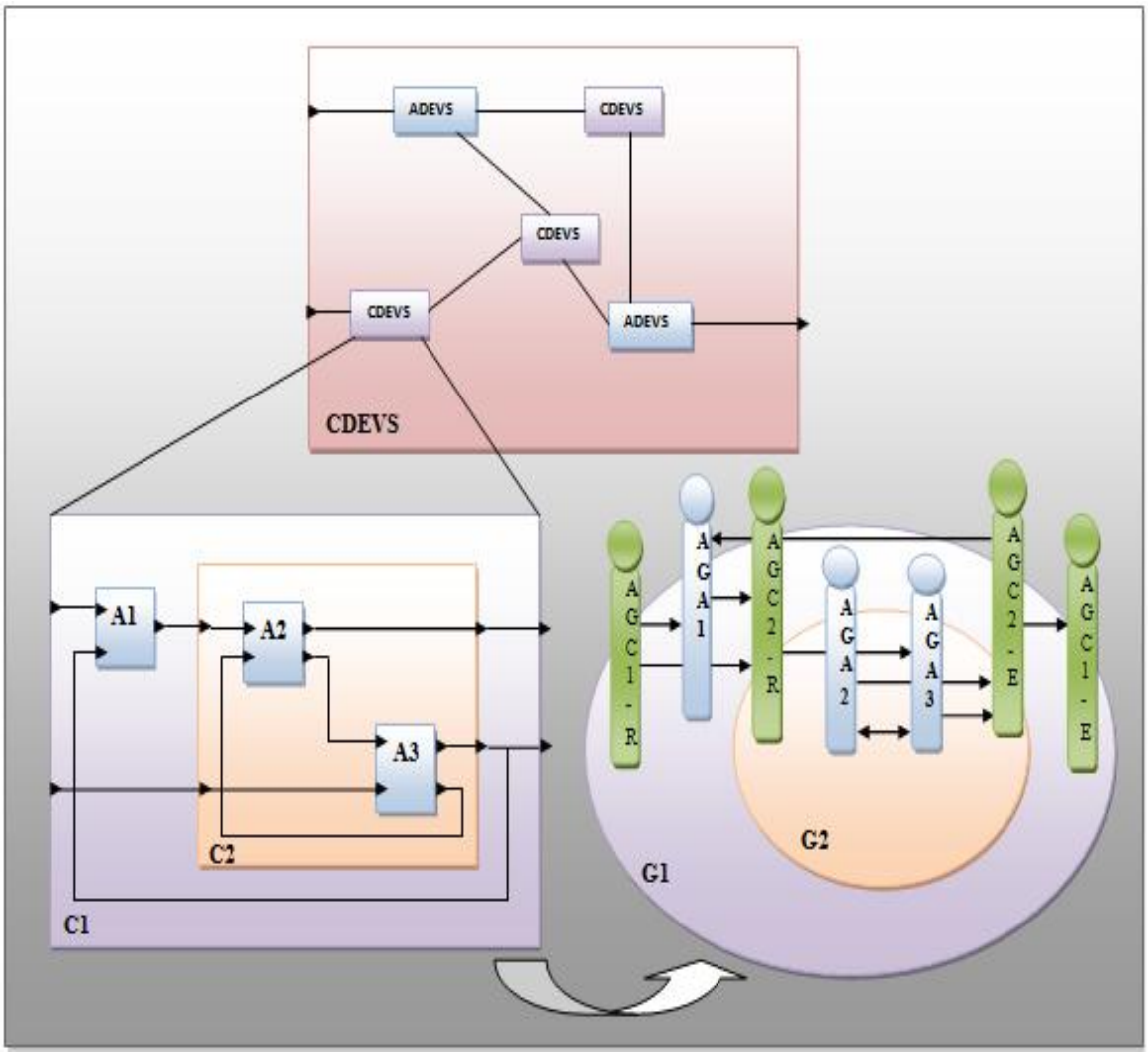

FIG. 4.12 – Modèle de transformation DEVS/AGR

4.6 Conclusion

Dans ce chapitre, nous avons proposé une approche basée DEVS/SMA pour la modélisation et la simulation des systèmes complexes. Pour cela, nous avons proposé une transformation de modèle DEVS vers un modèle AGR. Cette démarche est caractérisée par un ensemble de procédures et fonctions permettant la transformation systématique de ces modèles. L'avantage de notre approche est de profiter de la puissance formelle du formalisme DEVS pour la vérification et la validation d'une part et la puissance des outils évolués offerts par les plateformes SMA pour la mise en œuvre, le développement et l'implémentation d'autre part.

CHAPITRE 5

CAS D'APPLICATION

5.1 Introduction

Dans ce chapitre, nous allons appliquer notre approche présentée dans le chapitre précédent sur un exemple d'un système industriel complexe en décrivant les principaux constituants, le fonctionnement et la régulation du système à simuler. Pour la simulation de ce système nous avons proposé premièrement un modèle DEVS pour représenter et modéliser les différents composants et leurs régulations. Dans un deuxième temps, l'implémentation a été faite grâce un système multi-agents en utilisant notre approche montrée précédemment.

5.2 Description globale de système à simuler

Pour appliquer notre approche nous avons choisi, comme cas d'application, un système industriel qui concerne la régulation DCS (Distributed Control System) d'une chaudière utilisée dans le domaine du pétrole [Seddari et Redjimi, 2013 ; Seddari *et al.*, 2012, 2013, 2014].

Deux éléments majeurs composent le processus : l'eau d'alimentation qui fournit des quantités appropriées d'eau en fonction de la demande d'une chaudière qui génère la vapeur nécessaire à d'autres processus (turbo- compresseurs, turbines ...). Les figures 5.1 et 5.2 illustrent ce système.

5.2.1 Le Poste D'eau

Le poste d'eau est un procédé qui alimente le générateur de vapeur par l'eau alimentaire, il est constitué par :

- **Le Condenseur (15E01)**

Il reçoit l'eau d'extraction du condenseur sous vide 15B01 véhiculée par les pompes d'extraction passant à travers un filtre magnétique (auto-nettoyant) et traverse coté faisceau le 15E01.
Son rôle est de condenser VB (Vapeur Basse température) qui a servit au dégazage physique par l'intermédiaire de l'eau d'extraction plus froide, et la réinjection 15B02 se fait par une ligne munie d'une garde d'eau tout en laissant s'échapper les incondensables (Air) à l'atmosphère par une ligne avec vanne d'isolement et un orifice de restriction (15R024).

- **Le Dégazeur (15B02)**

L'eau d'extraction après avoir traversé le coté faisceau 15E01 entre dans le 15B02. C'est dans cet élément qui s'effectue le dégazage physique, il est composé d'un pulvérisateur et d'une arrivé VB (Vapeur Basse température).

L'eau pulvérisée en fines gouttelettes, se libère facilement de l'air avec l'aide de la VB, son 2ième rôle réchauffer l'eau aux environs de 135 C°.

Très souvent, on bouche de diffuseur au démarrage de la chaudière, pour remédier à cette anomalie, on faire arrêter la pompe d'extraction et puis on la redémarre.

- **La Bâche Alimentaire (15B01)**

L'eau dégazée préalablement au 15B02 arrive à la bâche alimentaire 15B01 qui est une sorte de garde liquide à l'aspiration des pompes alimentaires. Elle est munie d'une rampe de réchauffement VM (Vapeur Moyenne Température), elle est aussi équipé de :

- Deux PSV 05 A/B tarée à 4.5 bars.
- Un niveau a glace (alarme niveau haut LAH06, LAL07 niveau bas).
- Une injection d'hydrazine.
- Un trop plein commandé par la 15LCV15 vers le canal de rejet.
- Une purge à l'égout.

La 15 PIC V15 reliée au 15B02 maintient la pression à 2.2 bars au 15B01, c'est une vanne de détente (VM/VB) elle est commandée à partir de salle de contrôle et débit 30 t/h au max.

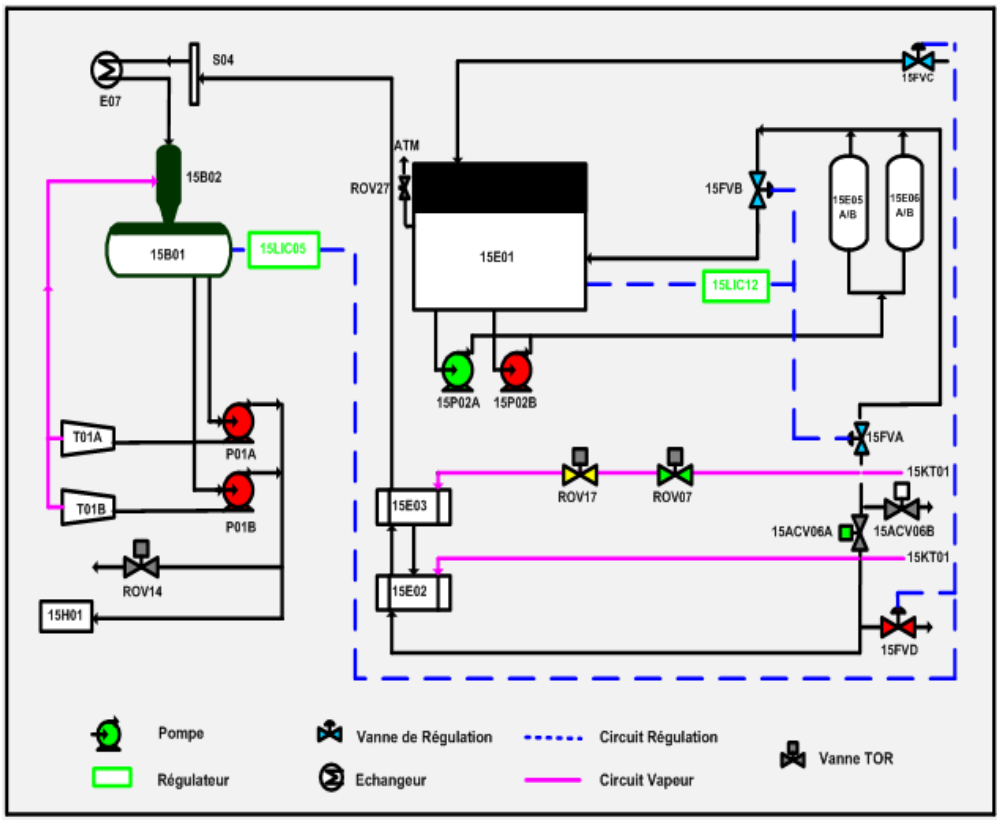

FIG. 5.1 – Schéma technique de Poste D'eau

5.2.1.1 Principe de fonctionnement

Les échanges d'alimentation en eau dans le poste d'eau se font entre deux circuits.

- Circuit de condensation.
- Circuit d'alimentation.

- **Circuit de condensation**

L'eau d'extraction va alimenter les réchauffeurs en passant par l'électrovanne 15 FVA vers le dégazeur (15B02). L'électrovanne 15FVB retourne vers le condenseur (15E01). Ces deux vannes sont commandées par le contrôleur 15 LIC 12. Le niveau d'eau dans le condenseur est réglée à 65% par une boucle de régulation 15 LIC 12.

- **Circuit d'alimentation**

Les électrovannes 15 FVD et 15FVC sont commandées en split range par le régulateur 15 LIC 05 qui maintient un niveau constant dans la bâche alimentaire.

L'eau d'extraction est ensuite dirigée vers les condenseurs 15 E 02 et 15 E 03 en contact des soutirages vapeur 2 et 3 du turbo ventilateur (15KT01). Les deux réchauffeurs sont munis de By-pass, l'eau d'extraction va ensuite alimenter la bâche (15B01). Le niveau d'eau dans la bâche alimentaire est réglée à 65% par une boucle de régulation 15 LIC 05.

5.2.2 La Chaudière

Une chaudière est un générateur de vapeur, elle à pour but d'élever la température de l'eau, jusqu'au changement d'état, c'est à dire sa transformation en vapeur, puis de l'amener à une pression et une température bien déterminées.

Dans la construction d'une chaudière nous distinguons :

- Une charpente métallique avec son habillage, maçonnerie casing.
- La chaudronnerie : tubes, boite de retour, écrans, vaporisateur, surchauffeurs, ballons.
- La chambre de combustion : brûleurs, circuits air et auxiliaires.

✓ **Les constituants de la chaudière**

- Elle est de type à circulation naturelle.
- Elle est de type pressurisé, alimentée en air par un turbo ventilateur de soufflage.
- Elle utilisé comme combustible du gaz de flash (N2=40%, C1=60%).
- Au démarrage elle consomme du gaz naturel.

La chaudière caractérise par :

❖ Vaporisation normale 320t/h.
❖ Vaporisation en pointe 350t/h (pendant 1 heure uniquement).
❖ Température eau alimentaire 135 C°.
❖ Pression vapeur sèche (Vs) 65 bars.
❖ Température vapeur sèche (Vs) 495 C°.
❖ Pression du timbre 82 bars effectifs.

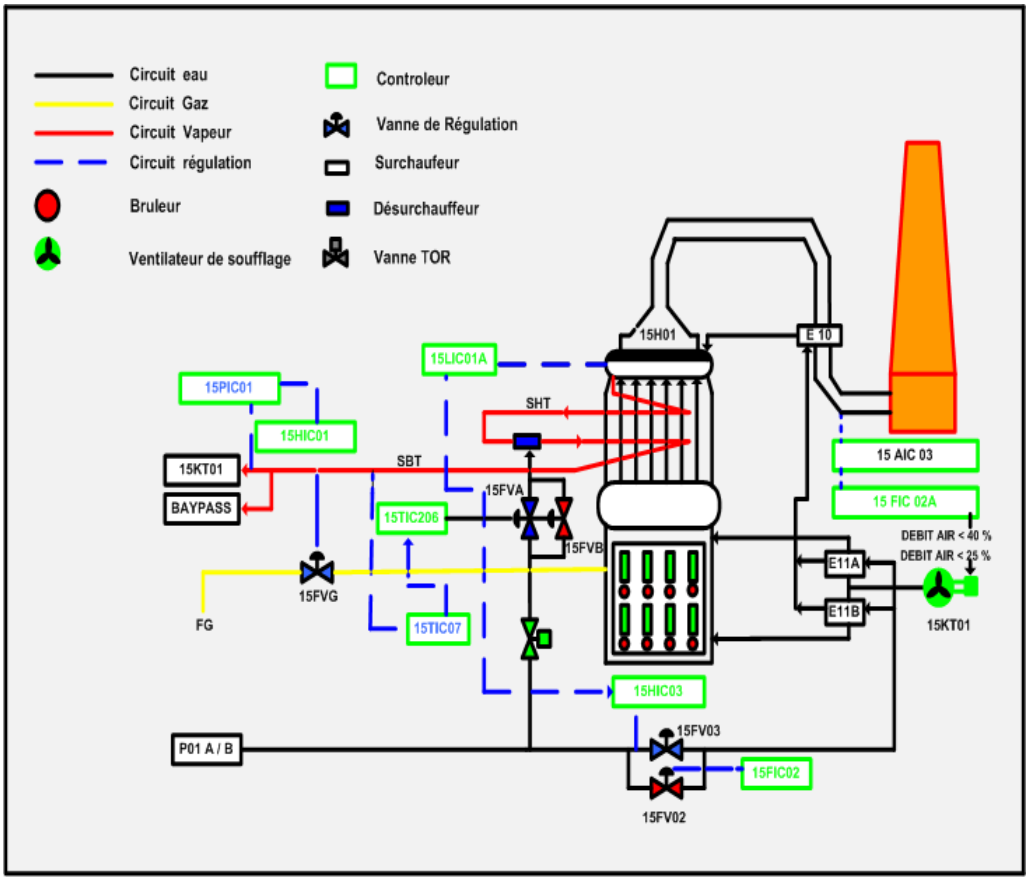

FIG. 5.2 – Schéma technique du générateur de vapeur

5.2.2.1 Principe de fonctionnement

On a dit qu'une chaudière est un générateur de vapeur destinée à produire de la vapeur sèche à une pression et une température bien déterminée.

Pour obtenir cette vapeur, on brûle dans la chaudière un combustible qui peut être :

- ✓ Soit un combustible solide (houille, charbon, bois, tourbe…).
- ✓ Soit un combustible liquide (rendue de pétrole, mazout, goudron houille asphalte).
- ✓ Soit un combustible gazeux (gaz panne, gaz de haut fourneau, gaz de raffinerie, ou bien gaz naturel comme dans notre cas).

Les premières chaudières étaient de simple récipient étanche sous lesquelles on place le foyer. La chaleur transmise et par conséquent la quantité de vapeur produite, étaient évidemment en relation direct avec la surface chauffée. On a cherché à augmenter celle-ci en décomposant la surface en éléments auxquels on a donné une section circulaire pour qu'ils restent à la pression, et un petit diamètre pour augmenter la surface réalisable dans un volume donné.

Le combustible est brûlé dans une capacité appelée foyer. Cette capacité peut être intérieure à la chaudière (chaudière marine cylindrique, chaudière de locomotion). Elle est généralement extérieure en particulier dans les chaudières à tubes d'eau. Elle est alors limitée sur un certain nombre de ces surfaces par des maçonneries réfractaires.

Les gaz produits par la combustion traversent la chaudière et cèdent une partie de leur chaleur aux parois en provoquant la vaporisation de l'eau. A la sortie de la chaudière se trouve une cheminée par laquelle ils sont évacués dans l'atmosphère. La différence de densité entre les gaz chauds de la cheminée et de l'air extérieur froid provoque la montée des gaz et assure le tirage qui mène l'air dans le foyer.

Dans les installations poussées, on active le tirage par un ventilateur ou même par deux ventilateurs l'un refoulant l'air dans le foyer (ventilateur de gonflage), l'autre aspirant les fumées (ventilateur de tirage).

L'eau est amenée à la chaudière par une pompe alimentaire. La puissance de la chaudière est caractérisée par l'intensité de vaporisation exprimée en Kg produit par heure.

Les échanges de vapeur dans une chaudière se font entre deux circuits.

- Le circuit de combustion.
- Le circuit hydraulique.

- **Le circuit de combustion**

L'air est fourni par un turbo ventilateur (15KT01), il est réchauffé ensuite par le réchauffeur d'air (E11A / E11B) avant d'entrer dans les brûleurs en utilisant la chaleur des gaz évacués. Le débit d'air est réglé en fonction du combustible (FG) afin d'avoir une combustion complète.

La vapeur saturée sort du ballon supérieur, passe par la surchauffeur (dans le foyer). Ainsi, sa température augmente tandis que sa pression varie légèrement. Le circuit de combustion est réglé à 65% par une boucle de régulation (15HIC01).

- **Le circuit hydraulique**

L'alimentation en eau se fait à partir d'une pompe contrôlée par deux vannes montées en parallèles (15FV03/ 15FV02). La première sert à remplir le ballon (15H01) et l'autre à la régulation (organe de réglage) ; d'où elle sort pour passer ensuite dans un économiseur (E11) afin de se réchauffer par les calories des fumées évacuées et rentre enfin dans le ballon supérieure pour produire de la vapeur.

Le circuit hydraulique est réglé à 65% par une boucle de régulation (15LIC01A).

5.2.3 Conduite et exploitation de la chaudière

5.2.3.1 Conduite

Les personnes de conduite des générateurs de vapeur est composé :

- D'un ″opérateur site″ situe localement à proximité des générateurs dont le rôle est d'assurer toutes les opérations préliminaires de mise à disposition des équipements, les opérations d'exploitation locales et les tâches d'allumage de la chaudière localement.
- D'un tableautiste situé en salle de contrôle ayant à charge (60%) le pilotage du générateur lorsque ce dernier est en position ″Distance″**.** Il a toutefois la possibilité également d'effectuer l'allumage de la chaudière.

Le matérielle d'acquisition, de régulation et d'automatisme est réparti dans les locaux techniques. Le matérielle de conduite, de gestion, de supervision est situé en salle de contrôle.

5.2.3.2 Exploitation, mode de conduite

Cette chaudière est prévue pour être exploitée avec une présence permanente de surveillance.

La chaudière est composée de 8 brûleurs et 8 allumeurs répartis en deux niveaux, chaque niveau se divise en deux diagonales. Une diagonale est constituée de 2 brûleurs et de 2 allumeurs.

Le turbo ventilateur est commandé localement depuis le tableau local ou à distance depuis la salle de contrôle.

L'allumage de la chaudière est fait localement depuis un tableau local d'allumage ou à distance en SdC. Le pilotage de la chaudière, après son allumage est réalisé depuis la salle de contrôle.

Le pilotage concerne la montée la pression et température, la conduite automatique des régulations en marche normale, l'arrêt manuel et le pilotage des isolements du réseau vapeur sortie chaudière.

Les automatismes propre aux brûleurs son traités dans le ″BMS″(Burner Management system).

Les fonctions de régulation et d'asservissement des auxiliaires ainsi que les fonctions de télécommande et d'affichage des données son assurées par un système de conduite distribué (DCS de Honeywell) dont les unités de traitement spécifiques à la chaudière sont reliées au réseau de communication général du complexe.

5.2.3.3 Système de Contrôle Distribué (DCS)

Le DCS c'est un système de contrôle commande qui collecte les données du terrain (Grandeur mesurés…), les analyses et les traite, pour apporter les corrections nécessaires et améliorer les paramètres du processus. Ces données du terrain peuvent être stockées et archivées pour les consulter et les utiliser pour développer et élaborer des stratégies de contrôle avancé.

5.2.3.3.1 Éléments constitutifs du système DCS

Le TPS est un système distribué, articulé autour de trois réseaux de communication :

- Le réseau local de commande (LCN) : données de contrôle et de commandes.
- Le réseau universel de commande (UCN) : données provenant d'équipements de conduit de procédé.
- Le réseau PIN "Plant Information Network" : données du procédés pour fin de supervision.

- **Réseau local de contrôle (LCN)**

Le réseau local de contrôle et de commande est l'épine dorsale du TDC3000 sa fonctionnalité majeur est de relier les stations universelles (US) les modules de traitement (application et historisation) et les modules d'interface.

- **Station universelle (US)**

Permet de visualiser toutes les informations reçues des équipements connectées au procédé, dessous système d'instrumentation sous forme de vues (synoptiques), à travers cette fenêtre les opérateurs émettent les commandes et les consignes aux instrumentations du site.

- **Station de travail (GUS)**

Elle possède une unité centrale PC du constricteur DELL qui lui permet de visualiser synoptiques et des informations récuses du procédé, elle représente une véritable interface homme machine on peut dire que c'est une amélioration de la station US.

- **Module historique (HM)**

C'est comme le disque dur de nos PCs. Il stocke les configurations du DCS aussi bien les configurations de tous points de l'usine. Il sauvegarde également les synoptiques et les graphiques qui sont affichés dans les US et les GUS. Il assure en plus l'historique et l'archivage des événements système ainsi que les données de fonctionnement de l'usine.

- **Module d'application (AM)**

Ce module offre plus de capacités de traitement et de calcul servant à développer des stratégies de contrôle plus performantes.

- **Modules d'interfaces**

La fonction de ces modules et la conversion de données entre le protocole de communication du LCN et celui d'un autre réseau ou d'un autre équipement pour assurer une disponibilité totale de toutes les informations et à tous les niveaux.
Les modules d'interface faisant partie de l'architecture retenue au complexe sont :

- ✓ **Modules d'interface réseau (NIM)**

La fonction de ces modules est la conversion de données entre le protocole de communication des différents réseaux locaux de DCS.

- ✓ **Modules d'interface réseau de calculateurs (PLNM)**

Il a un rôle de passerelle qui assure une communication du VAX le réseau LCN par le protocole LAT.

- **Réseau universel de commande (UCN)**

C'est un réseau quo sert à relier les équipements de régulation et le (xPM). Il est constitué de deux câbles coaxiaux redondants. La connexion avec le LCN se fait par une passerelle (NIM).

- ➢ **PM, APM, HPM**

Ceux-ci sont comme les cerveaux du DCS, ce sont les modules de contrôle commande du DCS. Équipement connecté au procédé avec un jeu de fonctions de régulation et d'acquisition très puisant, il permet aussi grâce à la carte E/S d'interface série, une interface à un grand nombre de points venant d'autres équipements d'acquisition de données (capteurs).

- **Le réseau PIN (Plant Information Network)**

C'est un réseau qui relie un ensemble de station de travail (GUS et PC des managers) et un serveur de type DEC VAX, interconnecte avec un réseau industriel (LCN) via un équipement d'interface à double entrée (PLNM).

5.1.3.3.2 Les entrées-sorties du DCS

Les entrées sorties du DCS peuvent être numériques ou analogique. Numérique comme des signaux de 'Marche/Arrêt' et de début/fin. La plupart des E/S du contrôleur de processus sont considérées analogique. Ce sont les points où les instruments de site sont (Capteur, Actionneur).

5.2.3.3.3 Avantages de la conduite par DCS

- ✓ Fiabilité et performance accrues dans le fonctionnement des installations.
- ✓ Fonctionnement automatique nécessaire peu d'interventions.
- ✓ Une plus grande sécurité des installations.
- ✓ Convivialité de la fenêtre de conduite du procédé qui peut être également un outil d'autoformation.
- ✓ Fiabilité de l'information et le gain de temps disponibilité totale et à tous les niveaux des informations du procédé.

5.2.4 Description des fonctions de régulations (Fonction DCS)

5.2.4.1 Régulation de la section Poste D'eau

Deux conditions sont nécessaires pour assure une bonne alimentation d'eau :

1. Assurer la stabilité du niveau d'eau à 65% dans le condenseur (15 E 01).
2. Assurer la stabilité du niveau d'eau à 65% dans la bâche alimentaire (15 B 01).

5.2.4.1.1 Régulations qui assurent le Poste D'eau

1. Le niveau d'eau dans le condenseur réglée à 65% par une boucle de régulation 15 LIC 12.
2. Le niveau d'eau dans la bâche alimentaire réglée à 65% par une boucle de régulation 15 LIC 05.

- **Vérifications préliminaires du Poste D'eau**

  - Vannes aspiration et refoulement 15 PM 02 A/B ouvertes.
  - Godets d'huile lubrification palier.
  - Arrosage des paliers en service. Vannes (04) entrée /sortie des 15 E 05 A/B ouvertes.
  - Vannes de garde 15 LCV 05 B ouvertes.
  - Vannes de garde 15 LCV 05 A (U50) disposées.
  - Vannes de garde 15 LCV 12 A/B disposées.
  - Vannes d'isolement 15AAH 06 ouverte.
  - Vannes entrée et sortie 15 E02 et 03 ouvertes.
  - Ouvrir la vanne à chaîne vers le dégazeur.
  - Mettre en configuration normale les appareils de contrôle, et régulation en service.

5.2.4.2 Régulation de la section Chaudière

Cinq conditions sont nécessaires pour assurer une bonne génération de vapeur :

- Régulation de chauffe (Un facteur air-gaz correct).
- Réglage des débits des combustibles.
- Réglage du débit d'air.
- Régulation du niveau ballon (à 50%).
- Régulation de température de vapeur surchauffée (495 C°).

5.2.4.2.1 Régulations qui assurent la Chaudière

- **Régulation de chauffe**

Le seul cas de marche envisagé est une marche au ″fuel gaz″.

Le régulateur de charge 15 PIC 01 pilote les quantités d'air et de combustible (gaz) introduites dans la chaudière.

Afin d'être que le rapport air/combustible soit, dans les régimes transitoires, toujours en excès d'air, un système de sélection de signal est prévu.

Le dispositif de contrôle de combustible et de l'air permet d'éviter manque ou un excès d'air pendant les variations de charge, son principe de fonctionnement de contrôle est le suivant :

- ✓ Pour éviter le manque d'air lors d'une augmentation du signale de charge, la consigne du régulateur de débit d'air suit immédiatement le signale de charge ce qui provoque une augmentation du débit d'air. Par contre, la consigne du régulateur de débit combustible est asservie à son tour débit d'air.
  En synthèse une montée de charge entraîne l'air et l'air entraîne à sa tour le combustible.

- ✓ Pour éviter le manque d'air lors d'une diminution du signale de charge, la consigne du régulateur de débit d'combustible suit immédiatement le signale de charge, ce qui provoque une diminution du débit combustible. Par contre, la consigne du débit d'air est asservie à son tour débit combustible.

  En synthèse une baisse de charge entraîne le combustible et le combustible entraîne l'air à sa tour le combustible.

- **Réglage de l'excès d'air**

Le régulateur d'excès d'air 15 AIC 03 (dont la mesure est issue de l'analyseur d'oxygène des fumées) reçoit en consigne un signale variant avec le débit vapeur, donc avec la charge réelle de la chaudière; cette correction est telle que l'excès d'air à basse charge est plus important qu'à forte charge afin de conserver une bonne combustion.

Le signal d'excès d'air venant du régulateur corrige le signal débit d'air entrant en mesure dans le régulateur de débit d'air 15 FIC 02 A.

- **Réglage des débits combustible**

Le débit de combustible (gaz) est contrôlé par 2 vannes en split-range. Les vannes, selon le cas de marche, ont sous la dépendance de l'un ou de l'autre des 2 régulateurs suivantes :

- ✓ Un régulateur de pression, qui permet de maintenir un débit suffisant dans les bruleurs.
- ✓ Un régulateur de débit qui reçoit en consigne, à travers un sélecteur bas, le signale de charge.

- **Réglage du débit d'air**

Le signale de sortie du régulateur de débit d'air 15 FIC 02 A passe par un module de sélection de signal haut d'une consigne de sortie d'un régulateur de débit mini 15 FIC 02 B.

Sur arrêt normal ou de sécurité de la chaudière, le ventilateur n'est pas arrêté automatiquement. Les ventelles restant en position : mémorisation de la position lors de l'arrêt pendant 1 mn.

- **Régulation du niveau ballon**

Cette régulation du niveau a pour but de maintenir le plan d'eau du ballon supérieur à une position prédéterminée quelle que soit la demande de vapeur en maitrisant les phénomènes de gonflement et de tassement apparaissant avec les variations importantes de la charge.

La régulation est de type à 1 élément (niveau) dans les périodes de démarrage (sur la petite vanne 15 FV 02) et à 3 éléments (niveau, débit vapeur, débit d'eau) sur la petite vanne 15 FV 02 puis la grosse vanne 15 FV 03.

Le passage de la régulation de niveau de 1 élément à 3 éléments se fait lorsque le débit de vapeur devient supérieur à 20%.

Le passage de la régulation de niveau de 3 éléments à 1 élément se fait lorsque la demande d'ouverture de la vanne principale devient faible et que l'on a un débit de vapeur inférieur à 20%.

Afin de limiter les conséquences liées au phénomène de tassement (et inversement pour le gonflement), la consigne de niveau varie en fonction de la charge.

- **Régulation de température de vapeur surchauffée**

Il s'agit de maintenir constante la température de vapeur surchauffée sortie chaudière.

La régulation est de forme ″cascade″ c'est-à-dire la sortie du régulateur principale 15 TIC 07 agit comme consigne du régulateur esclave 15 TIC 206.

La sortie du régulateur esclave 15 TIC 206 est envoyée aux deux vannes de désurchauffe 15 FV A/B.

Les vannes sont en régulation avec la présence de toutes les conductions suivant :

- Chaudière en marche.
- Vanne motorisée ouverte 15 FV M.
- Débit vapeur > 10%.

Ou à la fermeture de la vanne motorisée d'isolement (durée d'ouverture d'une minute). Sur compte rendu de fermeture des 2 vannes de régulation, fermeture de la vanne motorisée d'isolement.

5.3 Proposition d'un modèle DEVS du simulateur

Pour modéliser notre système, nous avons utilisé le formalisme DEVS. Nous avons, ainsi fait ressortir cinq modèles couplés pour notre système. Chaque modèle couplé se compose d'ensembles de modèles atomiques.

- Le modèle couplé alimentation.
- Le modèle couplé condensation.
- Le modèle couplé combustion.
- Le modèle couplé hydraulique.
- Le modèle couplé perturbation.

Les quatre premiers modèles couplés représentent toutes les opérations effectuées par des organes de la chaudière dans les quatre circuits constituants son fonctionnement. Ceci concerne les opérations des circuits alimentation, condensation, combustion et hydraulique. Le dernier modèle couplé simule les opérations qui perturbent le fonctionnement des quatre modèles couplés précédents.

5.3.1 Modèle couplé DEVS assurant l'opération de l'alimentation en eau de la chaudière (Alimentation)

- Le modèle (15LIC05) : représente le régulateur dont le rôle concerne la régulation de la bâche. Le régulateur 15LIC05 envoie les ordres d'ouverture et fermeture (o_al)[1] vers la vanne 15FVC jusqu'à ce le niveau mesuré soit égale au niveau désiré (65%).

- Le modèle (15B01) : représente la bâche 15B01, son rôle concerne la vérification du niveau de l'eau (n_b) dans la bâche.

- Le modèle (15FVC) : représente la vanne 15FVC, cette vanne contrôle le débit de l'eau (d_e_c) entrant dans le condenseur.

5.3.2 Modèle DEVS réalisant l'opération de condensation de l'eau (Condensation)

- Le modèle (15LIC12) : représente le régulateur 15LIC12, son rôle concerne la régulation du condenseur. Ce régulateur envoie les ordres d'ouverture et de fermeture (o_cn) vers la Vanne 15FVA jusqu'à ce le niveau mesuré soit égal au niveau désiré (65%).

- Le modèle (15E01) : représente le condenseur 15E01, son rôle concerne la vérification de la variation du niveau de l'eau (n_c) dans le condenseur.

- Le modèle (15FVA) : représente la vanne 15FVA, son rôle est de contrôler le débit de l'eau (d_e_b) sortant vers la bâche.

5.3.3 Modèle couplé DEVS permettant d'effectuer l'opération de combustion (Combustion)

- Le modèle (15HIC01) : représente le régulateur 15HIC01. Son rôle est de contrôler la combustion et de régler le débit de vapeur (d_v). En effet, si ce débit n'est pas compatible avec la charge de la chaudière, il envoie alors des commandes de régulation pour l'ouverture ou la fermeture (o_cm) vers le turbo ventilateur (15KT01) et la Vanne Gaz (15FVG) afin d'aboutir aux grandeurs désirées.
- Le modèle (15KT01) : représente le turbo ventilateur 15KT01, son rôle est de contrôler le débit de l'air (d_a).
- Le modèle (15FVG) : représente la vanne 15FVG. Son rôle est de contrôler le débit gaz (d_g).

[1]Les annexes 1 et 2 détaillent toutes les variables utilisées dans ce texte.

5.3.4 Le modèle couplé DEVS hydraulique

- Le modèle (15LIC01A) : Il représente le régulateur 15LIC01A. Son rôle est de régler le niveau du ballon. Il envoie les commandes d'ouverture et fermeture (o_hy) vers la Vanne 15FV03 jusqu'à ce le niveau mesuré soit égale au niveau désiré (50%).

- Le modèle (15FV03) : représente la vanne 15FV03. Son rôle est de contrôler le débit de l'eau d'alimentation (d_e) venant du poste d'eau.

- Le modèle (15H01) : représente le ballon 15H01. Son rôle est de calculer le niveau dans ce ballon (n_b) par l'utilisation des grandeurs fournies par le régulateur 15HIC01 et la vanne 15FV03.

5.3.5 Modèle couplé DEVS assurant l'opération de perturbation de système (Perturbation)

- Le modèle APM : Son rôle est de régler et surveiller le système. Il envoie périodiquement des ordres (o_s) et des mesures aux modèles du système ; il assure le raccordement entre ces éléments grâce à ces différentes voies de communication.

- Le modèle GUS : Il représente l'interface graphique utilisateur. Son rôle est de déclencher les perturbations, c'est à travers cet agent que l'opérateur pilote le système en saisissant les consignes de la charge de la génération de vapeur (c_g).

La figure 5.3 représente la structure globale du simulateur basé sur le modèle DEVS proposé.

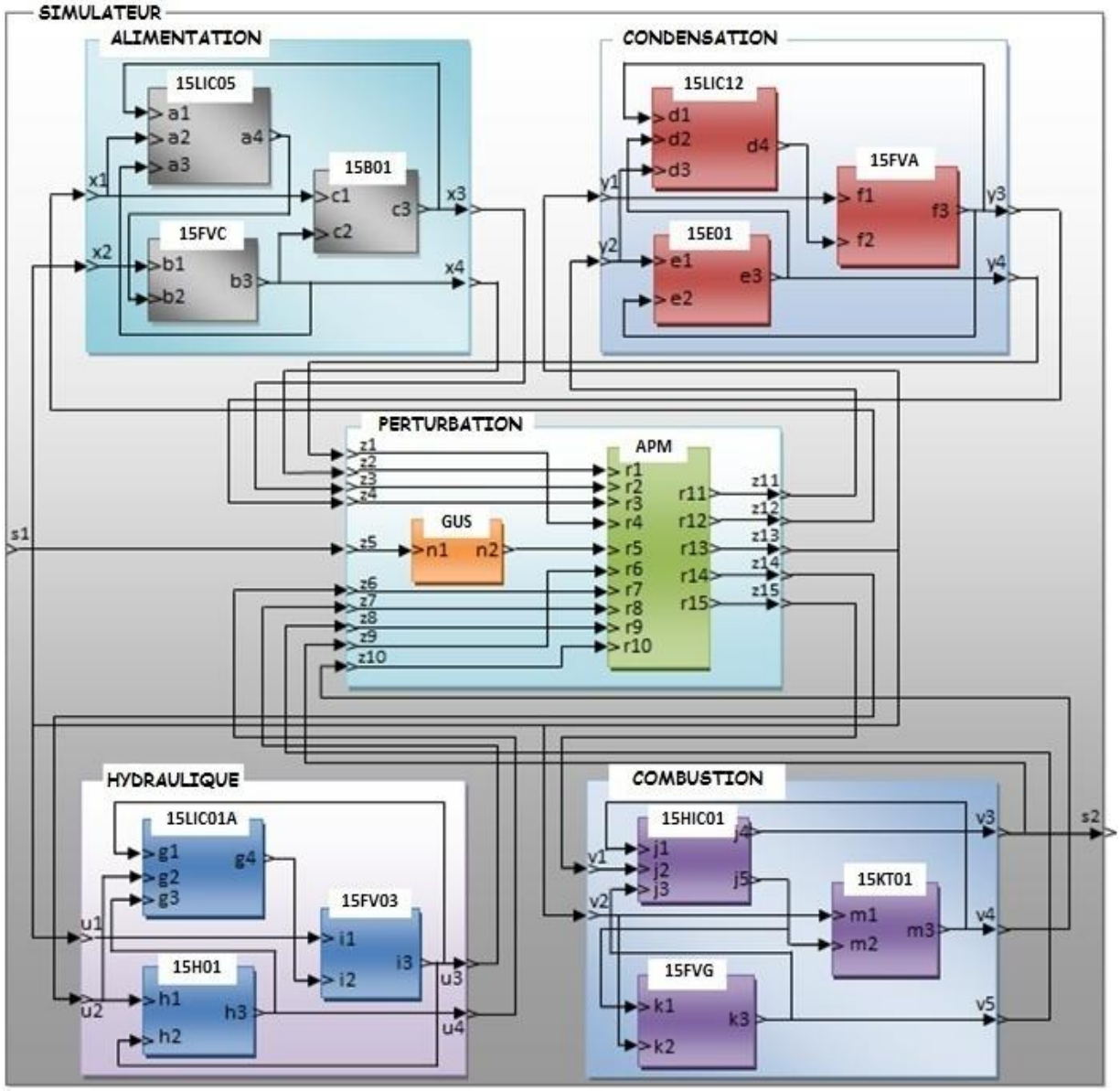

FIG. 5.3 – Modélisation DEVS du simulateur

5.3.6 Exemple de comportement d'un modèle atomique et spécification DEVS

Afin de donner un exemple pratique du comportement du système modélisé, considérons le régulateur 15LIC12 comme illustré dans la figure 5.3. Initialement, ce composant est dans la phase *'passif'*, quand de nouvelles entrées {*d_e_b*, *n_c*, *d_e_c*} arrivent respectivement sur les ports d'entrée {*d1, d2, d3}*, il va régler la condensation et place la phase *à 'actif'* et sigma (σ) *à ' temps_condensation'*. Les valeurs entrantes seront aperçu par la fonction de transition externe, quand une valeur arrive tandis que le régulateur est dans la phase *'actif '*, il l'ignore et met à jour σ, en laissant l'état inchangé. Quand la régulation est terminée, le régulateur place l'ordre {*o_cn*} sur le port {*d4*} et retourne à la phase *' passif '*. L'envoi des ordres est

fait par la fonction de sortie occurrente juste après la fonction de transition interne. Cette dernière renvoie le modèle à l'état vide { ∅ } dans lequel la phase est '*passif* ' et σ est " *infini* ". La spécification DEVS du régulateur 15LIC12 est présentée dans la table 5.1 ci-dessous.

15LIC12
• InPorts= {"d1", "d2","d3"} • X = {(d1, d_e_b), (d2, n_c), (d3, d_e_c)} • OutPorts = {"d4"} • Y= {(d4, o_cn)} • δ_{ext} (phase, σ, e, X) = ("actif ", temps_condensation) si phase = =" passif ", (phase, σ-e) sinon. • δ_{int} (phase, σ) = ("passif ", ∞) si 15LIC12.X.V= =Ø ("actif ", temps_condensation) sinon. • λ ("actif ") = o_cn • *ta* (phase, σ) = σ

TAB. 5.1 – Spécification DEVS atomique du régulateur 15LIC12

Les spécifications formelles DEVS de tous les modèles atomiques et couplés du simulateur sont résumées dans les annexes 1 et 2.

Les algorithmes de transformation présentés dans la section 4 permettent d'obtenir les groupes et les agents correspondants aux CDEVS et ADEVS définis précédemment.

Le tableau suivant montre les différents groupes et agents obtenus. Quelques exemples d'interactions entre ces agents sont fournis dans la section suivante.

Les différentes composantes de cette application ainsi que les interactions entre agents et les groupes d'agents sont présentés dans la figure 5.4.

Les agents AGC_PER_R et AGC_PER_E assurent respectivement les réceptions et les émissions de messages entre les groupes d'agents et le groupe perturbation qui à son tour est chargé de gérer l'interface de communication avec l'utilisateur. Ce schéma est conforme au formalisme DEVS modélisé en figure 5.3.

Groupes	Agents
Groupe perturbation	AGC_PER_R, APM, GUS, AGC_PER_E
Groupe alimentation	AGC_ALI_R, 15FVC, 15B01, 15LIC05, AGC_ALI_E
Groupe condensation	AGC_CON_R,15FVA, 15E01, 15LIC12, AGC_CON_E
Groupe hydraulique	AGC_HYD_R,15H01, 15FV03, 15LIC01, AGC_HYD_E
Groupe combustion	AGC_COM_R,15FV0, 15KT01, 15HIC01, AGC_COM_E

TAB. 5.2 – Groupes et agents

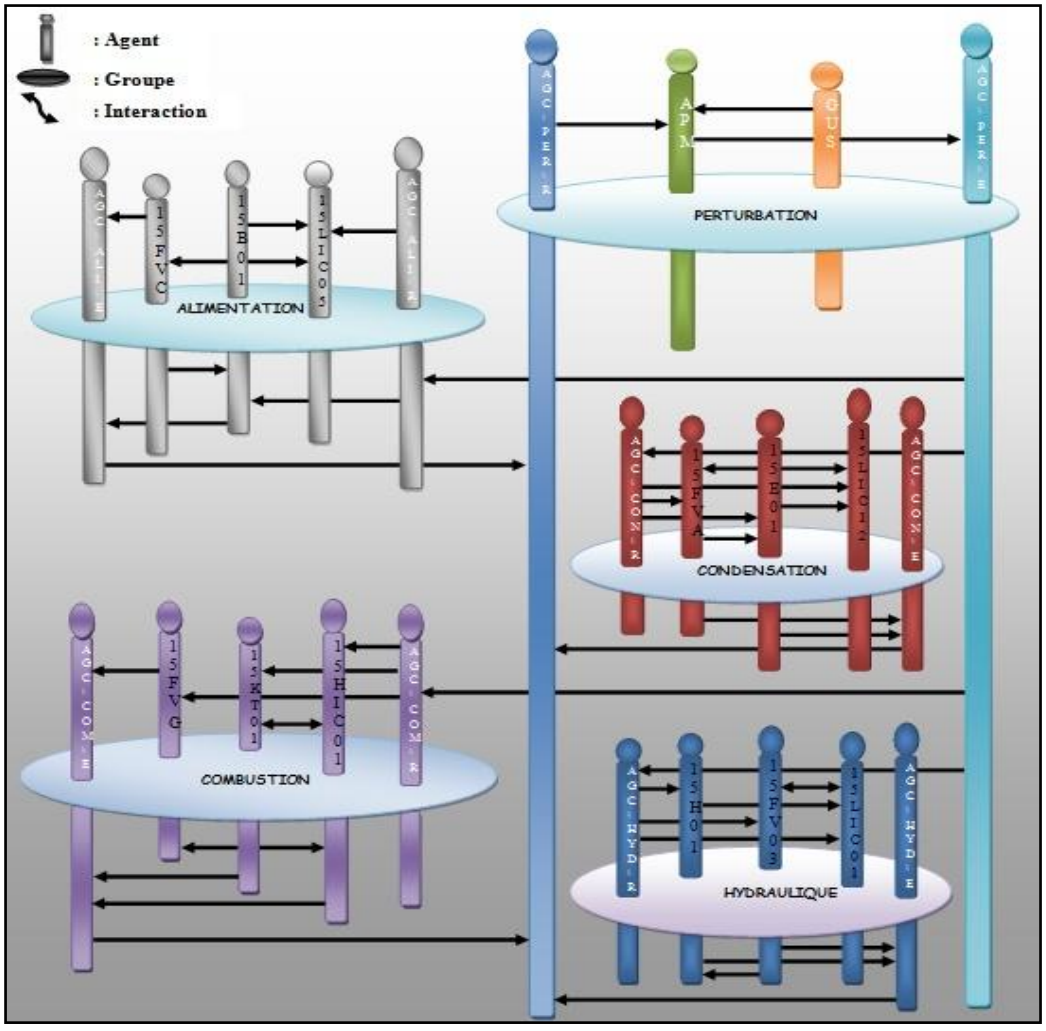

FIG. 5.4 – Structure organisationnelle du système.

5.3.7 Implémentation du système

Le simulateur est développé par l'utilisation de la plateforme multi-agents MadKit qui est un environnement logiciel offrant des outils évolués et puissants pour la gestion des agents en utilisant les concepts AGR (Agent/Groupe/Rôle) tels que définis dans Aalaadin.

MadKit est un ensemble de packages de classes Java qui met en œuvre un noyau d'agents. Plusieurs bibliothèques de messages, de requêtes ainsi que des outils de gestion de l'environnement de développement graphique et des modèles d'agent standard sont inclus dans MadKit.

L'architecture MadKit est basée sur trois principes (figure 5.5) :

- ✓ Architecture à micro-noyau.
- ✓ Agentification systématique des services.
- ✓ Utilisation d'un modèle graphique componentiel.

MADKIT utilise un agent appelé micro-noyau. Cet agent se charge de la création des groupes et de leur gestion ainsi que de la distribution des messages locaux. Tous les services hormis ceux assurés par le micro-noyau sont affectés à des agents à part entière. Ceci dans un souci d'unicité de modèle, mais également de mise à l'épreuve des systèmes multi-agents comme modèle de programmation. Cette focalisation sur l'infrastructure simplifie les phases de développement et de déploiement. Le principe essentiel de MadKit est d'utiliser partout où cela est possible la plateforme pour son propre fonctionnement [Gutknecht, 2001].

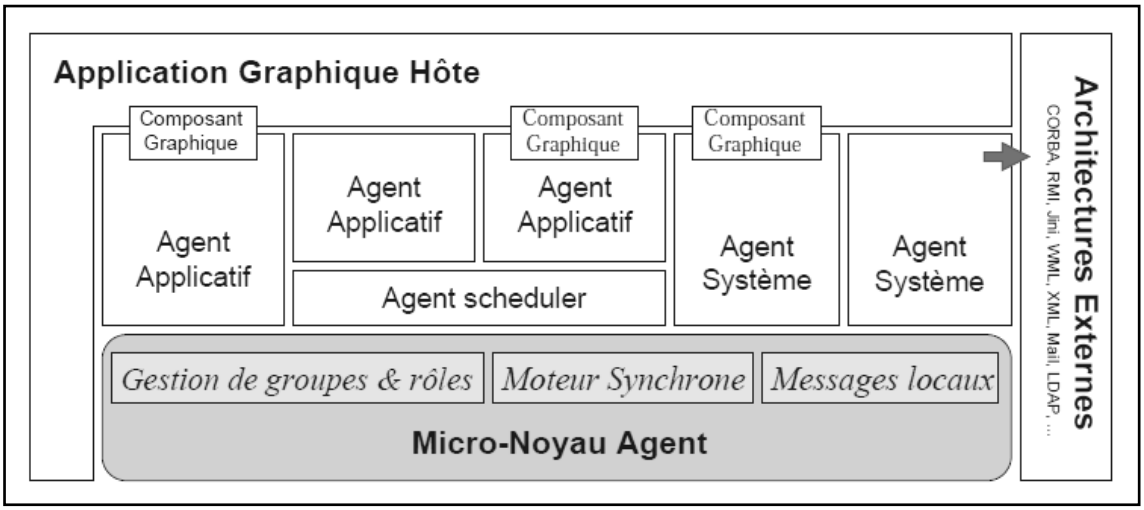

FIG. 5.5 – Architecture générale de Madkit.

Cet environnement permet d'implémenter les modèles XML DEVS tels que déterminés ci-dessus.

Pour le système concerné, les modèles peuvent être enregistrés sous format XML exploitable par MADKIT (voir figures 5.6 et 5.7). Les fichiers XML sont manipulés par la bibliothèque JDOM (Java Document Object Model) qui est une API Java qui permet la manipulation des fichiers XML. JDOM utilise un ensemble de classes plutôt que des interfaces. Ainsi, la création d'un nouvel élément revient à l'instanciation d'une classe. JDOM facilite, ainsi, la manipulation de documents XML : lecture d'un document, représentation sous forme d'arborescence, manipulation de cet arbre, définition d'un nouveau document, exportation vers plusieurs formats cibles. Le parseur défini via JAXP par défaut est utilisé par JDOM. JDOM permet d'exporter un arbre d'objets d'un document XML dans un flux, un arbre DOM ou un ensemble d'événements SAX. JDOM interagit donc avec SAX et DOM pour créer un document en utilisant ces parseurs ou pour exporter un document vers ces API, ce qui permet d'intégrer JDOM dans des traitements existants. JDOM propose cependant sa propre API [Doudoux, 2014]. La figure 5.8 présente la lecture d'un fichier XML.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<models>
  <AtomicDEVS>
    <id>1</id>
    <name>15LIC12</name>
    <inPorts>
      <port name="d1">
        <event id="" name="d_e_b" msg="v_d_e_b" />
      </port>
      <port name="d2">
        <event id="" name="n_c" msg="v_n_c" />
      </port>
      <port name="d3">
        <event id="" name="d_e_c" msg="v_d_e_c" />
      </port>
    </inPorts>
    <outPorts>
      <port name="d4">
        <event id="" name="o_cn" msg="v_o_cn" />
      </port>
    </outPorts>
    <states>
      <state name="actif_temps_con" sigma"="temps_con" />
      <state name="passif_sig" sigma"="sig" />
      <state name="actif_sig-e" sigma"="sig-e" />
      <state name="passif_infini" sigma"="infini" />
    </states>
    <f_ext>
      <delta_ext ="actif_sig-e;" phase="actif;" />
      <delta_ext ="actif_temps_con;" phase="passif;" />
    </f_ext>
    <f_int>
      <delta_int ="actif_temps_con" 15LIC12_d1="v_d_e_b" />
      <delta_int ="actif_temps_con" 15LIC12_d2="v_n_c" />
      <delta_int ="actif_temps_con" 15LIC12_d3="v_d_e_c" />
      <delta_int ="passif_infini" 15LIC12_d1 =" " 15LIC12_d2 =" " 15LIC12_d3 =" "/>
    </f_int>
    <lambda>
      <lambda = "o_cn;" phase="actif_temps_con" />
      <lambda = " " phase="passif" />
    </lambda>
  </AtomicDEVS>
  <.
   .
   .
```

FIG. 5.6 – Modèle DEVS 15LIC12 sous XML

```
  .
  .
  .
  <CoupledDEVS>
    <id>id_1</id>
    <name>CONDENSATION</name>
    <inPorts>
      <port name="y1">
        <event id="" name="o_s" msg="v_o_s" />
      </port>
      <port name="y2">
        <event id="" name="d_e_c" msg="v_d_e_c" />
      </port>
    </inPorts>
    <outPorts>
      <port name="y3">
        <event id="" name="d_e_b" msg="v_d_e_b" />
      </port>
      <port name="y4">
        <event id="" name="n_c" msg="v_n_c" />
      </port>
    </outPorts>
    <internalModels>
      <model name="15LIC12" />
      <model name="15FVA" />
      <model name="15E01" />
    </internalModels>
    <EIC>
      <coupling fromModel="CONDENSATION" from="y1" toModel="15FVA" to="f1" />
      <coupling fromModel="CONDENSATION" from="y2" toModel="15LIC12" to="d3" />
      <coupling fromModel="CONDENSATION" from="y2" toModel="15E01" to="e1" />
    </EIC>
    <IC>
      <coupling fromModel="15LIC12" from="d4" toModel="15FVA" to="f2" />
      <coupling fromModel="15E01" from="e3" toModel="15LIC12" to="d2" />
      <coupling fromModel="15FVA" from="f3" toModel="15LIC12" to="d1" />
    </IC>
    <EOC>
      <coupling fromModel="15FVA" from="f3" toModel="CONDENSATION" to="y3" />
      <coupling fromModel="15E01" from="e3" toModel="CONDENSATION" to="y4" />
    </EOC>
    <states />
    <f_ext />
    <f_int />
    <lambda />
  </CoupledDEVS>
</models>
```

FIG. 5.7 – Modèle DEVS CONDENSATION sous XML

```
static void lireFichier(String fichier) throws Exception
{
SAXBuilder sxb = new SAXBuilder();
document = sxb.build(new File(fichier));
racine = document.getRootElement();
}
```

FIG. 5.8 – Lecture d'un fichier XML par JDOM

5.3.7.1 Architecture et fonctionnement de système de simulation

La figure 5.9 présente l'architecture simplifiée du système proposé. Le module interaction (1) permet à l'agent système (administrateur) de déterminer tous les modèles atomiques et couplés du simulateur grâce à un ensemble de composants (2). Ce module contient un ensemble de modèles réutilisables que l'agent système peut enrichir au fur et à mesure. L'enregistrement des modèles se fait sous format de fiches XML (exemples : figures 5.6 et 5.7) dont les paramètres sont les composantes des équations des modèles atomiques et couplés présentées dans le chapitre 3.

Les modèles DEVS correspondants aux différents composants du procédé (2) peuvent être enregistrés sous fichiers XML (3). Le rôle principal du package JDOM (4) est de parcourir les fichiers XML.

L'extension CDEVS/AGR (5) permet d'obtenir un ensemble d'agents et de groupes d'agents conformes au modèle AGR vu auparavant (Chapitre 4). Ces agents sont intégrés dans le système multi-agents MadKit (7) pour obtenir le code exécutable du simulateur. Ainsi, l'opérateur (utilisateur) peut suivre les évolutions du procédé sur le système d'affichage (8) tout en ayant la possibilité d'interagir avec le système grâce au module interaction (1).

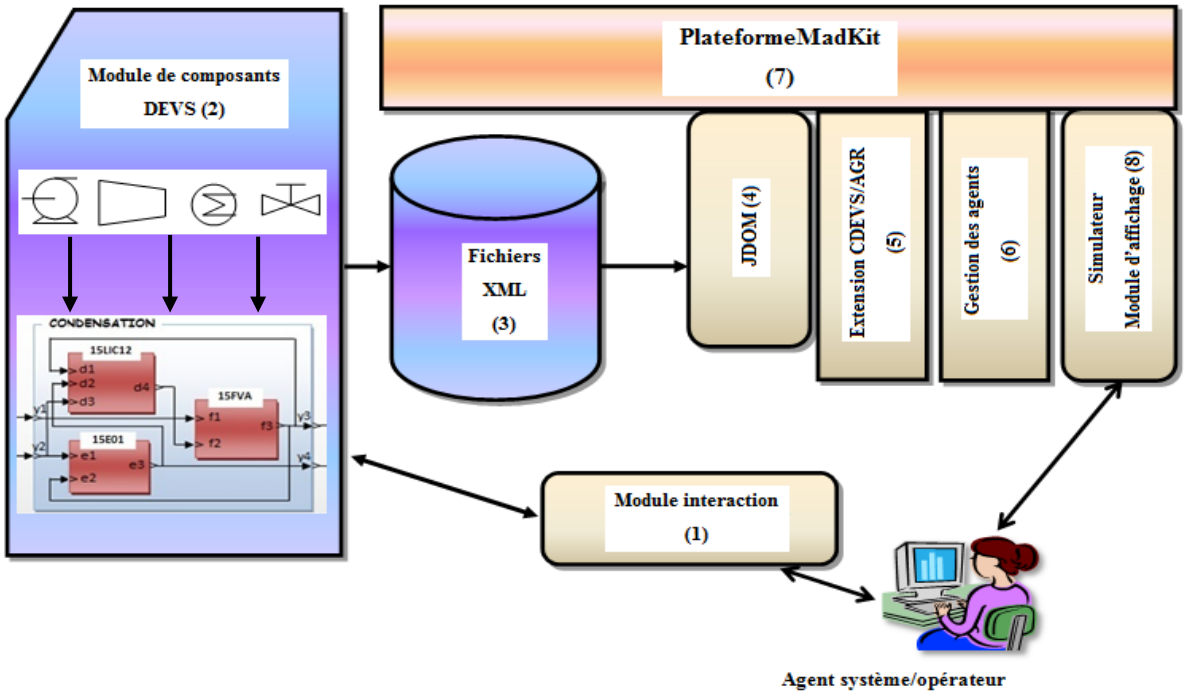

FIG. 5.9 – Architecture générale de système de simulation

Les figures 5.10, 5.11, 5.12 ci-dessous présentent quelques exemples des codes sources utilisés.

5.3.7.2 Gestion des groupes et des rôles

La création du groupe se fait par la méthode *createGroup*(..) et L'obtention d'un rôle se fait par l'instruction *requestRole*().

```
public class 15E01 extends AbstractAgent implements ReferenceableAgent
{
    public void activate()
  {
    createGroup(true,"condensation",null,null);
    requestRole("condensation","calculer_niveau_15E01");
  }

 }
```

FIG. 5.10 – Création du groupe condensation

L'agent 15E01 crée le groupe condensation et calcule le niveau dans le condenseur.

5.3.7.2.1 Envoi de messages

La figure 5.11 présente la procédure d'envoi de la valeur du nouveau débit d'eau par l'agent 15FVC (agent vanne) vers l'agent 15LIC05 (agent régulateur) :

```
void envoyer_mesure()
{
  Grand mes=new Grand();
  mes.setv1(débit_eau_c);
  mes.settype("débit_eau_c");
  ObjectMessage obj=new ObjectMessage((Object)mes);
  AgentAddress adr=getAgentWithRole("alimentation","agent_15LIC05");
  sendMessage(adr,obj);

}
```

FIG. 5.11 – Envoi de messages

5.3.7.2.2 Réception de messages

Pour la réception des messages, deux méthodes sont utilisées : *isMessageBoxEmpty()* et *nextMessage()*. La portion de code suivante montre l'utilisation de ces deux méthodes par l'agent 15LIC12 :

```
public void R_messages()
{

   while(!isMessageBoxEmpty())
   {
    msg_15LIC12=(ObjectMessage)nextMessage();
    traiter_message(msg_15LIC12);
   }

}
```

FIG. 5.12 – Réception de messages

5.3.7.2.3 Exécution du système

Pour permettre l'exécution, deux niveaux sont définis : ordonnanceur (*scheduler*) et activateur.

Le *scheduling* sous Mad-kit est délégué aux agents héritant de la classe *Scheduler*. Un agent *scheduler* a pour but de coordonner l'exécution des agents via des objets génériques appelés activateurs, ces derniers permettent au *scheduler* d'identifier un ensemble d'agents.

Le code d'un activateur contient toutes les méthodes des autres agents qu'il va invoquer au moment de son activation par l'agent *scheduler*. Ces méthodes invoquées par les activateurs sont celles des agents qui héritent de la class *AbstractAgent* et implémentent l'interface *ReferenceableAgent*et.

En effet, d'une part, l'utilisation des *scheduler* et des activateurs réduit le taux d'utilisation du processeur par les agents du système, d'autre part, l'utilisation de ces agents *scheduler* et activateurs offre plus de flexibilité.

La figure 5.13 montre la structure du *scheduler* de simulateur. Dans notre système, trois agents *scheduler* peuvent être utilisés : un agent *scheduler* poste d'eau, un agent *scheduler* perturbation et un agent *scheduler* chaudière. Ces agents utilisent un ensemble d'activateurs :

- Un activateur qui exécute le code de l'agent APM.
- Un activateur qui exécute le code de l'agent perturbation.
- Deux activateurs, un pour le traitement et l'autre pour l'affichage graphique, qui exécute les agents de poste d'eau.
- Deux activateurs, un pour le traitement et l'autre pour l'affichage graphique, qui exécute les agents de la chaudière.

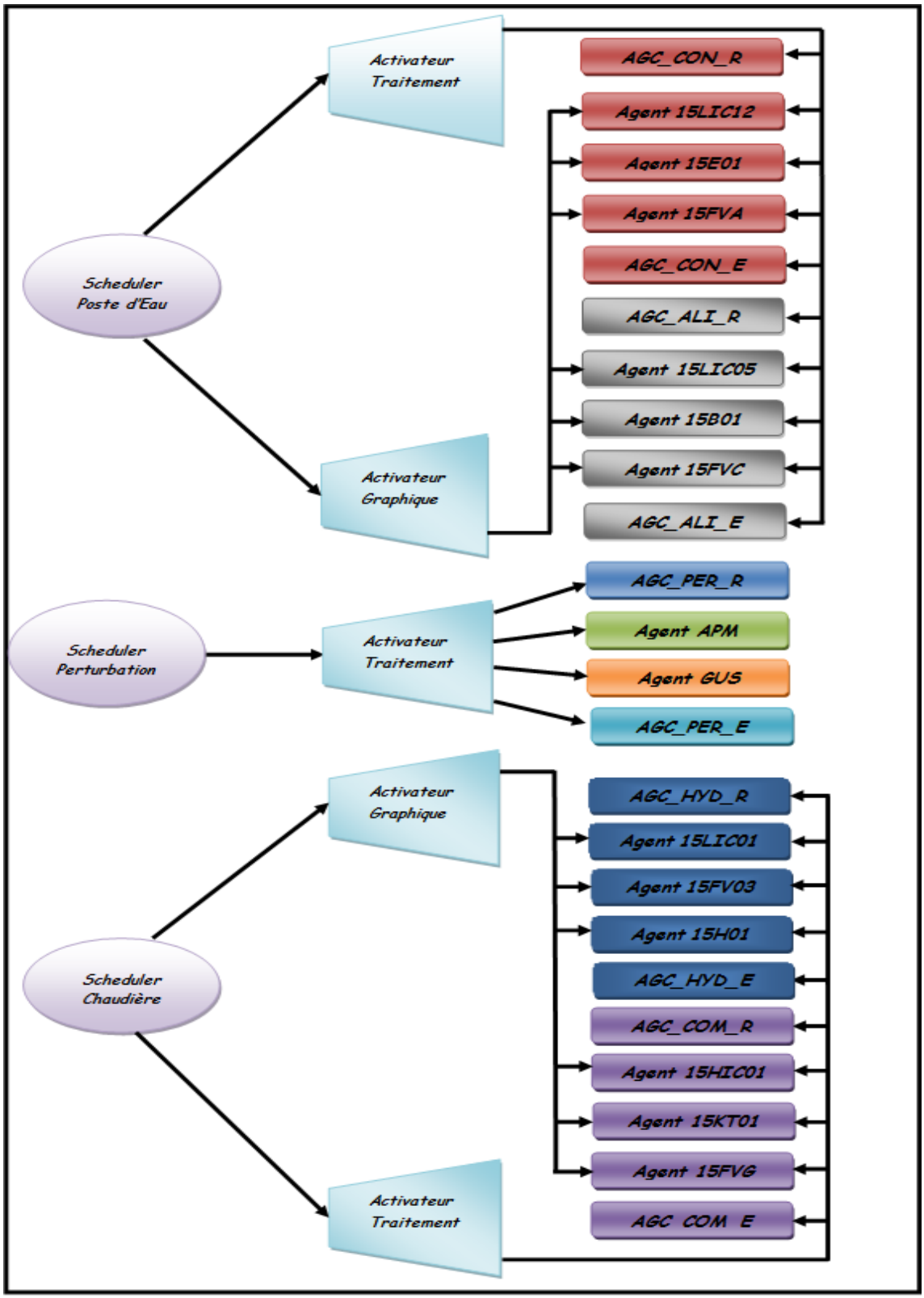

FIG. 5.13 – Structure du scheduler de simulateur

Les figures 5.14, 5.15, 5.16 et 5.17 montrent les interfaces du simulateur en vue de l'interaction utilisateur-système.

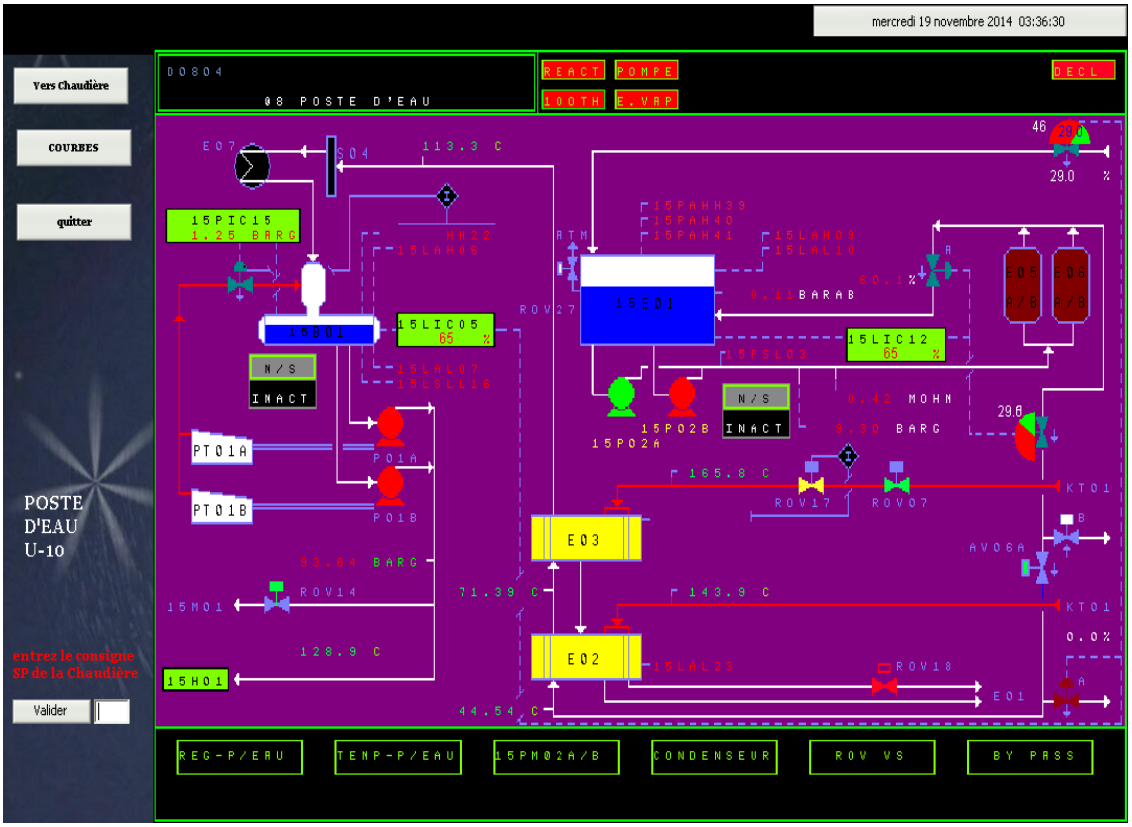

FIG. 5.14 – Interface de la section poste d'eau du simulateur

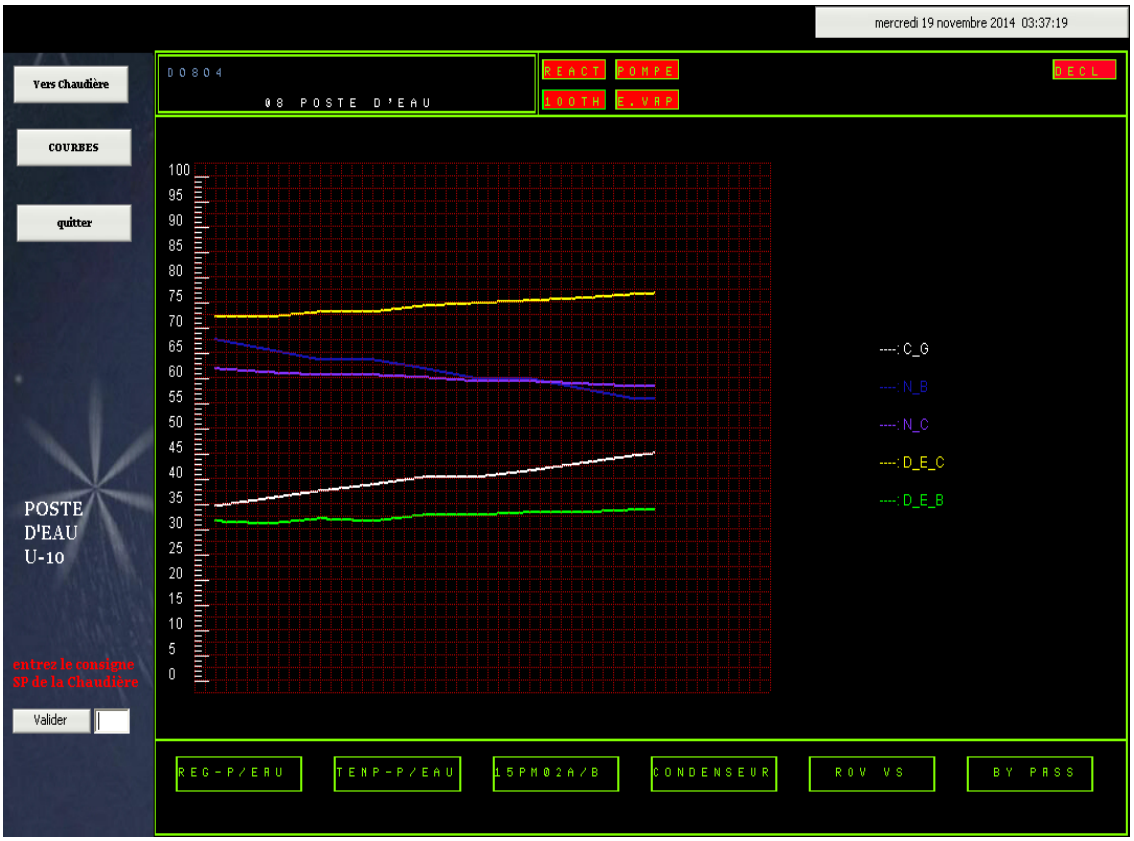

FIG. 5.15 – Interface d'évolution de la simulation poste d'eau avec le temps

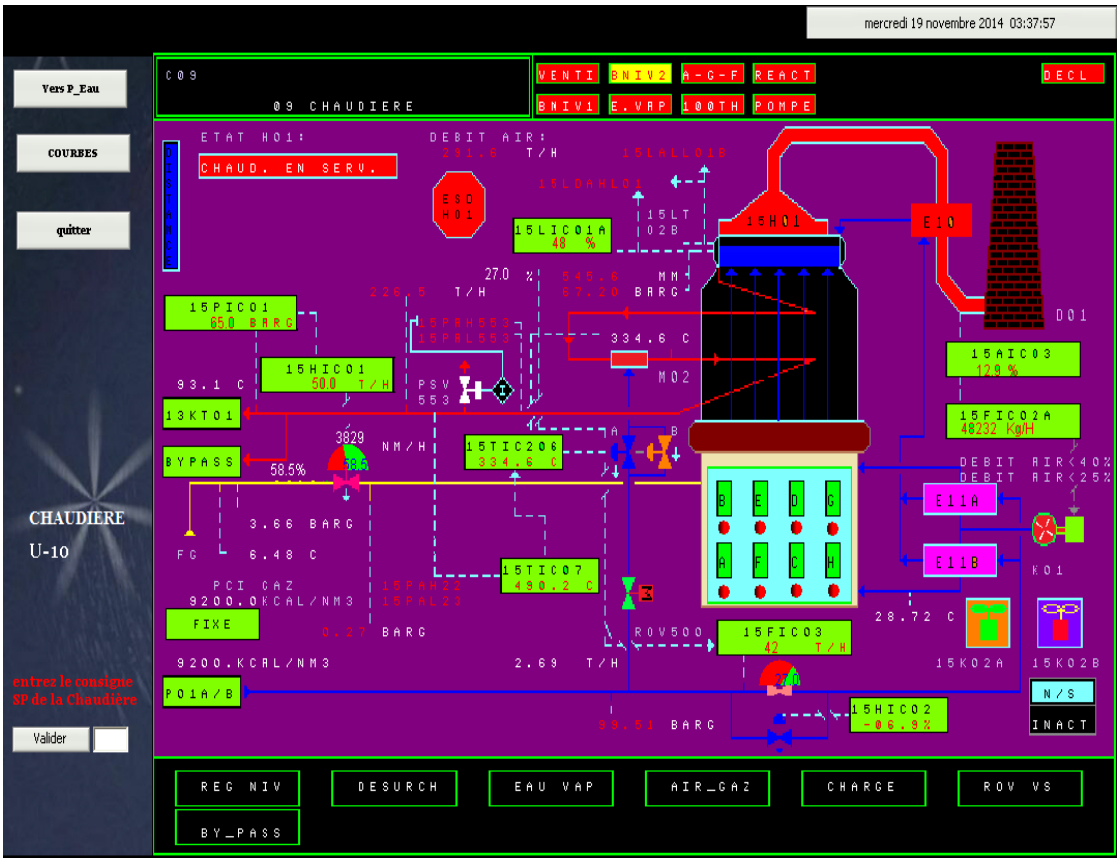

FIG. 5.16 – Interface de la section chaudière du simulateur

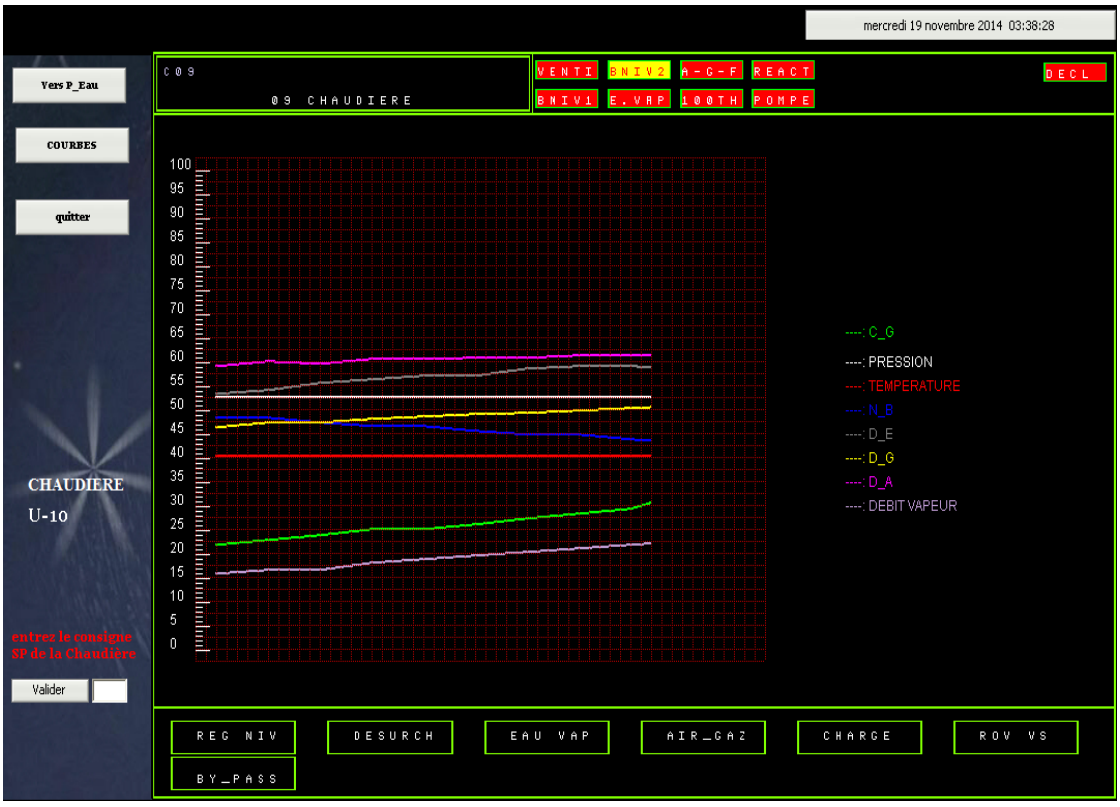

FIG. 5.17 – Interface d'évolution de la simulation chaudière avec le temps

5.3.7.3 Exploitation du simulateur

L'utilisateur dispose d'un ensemble d'entrées de commande et de données servant à initialiser le système. Un tableau synoptique lui permet de suivre les évolutions de ce système en fonction du temps.

Aussi, plusieurs modes d'exploitation du système sont offerts à l'utilisateur qui peut lancer la simulation de la section poste d'eau par les bouton "Vers_P_Eau" et "Vers_Chaudière". Ces boutons lui donneront la possibilité de basculer les interfaces de la section de poste d'eau et de la section de génération de vapeur (chaudière). Comme il peut saisir la charge du système dans le champ de texte. Les résultats de la simulation peuvent être représentés par des courbes avec le bouton " COURBES ". Les figures 5.14 et 5.16 présente la station "poste d'eau" et "chaudière" (voir les schémas techniques des figures 5.1 et 5.2) ceci permet à l'opérateur de piloter l'opération de génération de vapeur avec l'alimentation d'eau tout en observant les différentes données de régulation (n_c, d_e_b, d_v, n_b,d_a, d_e_c,...).

Ces données peuvent être représentées graphiquement et exprimées en pourcentages par des courbes (figures 5.15 et 5.17).

5.4 Conclusion

La modélisation et la simulation sont des outils très appréciés pour l'apprentissage. Dans le milieu industriel, elles permettent aux nouveaux opérateurs une rapide intégration et une bonne assimilation des procédés qui y sont employés tout en leur évitant de côtoyer directement les processus réels qui présentent parfois des risques et dangers très élevés. L'arrêt de ces machines pour leur observation est souvent pénalisant pour l'usine voire même impossible.

Le système présenté ici nécessite encore un ensemble de travaux et de développement pour aboutir à un système pouvant assurer à la fois un découpage judicieux et une garantie d'extension pour intégrer des composants futurs.

Conclusion générale

Le travail que nous avons présenté se situe dans le cadre de la modélisation et de la simulation des systèmes complexes et particulièrement, les systèmes industriels. Cette thèse combine deux approches de modélisation et de simulation : la première, DEVS, qui est à la fois un outil de modélisation et de simulation, et l'approche méthodique AGR telle que définie dans AALAADIN et validée par la plateforme SMA (MadKit) aboutit à offrir un environnement pour la conception et le développement de cadre de modélisation et de simulation très souple, extensible et facile à mettre en œuvre.

Dans le présent travail, nous avons montré la force du formalisme DEVS pour la représentation et la validation des systèmes complexes d'une part et la puissance des outils évolués des SMA pour l'implémentation de ces systèmes d'autre part.

Nous avons proposé une approche de transformation de modèles DEVS en modèles AGR. Dans laquelle, nous avons spécifié les règles et l'ensemble de fonctions et procédures permettant les transformations systématiques de ces modèles.

Finalement, nous avons soutenu notre approche par un exemple d'application, dans le cas de la régulation d'un processus industriel complexe et nous avons construit un simulateur industriel de ce système.

Ce simulateur peut être considéré comme un assistant de formation et d'apprentissage destiné aux opérateurs novices, travailleurs nouveaux recrutés et stagiaires leur permettant de comprendre et de simuler le fonctionnement du processus industriel sans agir sur le système réel et ne pas causer l'arrêt du procédé qui pourrait engendrer des pertes en matière d'argent et de temps.

Dans de futurs travaux, nos perspectives portent sur la généralisation de l'approche développée dans cette thèse et nous visons à proposer une méthode générale pour construire des simulateurs capables de fonctionner sur plusieurs classes des systèmes industriels et nous envisageons aussi à la mise en application de tels transformations sur d'autre type de systèmes complexes ; particulièrement, à événements discrets.

BIBLIOGRAPHIE

[AgentBuilder R.M., 2000] *An Integrated Toolkit for Constructing Intelligent Sofware Agents*, AgentBuilder, Reference Manual, Avril 2000.

[AgentBuilder U.G., 2000] *An Integrated Toolkit for Constructing Intelligent Sofware Agents*, AgentBuilder, User's Guide, Avril 2000.

[Akehurst *et al.*, 2005] D. H. Akehurst, W. G. Howells, and K. McDonald-Maier. Kent model transformation language. *In Proceedings of Model Transformations in Practice Workshop (MTIP) at MoDELS Conference*, Montego Bay, Jamaica, 2005. URL http://sosym.dcs.kcl.ac.uk/events/mtip05/programme.html.

[Akplogan, 2013] M. Akplogan, *Approche modulaire pour la planification continue 'Application à la conduite des systèmes de culture'*, Thèse Doctorat, Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier), 2013.

[Alanen et Porres, 2004] M. Alanen et I. Porres, "Coral: A Metamodel Kernel for Transformation *Engines". Presented at Second European Workshop on Model Driven Architecture (MDA),* Canterbury, United Kingdom, September 7th-8th, 2004.

[Amelunxen *et al.*, 2006] C. Amelunxen, A. Königs, T. Rötschke et A. Schürr, MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations. *In Proceedings of the 2nd European Conference on Model Driven Architecture - Foundations and Applications, ECMDA-FA*, pages 361-375, Bilbao, Spain, 2006.

[Anderson, 1974] R. Jock Anderson, Simulation: Methodology and application in agricultural economics. *Review of Marketing and Agricultural Economics*, 42(01), March 1974.

[Anglani *et al.*, 2000a] A. Anglani, P. Caricato, A. Grieco, F. Nucci, A. Matta, G. Semeraro et T. Tolio, Evaluation of capacity expansion by means of fuzzy-devs. citeseer.ist.psu.edu/499458.html, *14th European Simulation Multi Conference*. Gent, Belgium, May, 2000.

[Anglani *et al.*, 2000b] P. Anglani, A. Grieco, F. Nicci, Q. Semeraro, et T. Tolio, A New Algorithm to Rank Temporal Fuzzy Sets in Fuzzy Discrete Event Simulation. *In The Ninth IEEE International Conference on Fuzzy Systems. FUZZ IEEE 2000*. San Antonio, TX, 2000.

[Atigui, 2013] F. Atigui, *Approche dirigée par les modèles pour l'implantation et la réduction d'entrepôts de données*. Thèse doctorat, *Université Toulouse 1 Capitole (UT1 Capitole)*, 2013.

[Azarmi et Thompson, 2000] N. Azarmi et S. Thompson, *Zeus:*A toolkit for building multi-agent systems, *Proceedings of fifth annual Embracing Complexity conference*, Paris, 2000.

[Barros, 1995] F. J. Barros, Dynamic structure discrete event system specification: a new formalism for dynamic structure modeling and simulation. Dans WSC '95:*Proceedings of the 27th conference on Winter simulation*, pages 781–785, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-7803-3018-8. (Cité pages 193 et 194.)

[Barros, 1996]F. J. Barros, Dynamic Structure Discret Event System Specification: Formalism, *Abstract Simulators and Applications*. 13(1):35–46, 1996. (Cité page 193.)

[Barros, 1997] F. J. Barros, Modeling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 7:501-515, October, 1997. (Cité pages 80, 193, 194 et 195.)

[Barros, 1998] F. J. Barros, Abstract simulators for the dsdevs formalism. *Dans Proceedings of the 30th conference on Winter simulation, WSC '98*, pages 407–412, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press. ISBN 0-7803-5134-7. (Cité pages 194 et 228.)

[Barros, 2003]F. J. Barros. Dynamic Structure Multiparadigm Modeling and Simulation. *ACM Transactions on Modeling and Computer Simulation*, 13(3) :259–275, 2003. (Citépage 193.)

[Bellifemine *et al*, 1999] F. Bellifemine, A. Poggi et G. Rimassa, JADE - A FIPA - *compliant agent framework*, *PMAA`99*, p. 97-108, Londre, Avril, 1999.

[Bellifemine *et al*, 2002a] F. Bellifemine, G. Caire, T. Trucco, G. Rimassa, *JADE*, Administrator's Guide, Janvier, 2002.

[Bellifemine *et al*, 2002b] F. Bellifemine, G. Caire, T. Trucco, G. Rimassa, *JADE*, Programmer's Guide, Février, 2002.

[Bellefemine *et al*, 2004] F. Bellefemine, G. Caire, T. Trucco et G. Rimassa, « *JADE* programmer's guide (JADE 3.2) ». Rapport, TILAB S.p.A, 2004.

[Bernon *et al.*, 2005] C. Bernon, M. Cossentino, et J. Pavon, *An Overview of Current Trends in European AOSE Research*. Informatica (Slovenia), pages 379–390, 2005.

[Bernon et al., 2009] Carole Bernon, Marie-Pierre Gleizes et Gauthier Picard, Chapitre 2 : Méthodes orientées agent et multi-agent, 2009. http://www.emse.fr/~picard/publications/bernon09industrie.pdf

[Bertalanffy, 1968] L. V. Bertalanffy, *Théorie générale des systèmes*. Dunod, 1968.

[Bézivin et Gerbé, 2001] J. Bézivin et O. Gerbé, Towards a Precise Definition of the OMG/MDA Framework. *In Proceedings of the 16th International Conference on Automated Software Engineering, ASE*, pages 273-280, Washington, DC, USA, 2001.

[Bézivin, 2004] J. Bézivin, *Sur les principes de base de l'ingénierie des modèles*. RSTI-L'Objet, 10(4):145–157, 2004. (Cité pages 28, 32, et 41.)

[Bézivin, 2005] J. Bézivin, On the unification power of models. *Software and System Modeling (SoSym)*, 4(2):171–188, 2005. (Cité page 28.)

[Bézivin *et al.*, 2008] F. Jouault, F. Allilaire, J. Bézivin et I. Kurtev, "Atl: A model transformation tool", *Science of Computer Programming*, vol. 72, no. 1-2, pp. 31-39, Juin 2008.

[Bisgambiglia, 2008] P.A. Bisgambiglia, *Approche de modélisation approximative pour des systèmes à événements discrets 'Application à l'étude de propagation de feux de forêt'*, Thèse Doctorat, Université de Corse – Pasquale Paoli (UFR Sciences et Techniques), 2008.

[Bohlen, 2007] M. Bohlen, "*Andro MDA Model Driven Architecture Framework*", http://galaxy.andromda.org/docs-3.2/, 2007.

[Boissier et Demazeau, 1996] O. Boissier et Y. Demazeau, ASIC : An Architecture for Social and Individual Control and its Application to Computer Vision. *Dans* J.W. PERRAM et J.-P. MÜLLER, éditeurs : *Distributed Software Agents and Applications, 6th European Workshop on Modelling Autonomous Agents, MAAMAW '94, Odense, Denmark, August 3-5, 1994, Proceedings*, volume 1069de *Lecture Notes in Computer Science (LNCS)*, pages 135–149. Springer-Verlag, 1996.

[Boissier *et al.*, 1998] O. Boissier, P. Beaune, H. Proton, M. Hannoun, T. Carron, L. Vercouter, et C. Sayettat, *The Multi-Agent System Toolkit*. Technical report, SIC/ENSM-SE, 1998.

[Boissier *et al.*, 2004] O. Boissier, S. Gitton, P. Glize, Caractéristiques des Systèmes et des Applications. Systèmes Multi-Agents, *Observatoire Français des Techniques Avancées, ARAGO 29*,Diffusion Editions TEC & DOC, p. 25-54, 2004.

[Bolduc et Vangheluwe, 2002] J. S. Bolduc et H. Vangheluwe, *A modeling and simulation package for classic hierarchical DEVS*. Technical report, 2002.

[Bond, 1990] A. H. Bond, « Distributed Decision Making in Organization ». *IEEE Transactions on systems*, man and cybernetics Conference, November 1990.

[Bondé, 2006] L. Bondé, *Transformations de Modèles et Interopérabilité dans la Conception de Systèmes Hétérogènes sur Puce à Base d'IP*, Thèse doctorat, Université des Sciences et Technologies de Lille, 2006.

[Booch *et al.*, 1997] G. Booch, J. Rumbaugh et I. Jacobson, *Unified Modelling Language* User Guide. ed. Addison Wesley, 1997.

[Borland et Vangheluwe, 2003] S. Borland et H. Vangheluwe, Transforming state charts to devs. *Dans Summer Computer Simulation Conference, Student Workshop - Society for Computer Simulation International*, pages 154 – 159, Montréal, Canada. A. Bruzzone and Mhamed Itmi editors, 2003.

[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.

[Bradshaw *et al.*, 1996] J. M. Bradshaw, S. Dutfield, P. Benoit et J. D. Woolley, *KAoS: Toward an Industrial-Strength Open Agent Architecture*, G. M. O'Hare, N. R. Jennings, eds.Foundations of Distributed Artificial Intelligence, John Wiley & Sons, p. 375 – 418, 1996.

[Brazier *et al.*, 1998] F. Brazier, C.M. Jonker et J. Treur, Principles of Compositional Multi-Agent System Development. *Dans J. CUENA, éditeur : Proceeding of the IFIP'98 Conference IT & KNOWS'98*. Chapman & Hall, 1998.

[Bresciani *et al.*, 2004] P. Bresciani, A. Perini, P. Giorgini, F. Guinchiglia et J. Mylopoulos , Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Sytems*, 8, 203–236, 2004

[Brooks et Connell, 1986] R. Brooks et J.H. Connell, Asynchronous distributed control system for a mobile robot. Pages 77–84 of :Wolfe,W., & Marquina, N. (eds), *SPIE's Cambridge Symposium on Optical and Opto-Elecronic Engineering*, vol. 727. October, 1986.

[Brooks, 1989] R. Brooks, A robot that walks : emergent behaviour from a carefully evolved network, *Neural computation*, 1:253-62, 1989.

[Busetta et Ramamohanarao, 1998] P. Busetta et K. Ramamohanarao "*The BDIM Agent Toolkit Design*"., Department of Computer Sciences, Victoria. Australia. 1998.

[Busetta et *al.*, 1999], P. Busetta, R. Ronnquist, A. Hodgson, A. Lucas, *Jack intelligent agents - components for intelligent agents in java*, Agent Link News Letter, Janvier, 1999.

[Caire *et al.*, 2001] G. Caire,W. Coulier, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans et P. Massonet, Agent Oriented Analysis Using Message/UML. *Dans* M. WOOLDRIDGE, G. WEISS et P. CIANCARINI, éditeurs : *Agent-Oriented Software Engineering II, Second International Workshop, AOSE 2001*, volume 2222 de *Lecture Notes in Computer Science (LNCS)*, pages 119–135. Springer-Verlag, 2001.de l'art, 2008. URL http://hal.archives-ouvertes.fr/ hal-00371565.

[Campagne, 2005] J.C. Campagne, *Systèmes multi-agents et morphologie*, Thèse de doctorat, Université de Paris VI, 2005.

[Chaib-draa, 1994] B. Chaib-draa, « Distributed Artificial Intelligence : an overview ». In *Encyclopedia of Computer Science and Techonlogy*, A. KEN, J. WILLIAMS, C. HALL etR. KENT (Ed.), pp. 215-243. 1994.

[Chaib-draa, 1995] B. Chaib-draa, «Industrial applications of distributed AI». *Communications of the ACM*, 38(11), pp. 49-53. 1995.

[Chaib-draa, 1996] B. Chaib-draa, « Interaction between agents in routine, familiar and unfamiliar situations ». *International Journal of Experimental and Theorical AI (JETAI)*, 1(5), pp. 7-20. 1996.

[Chaib-draa *et al.*, 2001] B. Chaib-draa, I. Jarras et B. Moulin, « *Systèmes multiagents : Principes généraux et applications* », Editions J.P. Briot, Y. Demazeau, Agent et systèmes multiagents, Hermes, 2001.

[Chaib-draa et Demolombe, 2002] B. Chaib-draa et R. Demolombe, *Dans : Information-Interaction- Intelligence*, 2002.

[Chaib-draa et Jarras, 2002] B. Chaib-draa et I. Jarras, «*Aperçu sur les systèmes multi agents*»,Série Scientifique, Montréal, 2002.

[Chauhan, 1997] D. Chauhan, JAFMAS: *A Java-Based Agent Framework for Multi-Agent Systems Development and Implementation*, Masters Thesis, ECECS Department, University of Cincinnati, Juillet 1997.

[Chow et Zeigler, 1994] A.C.H. Chow et B.P. Zeigler. Parallel DEVS : a parallel, hierarchical, modular, modeling formalism. Dans *Proceedings of the 26th conference on Winter simulation*, pages 716–722, Orlando, Florida, United States, 1994. (Cité pages 67 et 191.)

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

[Clavel *et al.*, 1997] G. Clavel, N. Lopez, et L. Veillon, *Programmer objets avec Smalltalk. Dunod*. ISBN 2225851573, 1997.

[Codagen, 2004] Codagen. *Codagen Architect*. Disponible surhttp://www.codagen.com/products/architect/,december 2004.

[Coleman *et al.*, 1994] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, et P. Jeremaes, « *Object-Oriented Development: The Fusion Method* », Prentice Hall, 1994.

[Collinot et Drogoul, 1996] A. Collinot et A. Drogoul, Agent Oriented Design of a Soccer Robot Team. *Dans*. TOKORO, éditeur : *Second International Conference on Multi-Agent Systems (ICMAS'96), Nara, Japan*, pages 41–47. AAAI Press, 1996.

[Collinot et Drogoul, 1998] A. Collinot et A. Drogoul, La méthode de conception multi-agent CASSIOPEE : application à la robotique collective. *Revue d'intelligence artificielle*, 12(1):125–147, 1998.

[Collis *et al.*, 1998] J. C. Collis, H. Nwana, D. Ndumu et L. C. Lee, *Zeus: A* collaborative agents toolkit, A collaborative agents toolkit. *In Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, p. 377-392, 1998.

[Collis et Lee, 1998] J. C. Collis et L. C. Lee, *Building Electronic Marketplaces with the {ZEUS} Agent Tool-kit*, AMET, p. 1-24, 1998.

[Collis *et al.*, 1999] J. C. Collis, L. C. Lee, D. T. Ndumu et H. S. Nwana, *ZEUS: a toolkit and approach for building distributed multi-agent systems*, ACM Press, p. 360-361, 1999.

[Collis et Ndumu, 1999a] J. Collis et D. Ndumu, *The Zeus Agent Building Toolkit*, The Role modelling Guide, Août 1999.

[Collis et Ndumu, 1999b] J. Collis et D. Ndumu, *The Zeus Agent Building Toolkit*, The Application Realisation Guide, Septembre 1999.

[Collis et Ndumu, 1999c] J. Collis, D. Ndumu, *The Zeus Agent Building Toolkit*, The Runtime Guide, Novembre 1999.

[Collis et Ndumu, 1999d] J. Collis, D. Ndumu, *The Zeus Agent Building Toolkit*, Technical Manual, Septembre 1999.

[Combemale *et al.*, 2008] B. Combemale, *Ingénierie Dirigée par les Modèles (IDM)* ,2008.

[CORMAS, 2001] *Présentation de Cormas*, Disponible sur http://cormas.cirad.fr/fr/outil/present.htm, 2001.

[Cossentino *et al.*, 2009] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, et A. Koukam, ASPECS : an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 2009.

[Cossentino, 2001] M. Cossentino, Different Perspectives in Designing Multi-Agent System. *Dans Designing Multi-Agent System, AgeS'02 (Agent Technology and Software Engineering) Workshop at NodE' 02*, 2001.

[Cossentino et Potts, 2002] M. Cossentino et C. Potts, A CASE tool supported methodology for the design of multiagent systems. *Dans The Proceedings of the International Conference on Software Engineering Research and Practice (SERP'02)*, 2002.

[Cossentino *et al.*, 2005] M. Cossentino, S. Gaglio, L. Sabatucci, et V. Seidita. « The PASSI and Agile PASSI MAS Meta-Models Compared with a Unifying Proposal». *Dans: Proceedings of the CEEMAS' 05 Conference*, pages 183–192., Budapest, Hungary, septembre 2005.

[Coulier *et al.*, 2004] W. Coulier, F. Garijo, J. Gomez, J. Pavon, P. Kearney et P. Massonet, *MESSAGE : a methodology for the development of agent-based applications*. Kluwer, 2004.

[Czarnecki et Helsen, 2003]K. Czarnecki et S. Helsen, Classification of Model Transformation Approaches. *In Proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA*, Anaheim, CA, USA. 2003.

[Da Silva et Demazeau, 2002] J-L. T. da Silva et Y. Demazeau, Vowels Co-ordination Model. *Dans* M. GINI, T. ISHIDA, C. CASTELFRANCHI et W.L. JOHNSON, éditeurs : *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 1129–1136. ACM Press, 2002.

[Davilla et Uzcategui, 2000] J. Davilla et M. Uzcategui, GALATEA: A multi_agent simulation platform, *in international conference on modelling, simulation and neural networks MSNN'200*0, Mérida Venezuela, 2000.

[Davilla *et al.*, 2005] J. Davilla, E. Gomez, K. Lafaille, K. Tucci et M. Uzcategui, Multi Agent Distributed Simulation with GALATEA ;*Proceedings of the 2005 Ninth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'05)*, 2005.

[Degirmenciyan et Marc, 2002] I. Degirmenciyan et F. Marc, SCALA une approche multi-agents pour la conception de systèmes complexes application à la simulation de missions aériennes, *JFIADSMA 2002*, Lille, France.

[De Lara et Vangheluwe, 2002] J. De Lara et H. Vangheluwe, AToM3: A tool for multi-formalism and meta-modelling. *Dans Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering table of contents.* Springer-Verlag, London, UK, 2002.

[De Lara et Vangheluwe, 2002] J. De Lara and Hans Vangheluwe. Atom3 : A tool for multi-formalism and meta-modelling. *In Proceedings of the 5th International Conference on*

Funda- mental Approaches to Software Engineering, FASE, pages 174-188, London, UK, 2002.

[Del Fabro *et al.*, 2005] M.D. Del Fabro, J. Bézivin, F. Jouault, E. Breton et G. Gueltas, AMW : a generic model weaver. In *Actes des 1ères Journée sur l'Ingénierie Dirigée par les Modèles*, IDM, 2005.

[Deloach, 1999] S.A. Deloach, Multiagent Systems Engineering : a Methodology and Language for Designing Agent Systems. *Dans Proceedings of Agent Oriented Information Systems '99 (AOIS'99)*, pages 45–57, 1999.

[Deloach et Wood, 2000] S.A. Deloach et M.F. Wood : Developping Multi Agent Systems with agentTool. *Dans* C. CASTELFRANCHI et Y. LESPERANCE, éditeurs : *The Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, volume 1986 de *Lecture Notes in Computer Science (LNCS)*, pages 46–60. Springer-Verlag, 2000.

[Deloach *et al.*, 2001] S.A. Deloach, M.F. Wood et C.H. Sparkman, Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[Deloach et Wood, 2001] S. A. DeLoach, M. Wood, Developing Multiagent Systems with agentTool, *(ATAL'2000)*, Berlin, 2001.

[Deloach, 2002] S. Deloach, « Modeling organizational rules in the multiagent systems methodology ». *Dans: Proceedings of the 15th Canadian Conference on Artificial Intelligence Calgary*, Alberta, Canada, May 27-29, 2002.

[Demazeau, 1995] Y. Demazeau, « From Interactions to Collective Behaviour in Agent-Based Systems ». *Dans: Proceedings of the 1st European Conference on Cognitive Science*. Saint-Malo, France, 1995.

[Demazeau et Costa, 1996] Y. Demazeau et A. R. Costa," Populations and organisations in open multi-agent systems". *In 1st Symposium on Parallel and Distributed AI, Hyderabad*, India, 1996.

[Demazeau, 2001a] Y. Demazeau, *« voyelles », rapport d'habilitaion à diriger des recherches*, Tech. report, Institut National Polytechnique de Grenoble, Laboratoire Leibniz, Grenoble, Avril 2001a.

[Demazeau, 2001b] Y. Demazeau, M*AS Methodology. Présentation lors du Séminaire IRIT - Cycle Systèmes Multi-Agents*, Institut de Recherche en Informatique de Toulouse - Université Paul Sabatier, 15 Novembre 2001b.

[Demazeau, 2003] Y. Demazeau, Créativité émergente centrée utilisateur. *Dans* J.P. BRIOT et G. KHALED, éditeurs : *Déploiement des systèmes multi-agents – Vers un passage à l'échelle (Journées Francophonessur les Systèmes Multi-Agents, JFSMA'03)*, pages 31–36. Hermès - Lavoisier (Revue RSTI Hors-série), 2003.

[Deneubourg *et al.*, 1991] J-L. Deneubourg, S. Goss, N.R. Francks, A. Sendova-Franks, C. Detrain et L. Chretien, The dynamics of collective sorting : robot-like ants and ant-like robots, *in Meyer J-A et Wilson S. Eds, Simulation of Adaptive Behaviour : from animals to animats*, Cambridge, Mass. : MIT Press, 356-65, 1991.

[Dieng, 1990] R. Dieng, Relations Linking Cooperating Agents. *In Proceedings of the 2nd European Workshop MAAMAW'90*, pages 185-202, Saint-Quentin en Yvelines-France, August 1990.

[D'Iverno *et al.*, 1997] M. d'Inverno, D. Kinny, M. Luck et M. Wooldridge, *A Formal Specification of dMARS*, Agent Theories, Architectures, and Languages, p. 155-176, 1997.

[Djeddai, 2013] S. Djeddai. *Combining Formal Verification Environments and Model-Driven Engineering*. PhD thesis, University Paul Sabatier, Toulouse, 2013.

[Drogoul, 1993] A. Drogoul, *De la simulation multi-agent à la résolution collective de problèmes*, Thèse de l'Université P et M Curie, 1993.

[Drogoul, 2000] A. Drogoul, *Systèmes multi-agents situés*. Mémoire d'habilitation à diriger des recherches, Université Paris 6, 2000.

[Dumoulin, 2004] C. Dumoulin, ModTransf : A *model to model transformation engine*,. http://www.lifl.fr/west/modtransf, November, 2004.

[Durfee et Lesser, 1989] E. H. Durfee et V. R. Lesser, « Negotiating task decomposition and allocation using partial global planning ». *Dans: Distributed Artificial Intelligence*. Volume II, pages 229-244. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[Edmonds et Hales, 2003] B. Edmonds et D. Hales, Replication, replication and replication : Some hardlessons from model alignment. *Artificial Societies and Social Simulation*, 6(4), 2003.

[Edmonds, 2005] B. Edmonds, Simulation and complexity - how they can relate. *In Virtual Worlds of Precision - computer-based simulations in the sciences and social sciences*, pages 5–32. *LitVerlag, feldmann,* valerie and mühlfeld, katrin edition, 2005.

[Erceau, 1993] J. Erceau, Intelligence Artificielle Distribuée et Systèmes Multi-Agents - de la théorie aux applications. *23ème Ecole Internationale d'Informatique de l'AFCET*, Neuchâtel, 1993.

[EURESCOM, 2000] EURESCOM, MESSAGE : *Methodology for Engineering Systems of Software Agents Deliverable 1* - Initial Methodology. Project P907-GI, EURESCOM, 2000.

[Ferber, 1995] J.Ferber, Les *systèmes multi-agents : vers une intelligence collective*, InterEditions, ISBN : 2- 72-96-0572-X, 1995.

[Ferber, 1997] J. Ferber , Les Systèmes Multi-Agents : Un Aperçu Général, *Revue Technique et Science Informatiques*, Hermes-Lavoisier, 1997.

[Ferber et Gutknecht, 1998] J. Ferber et O. Gutknecht, *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*. Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), IEEE CS Press 128-135. 1998.

[Ferber, 1999] J. Ferber, *Multi-Agent Systems - An introduction to distributed artificial intelligence*, Addison- Wesley, ISBN 0-201-36048-9, 1999.

[Ferber et Gutknecht, 1999] J. Ferber et O. Gutknecht, *Operational semantics of a role-based agent architecture*, ATAL, 1999.

[Ferber *et al.*, 2003] J. Ferber, O. Gutknecht et F. Michel, From agents to organizations : An organizational view of multi-agent systems. *In Paolo Giorgini, Jörg P. Müller, and James Odell, editors, AOSE*, volume 2935 of Lecture Notes in Computer Science, pages 214–230. Springer, 2003.

[Filippi *et al.*, 2002a] J.B. Filippi, F. Bernardi et M. Delhom, The JDEVS environmental modeling and simulation environment. *IEMSS, Integrated Assessment and Decision Support*, Lugano Suisse, pages 283–288, 2002.

[Filippi *et al.*, 2002b] J.B. Filippi, P. Bisgambiglia et M. Delhom, Neuro-devs, an hybrid methodology to describe complex systems. *Dans Processings of the Society for Computer Simulation International European Symposium on Simulation 2002 conference on simulation in industry*, volume 1, 2002.

[Filippi, 2003] J.B. Filippi, *Une architecture logicielle pour la multi-modélisation et la simulation à évènements discrets de systèmes naturels complexes.* Thèse de doctorat, Université de Corse, 2003.

[Filippi et Bisgambiglia, 2004] J.B. Filippi et P. Bisgambiglia, JDEVS: *An implementation of a DEVS based formal framework for environmental modelling*.http://www citeseer.ist.psu.edu/filippi03jdevs.html, 2003.

[Fini et Wiederhold, 1993] T. Fini, G. Wiederhold, "A*n Overview of KQML: A knowledge Query and Manipulation Language*", Department of Computer Science, Stanford University,
1993.

[Fini *et al.*, 1994a] T. Fini, R. Fritzon, D. Mckay et R. Mc Entire, "KQML as an agent communication language", *In Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*, ACM Press, 1994.

[Fini *et al.*, 1994b] T. Fini, R. Fritzon, D. McKay et R. McEntire, "*KQML A Language and Protocol for Knowledge and Information Exchange*", (Technical Report No.CS-94-02). University of Maryland, Departmentof Computer Science, 1994.

[Finin *et al.*, 1995] T. Finin, Y. Labrou, et J. Mayfield, KQML as an agent communication language. *Jeff Bradshaw (Ed.), "Software Agents", MIT Press, Cambridge*, 1995.

[FIPA , 1999] FIPA, FIPA Specification. rapport, 1999, *FIPA Association*, 1999.

[Fishwick, 1994] P.A. Fishwick, Simulation model design. *In Proceedings of the 26th conference on Winter simulation (WSC '94),* pages 173–175, San Diego, CA, USA, 1994.

[Fishwick, 1995] P.A. Fishwick, Simulation Model Design and Exe*cution. Digital Worlds*, Prentice Hall, 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.

[Fox, 1981] M.S. Fox, An *organizational view of distributed systems*. IEEE Trans. Syst.Man. Univ. Cybern., vol. SMC-11; 1981, pp. 70-80, 1996.

[Freigassner *et al.*, 2000] R. Freigassner, H. Praehofer et B. P. Zeigler, Systems approach to validation of simulation models. *Cybernetics and Systems*, pp. 52–57, 2000.

[Frontier et Pinchod-Viale, 1995] S. Frontier et D. Pinchod-Viale, *Ecosystèmes. Structure-fonctionnement évolution*. Masson, 1995.

[Galán *et al.*, 2009] J. M. Galán, L. R. Izquierdo, S. S. Izquierdo, J. I. Santos, R. del Olmo, A. López-Paredes, et B. Edmonds, Errors and artefacts inagent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1):1, 2009.

[Gallone *et al.*, 1994] J-M. Gallone, F. Charpillet et V. Chevrier, Un modèle d'agent pour un raisonnement contraint par le temps, - *In: Actes Journée PRC-IA sur les Systèmes Multi-Agents*, Paris, Décembre 1994.

[Garredu, 2013] S. Garredu, *Approche de méta-modélisation et transformations de modèles dans le contexte de la modélisation et simulation à évènements discrets 'application au formalisme DEVS',* Thèse de Doctorat, Université de Corse – Pasquale Paoli, 2013.

[Garneau et Delisle, 2002] T. Garneau. S. Delisle. *Programmation orientée agent : évaluation comparative d'outils et environnements*, Actes des Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'02), Lille(France), Octobre 28-30 2002. *Systèmes multi-agents et systèmes complexes (ingénierie, résolution de problèmes et simulation)* JFIADSMA'02, Hermes Sciences, 2002, 111-123.

[Gasser *et al.*, 1987] L. Gasser, C.sd Braganza et N. Herman, *MACE: A flexible tested for distributed AI*, Pitman, 1987.

[Geiger *et al.*, 2006] L. Geiger et A. Zündorf, "Tool modeling with fujaba", *Electronic Notes in Theoretical Computer Science*, vol. 148, no. 1, pp. 173-186, février 2006.

[Giambiasi, 2001] N. Giambiasi, *Cours DEA MCAO* Université Aix Marseille III, 2001.

[Giambiasi et Ghosh, 2001] N. Giambiasi et S. Ghosh, Min-Max-DEVS : A new formalism for the specification of discrete event models with min-max delays. pages 616–621. 13th *European Simulation Symposium*, 2001.

[Giese *et al.*, 2009] H. Giese et R. Wagner, "From model transformation to incremental bidirectional model synchronization", *Software and Systems Modeling*, vol. 8, no. 1, pp. 21-43, Février 2009.

[Gilbert et Troitzsch, 2005] N. Gilbert et K. G. Troitzsch, *Simulation for the Social Scientist*. Open University Press, Maidenhead and New York, 2 edition, 2005.

[Ginot and Le Page, 1998] V. Ginot et C. Le Page, Mobidyc, a generic multi-agent simulator for modeling communities dynamics. *In IEA-98-AIE*, volume 1416 of LNAI, pages 805–814. Springer, 1998.

[Ginot *et al.*, 2002] V. Ginot, C. Le Page et S. Souissi. A multi-agent architecture to enhance end-user individual-based modelling. *Ecological Modeling*, 157 :23–41, 2002.

[Glinsky et Wainer, 2002] E. Glinsky et G. Wainer, Definition of Real Time simulation in CD++ toolkit. *Dans Proceedings of the 2002 Summer Computer Simulation Conference*, San Diego, USA, 2002.

[Gomez Sanz et Fuentes, 2002] J. Gomez Sanz et R. Fuentes, Agent oriented system engineering with ingenias. *Dans Fourth Iberoamerican Workshop on Multi-Agent Systems, Iberagents'02*, 2002.

[Griffin, 2004] C. Griffin, Model Transformtion Framework (MTF). *IBM Hursley*, available from IBM Alphaworks, 2004.

[Grimm, 1999] V. Grimm, Ten years of individual-based modelling in ecology: what we have learned and what could we learn in the futur. *Ecological Modelling*,115:129–148, 1999.

[Guerra et de Lara, 2004] E. Guerra et J. de Lara, "*Event-driven grammars: Towards the integration of meta-modelling and graph transformation*", pp. 54-69, 2004.

[Gutknecht et Ferber, 1998a] O. Gutknecht et J. Ferber, *A model for social structures in multiagent systems*, Tech. report, RR.LIRMM 98040, 1998.

[Gutknecht et Ferber, 1998b] O. Gutknecht et J. Ferber , Un modèle d'analyse, de construction et d'exécution de systèmes multi-agents, *Actes des JFIADSMA'98* (Editions Hermès,ed.), 1998.

[Gutknecht et Ferber, 1999] O. Gutknecht et J. Ferber, Vers une méthodologie organisationnelle de conception. *Dans* M.-P. GLEIZES et P. MARCENAC, éditeurs : *Ingénierie des systèmes multi-agents – Actes des 7èmes JFIADSMA*, pages 93–104. Hermès, 1999.

[Gutknecht et Ferber, 2000] O. Gutknecht et J. Ferber, The MADKIT agent platform architecture, *Agents workshop on infrastructure for multiagent systems*, pp. 48–55, 2000.

[Gutknecht, 2001] O. Gutknecht, *Proposition d'un modèle organisationnel générique de systèmes multi-agents. examen de ses conséquences formelles, implémantatoires et méthodologiques*, Ph.D. thesis, Université Montpellier II, 2001.

[Hall et Fagen, 1956] A. Hall, et R. Fagen, *Yearbook of the Society for the Advancement of General Systems Theory*. volume General Systems I, chapter Definition of System. Ann Arbor, 1956.

[Hammoudi, 2010] S. Hammoudi, *Contribution à l'étude et à l'application de l'ingénierie dirigée par les modèles*, Habilitation à diriger des recherches (HDR), Université d'Angers, 2010.

[Hamri *et al.*, 2006] A. Hamri, N. Giambiasi et C. Frydman, Min-Max DEVS modeling and simulation. *Simulation Modelling Practice and Theory (SIMPAT)*, 14(7) :909–929, October . Ed. Elsevier, ISSN 1569–190X, 2006.

[Hild, 2000] D.R. Hild, *Discrete Event System Specification Distributed Object Computing Modeling and Simulation*. PhD thesis, 2000.

[Hill, 1996] D.R.C. Hill, *Object-Oriented Analysis and Simulation*. - Addison-Wesley, 1996.

[Himmelspach *et al.*, 2010] J. Himmelspach, M. Röhl, et A. M. Uhrmacher, Component based modelling and simulation for valid multi-agent system simulations. *International journal for Applied Artificial Intelligence*, 24(5):414–442, May 2010.

[Himmelspach, 2012] J. Himmelspach, JAMES II: Extending, Using, and Expeiments, *in: Proceedings of the 5thSimutools Conference ICST*, pp 208-210, 2012.

[Hubert, 1969] P. Hubert, *Le Glossaire International d'Hydrologie*, novembre, 1996.

[Hubert, 2005] K. Hubert, *Conception orienté objet guidée par les modèles*. Dunod, 2005.

[Hübner *et al.*, 2007] J. F. Hübner, J. S. Sichman et O. Boissier, Developing organised multiagent systems using the MOISE. *IJAOSE*, 1(3/4):370–395, 2007. URL http://dx.doi.org/10.1504/IJAOSE. 2007.016266.

[Doudoux, 2014] J.M. Doudoux, Développons en Java, *JDOM Copyright (C) 1999-2014*, 2014.

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

[IOPT] Interactives Objects and Project Technology, Mofquery/views/transformations, revised submission. *omg document* : ad/03-08-11, ad/03-08-12, ad/03-08-13. http: //frontline.compuware.com/javacentral/tools/.

[Jack, 2001a] *Jack Development Environment*, User Guide, 2001.

[Jack, 2001b] *Jack Intelligent Agents*, User Guide, 2001.

[Jacobson *et al.*, 1997] I. Jacobson, G. Booch et J. Rumbaugh, The Objectory Software Development Process. ed. Addison Wesley.copépode. *Dans Colloque Joint Conference on Multi-Agent Modelling for Environmental Management*, Cemagref, pages 35–49, Clermont-ferrand, France, 1997.

[Jacobson *et al.*, 1999] I. Jacobson, G. Booch et J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley Longman Publishing, 1999.

[Jacques et Wainer, 2002] C. Jacques et G.A. Wainer, Using the cd++ DEVS tookit to develop petrinets. *In SCS, editor, Proceedings of the SCS Conference*, 2002.

[Jamda] SourceFORGE.net. *Jamda*, Disponible sur http://jamda.sourceforge.net/, december 2004.

[Jarras et Chaib-draa, 2002] I. Jarras et B. Chaib-draa, « *Aperçu sur les systèmes multiagents* », Série scientifique, Centre interuniversitaire de recherche en analyse des organisations (CIRNO), Université de Montréal, 2002.

[Jean-Pierre, 2003] G. Jean-Pierre, The amas theory for complex problem solving based on self-organizing cooperative agents, *First european workshop on multi-agent systems* (Oxford, UK), vol. 1, 2003.

[Jennings, 1996] N. R. Jennings, « Coordination Techniques for Distributed Artificial Intelligence». *Foundations of Distributed Artificial Intelligence*, Editions G. M. P. O'Hare et N. R. Jennings, Wiley, 1996.

[Jennings *et al.*, 1998] N. R. Jennings, K. Sycara et M. Wooldridge, A Roadmap of Agent Research and Development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1998.

[Jennings et Wooldridge, 1998] N. R. Jennings et M. Wooldridge, "Applications of Intelligent Agents". Dans: *Agent Technology: Foundations, Applications, and Markets (eds. N. R. Jennings and M. Wooldridge) Springer-Verlag*: Berlin, Germany. pp. 3-28, 1998.

[Jeennings et Wooldridge, 2000] N. R. Jeennings et M. Wooldridge, «Agent-Oriented Software Engineering » *in Handbook of Technology (ed. J. Bradshaw) AAAI/MIT Press*, 2000.

[JET] *Java Emitter Templates* (*JET*). Disponible sur http://www.eclipse.com/emf, december 2004.

[Jézéquel, 2006] J.M. Jézéquel, *L'ingénierie des modèles*, Archive ouverte HAL-INRIA, Disponible sur : https://hal.inria.fr/inria-00512547, 2006.

[Jouault, 2006] F. Jouault, *Contribution à l étude des langages de transformation de* modèles. Thèse de doctorat, Université de Nantes, 2006.

[Jouault et Kurtev, 2006] F. Jouault et I. Kurtev, Transforming models with ATL. *In International Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005)*, MTiP, pages 128-138, Montego Bay, Jamaica, 2006.

[Kay, 2001] M. H. Kay, "*XSLT Programmer's Reference*", 2nd Edition, Peer Information, Avril 2001.

[Kendall *et al., 2000*] E. A. Kendall, P. V. Krishna, C. V. Pathak et C. B. Suresh, *A Java Application Framework for Agent Based Systems*, Computer Systems Engineering Royal Melbourne Institute of Technology, Melbourne, Australia,2000.

[Kermeta] *Kermeta*, http://www.kermeta.org/.

[Kinny *et al.*, 1996] D. Kinny, M. Georgeff et A. Rao, A Methodology and Modelling Technique for Systems of BDI agents. *Dans* W. Van de VELDE et J. W. PERRAM, éditeurs : *Agents Breaking Away : Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi Agent World*, volume 1038 de *Lecture Notes in Artificial Intelligence (LNAI)*, pages 51–71. Springer-Verlag, 1996.

[Kleppe *et al.*, 2003] A. G. Kleppe, J. Warmer et W. Bast, MDA Explained: The Mo-del Driven Architecture: *Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003. ISBN 032119442X.

[Klusch, 1998] M. Klusch, *Intelligent Information agents: Agent-based information discovery and management on the internet*, Springer preface, 1998.

[Kuhn, 1972] T. Kuhn, *La Structure des révolutions scientifiques*. - Flammarion, 1972, nouvelle traduction par Laure Meyer, 1983.

[Kofman *et al.*, 2000] E. Kofman, N. Giambiasi et S. Junco, Fdevs: A general DEVS based Formalism for fault modeling and simulation. *European Simulation Symposium*, volume 1 Hamburg, pages 77–82, 2000.

[Kubera *et al.*, 2008] Y. Kubera, P. Mathieu et S. Picault, Interaction oriented agent simulations: From theory to implementation. *In ECAI 2008: Proceedings, 18th European Conference on Artificial Intelligence*, pages 383–387. IOS Press, July 21-25, 2008.

[Kubera , 2010] Y. Kubera, *Simulations orientées-interaction des systèmes complexes*, Thèse doctorat, Université Lille 1 - Sciences et Technologies, 2010.

[Kwon *et al.*, 1996] Y. Kwon, H. Park, S. Jung, et T. Kim, Fuzzy-DEVS Formalisme: Concepts, *Realization and Application*. *Proceedings AIS 1996*, pages 227.234. 1996.

[Labidi et Lejouad, 1993] S. Labidi et W. Lejouad, « *De l'Intelligence Artificielle Distribuée aux Systèmes Multi-Agents* », Rapport de recherche n°2004 , 39 pages, 1993.

[Law et McComas, 1991] A.M. Law et M.G. McComas, Secrets of successful simulation studies. *In Simulation Conference, 1991. Proceedings.*, Winter, pages 21–27, Dec 1991.

[Law et Kelton, 2000] A. M. Law, et W. D. Kelton, *Simulation Modeling and Analysis*. McGrawHill, 3rd edition, 2000.

[Lawley et Steel, 2005] M. Lawley et J. Steel, Practical declarative model transformation with Tefkat. *In Proceedings of Model Transformations in Practice Workshop (MTIP) at MoDELS Conference*, Montego Bay, Jamaica, 2005.

[Luke *et al.*, 2005] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan et G. Balan, MASON : A Multi agent Simulation Environment. Simulation, 81(7) :517–527, 2005.

[Lind, 2001] J. Lind, *Iterative Software Engineering for Multiagent Systems : the MASSIVE Method*, volume 1994 de *Lecture Notes in Artificial Intelligence (LNAI)*. Springer Verlag, 2001.

[Maes, 1995] P. Maes, Intelligent Software, *Scientific American*, volume 273, numéro3, pp. 84-86. Septembre, 1995.

[Maes *et al.*, 1996] P. Maes, M. Mataric, J. Pollack, J-A. Meyer et S. Wilson, From Animals to Animats *4. Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. The MIT Press/Bradford Books, 1996.

[Mandiau et Lestrugeon, 2002] R. Mandiau et E. G. Lestrugeon « *Systèmes Multi-agents*». Techniques de l'ingénieur, traité Informatique Industrielle S7216. 2002

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

[Marcenac, 1997] P. Marcenac, *Modélisation de systèmes complexes par agents*. Techniques et Sciences Informatiques, 16(8), 1997.

[Marcenac et Giroux, 1998] P. Marcenac et S. Giroux, Geamas : A generic architecture for agent-oriented simulations of complex processes. *Applied Intelligence*, 8(3):247–267, May 1998.

[Marschall et Braun, 2003] F. Marschall et P. Braun, *Model transformations for mda with botl*. University of Twente, 2003.

[Martin *et al.*, 1999] D. Martin, A. Cheyer et D. Moran, *The Open Agent Architecture: a framework for building distributed software systems*, Applied Artificial Intelligence, Vol 13, No ½,p. 91-128, 1999.

[McEntire *et al.*, 1994c] T. Mcentire, J. Weber, G. Wiederhold, M. Genesereth, R. Fritzon, J. McGuire, S. Shapiro, D. McKay, R. Pelavin et C. Beck, "*Specification of the KQML Agent Communication Language plus example agent policies and architectures(DRAFT Report)*", DARPA Knowledge Sharing Initiative External Interface Working Group, 1994.

[McGeaty, 2001] F. McGeaty, *DECAF Programming : An Introduction*, Avril, 2001.

[Mcgeary et Decker, 2001] F. Mcgeary et K. Decker, DECAF Programming*:* Agents for Undergraduates. *In: Proc. Workshop on Infrastructure for Scalable Multi-Agent Systems.* 5th International Conference on Autonomous Agents, 2001.

[Menet, 2010] L. Menet, *Formalisation d'une approche d'Ingénierie Dirigée par les Modèles appliquée au domaine de la Gestion des Données de Référence*, Thèse de doctorat, Université de Paris VIII, 2010.

[Meurisse, 2004] T. Meurisse, Simulation multi-agent : du modèle à l'opérationnalisation. *PhD thesis*, Universite de Paris 6, 2004.

[Mia, 2009] Mia-software, http://www.mia-software.com/, 2009.

[Minar *et al.*, 1996] N. Minar, R. Burkhart, C. Langton et M. Askenazi, The Swarm Simulation System : *A Toolkit for Building Multi-Agent Simulations*. Santa Fe Institute, 1996.

[Michel *et al.*, 2009] F. Michel, J. Ferber et A. Drogoul, : Multi-Agent Systems and Simulation : A survey from the agent's community perspective. Multi-Agent systems : simulation and application edited by A. M. Uhrmacher, D. Weyns– CRC Press- *Taylor and Francis Group* , pp. 3-52, 2009.

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

[MOC, 2001] MoCAH, *Disponible sur* http ://www.ai.univ-paris8.fr/~abchiche/

[Morin, 1977] E. Morin, «*La Méthode (1): la Nature de la Nature* », Le Seuil, Paris, 1977.

[Ndumu *et al.*, 1998] D. T. Ndumu H. S. Nwana et L. C. Lee, ZEUS *:* an advanced toolkit for engineering distributed multi-agent systems, 3rd *int. conf. practical appl. intelligent agents and multi-agent technology*,pp. 377-391, 1998.

[North *et al.*, 2006] M.J. North, N.T. Collier et J.R. Vos, Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit, *ACM Transactions on Modeling and Computer Simulation* 16 (1): 1–25, 2006.

[North *et al.*, 2007] M.J. North, E. Tatara, N.T. Collier et J. Ozik, *Visual Agent-based Model Development with Repast Simphony*", in *CCISE*, 2007.

[Ntaimo et Zeigler, 2004] L. Ntaimo et B.P. Zeigler (2004), Expressing a forest cell model in parallelDEVS and timed cell-DEVS formalisms. *Proceedings of the 2004 Summer Computer Simulation Conference*, 2004.

[Nutaro, 2003] J. NUTARO, adevs, *A Discrete EVent System simulator*. http: //www.ornl.gov/1qn/adevs/, 2003.

[Nwana, 1996] H. S. Nwana, Software Agents: An Overview. *In Knowledge Engineering Review*, Vol. 11(3), pp.205-244, 1996.

[Nwana *et al.*, 1996] H. Nwana, L. Lee et N. Jennings, « *Coordination in software agent systems* », BT Laboratories internal report, 1996.

[Occello *et al.*, 2001] M. Occello, J.-L. Koning et C. Baeijs, « Conception de SMA. Quelques éléments de réflexion méthodologique », *Revue technique et science informatiques*, vol.20, n° 2, Hermès, 2001.

[OMG, 2002] *OMG*, "*ArcStyler MDA-Business*, Transformer Tutorial", 2002.

[OMG, 2003] *Object Management Group OMG*, MDA Guide Version 1.0.1. URL http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, June, 2003.

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

[OMG, 2008] *OMG*, Qvt 2.0 transformation spec, http://www.omg.org/spec/qvt/1.0/pdf/. Rapport technique, OMG, 2008.

[OMG, 2011a] *Object Management Group OMG*, Meta Object Facility (MOF) Core Specification, , Version 2.4.1, URL http://www.omg.org/spec/MOF/ 2.4.1/PDF/, August, 2011.

[OMG, 2011b] *Object Management Group OMG,* Meta Object Facility (MOF) 2.0 Query/ View/ Transformation Specification1.1. URL http://www.omg.org/spec/QVT/1.1/PDF/, January, 2011.

[OptimalJ] Compuware, *Optimalj* 3.0, user's guide, http://www. compuware.com /products/ optimalj.

[Orën, 1987]. T.I. Orën, *Taxonomy of simulation Model processing in encyclopedia of systems and control (eds. M. Singh),* Pergamon Press, 1987.

[OSA, 2001] OSACA, *Disponible sur* http ://www.utc.fr/, 2001.

[Oussalah, 1988] C. Oussalah, *Modèles hiérachisés multi-vues pour le support de raisonnement dans les domaines techniques*. Technical report, 1988.

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

[Padgham et Winikoff, 2002] L. Padgham et M. Winikoff, Prometheus : a Pragmatic Methodology for Engineering Intelligent Agents. *Dans* J. DEBENHAM, B. HENDERSON-SELLERS, N. JENNINGS et J. ODELL, éditeurs : *Proceedings of the OOPSLA 02 - Workshop on Agent-Oriented Methodologies*. COTAR, 2002.

[Padgham et Winikoff, 2003] L. Padgham et M. Winikoff, Prometheus : A Methodology for Developing Intelligent Agents. *Dans* F. GIUNCHIGLIA, J. ODELL et G. WEISS, éditeurs : *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002, Revised Papers and Invited Contributions*, volume 2585 de *Lecture Notes in Computer Science (LNCS),*pages 174–185. Springer-Verlag, 2003.

[PathfinderSolutions, 2003] PathfinderSolutions, *PathMATE*: Transformation Engine, *Electronic Source*: http://www.pathfindermda.com/

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

[Pavon et Gomez-Sanz, 2003] J. Pavon et J. Gomez-Sanz. Agent-oriented Software Engineering with INGENIAS. In *3rd Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03)*, Prague, Czech Republic., 2003.

[Peterson, 1977] J. L. Peterson, Petri nets. *Computing Surveys*, 9(3):223–252, 1977.

[Picard, 2004] G. Picard, *Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente*. Thèse doctorat, Université Paul Sabatier de Toulouse III, 2004.

[Picard et Gleizes, 2004] G. Picard et M.-P. Gleizes, The ADELFE Methodology – Designing Adaptive Cooperative Multi-Agent Systems, Bergenti F., Gleizes M.-P., Zambonelli F. (dir.), *Methodologies and Software Engineering for Agent Systems*, Kluwer Publishing, p. 157-176, 2004.

[Petri, 1962] C.A. Petri, *Kommunikation mit Automaten*. Thèse, Institut für Instrumentelle Mathematik, Bonn, Germany, 1962.

[Pitt et Bellifemine, 1997] J. Pitt, F. Bellifemine, *A Protocol-BasedSemantics for FIPA'97ACL and its Implementation in JADE*. 1997.

[Poggi *et al.*, 2001], A. Poggi , G. Rimassa et M. Tomaiuolo, *Multi-User and Security Support for Multi-Agent Systems*. *In: Proceedings of WOA*, 2001.

[Quesnel, 2006] G. Quesnel, *Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes,* Université du Littoral – Côte d'Opale, Thèse Doctorat, 2006.

[Quesnel *et al.*, 2009] G. Quesnel, R. Duboz et E. Ramat, : The Virtual Laboratory Environment - An operational framework for multi-modelling and analysis of complex dynamical systems, *Simulation and modeling practice and theory*, vol 17, N°4 pp. 641-653, April 2009.

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

[Ramat et Preux, 2003] E. Ramat et P. Preux, : Virtual Laboratory Environment (VLE): a software environment oriented agent and object for modeling and simulation of complex systems, *Simulation and modeling practice and theory*, vol 11, N°01 pp. 45-55, March 2003.

[Rao et Georgeff, 1992] A. S. Rao et M. P. Georgeff, An Abstract Architecture for Rational Agents. Pages 439–449 of : Nebel, Bernhard, Rich, Charles, & Swartout, William R. (eds), *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*. October 25-29, 1992.

[Rao *et al.*, 1995] A. S. Rao et M. P. Georgeff, *BDI-agents : from theory to practice. Rapport technique 56*, Australian Artificial Intelligence Institute, Melbourne, Australia, 1995.

[Reichgelt, 1990] H. Reichgelt, "Differents styles of agents architectures", *Proceedings of the 1st belief representation and agent architectures workshop*, Gallier J.R. (ed), pp29-39, Cambridge, UK, mai 1990.

[Redjimi et Boukelkoul, 2013] M. Redjimi et S. Boukelkoul, Algorthmic tools for the Petri Nets to DEVS transformation, *INFORMATICA 37(4)*, p.41- 418 – ISSN 0350-5596, 2013.

[Rhodes, 2000] J. Rhodes, *Just-In-Time Information Retrieval*. Ph.D. Thesis, MIT Media Lab, May, 2000.

[Ricordel et Demazeau, 2000] P.M. Ricordel et Y. Demazeau, From Analysis to Deployment : A Multi-agent Platform Survey. *Dans* A. OMICINI, R. TOLKSDORF et F. ZAMBONELLI, éditeurs : *Engineering Societiesin the Agent World, First International Workshop, ESAW 2000, Berlin, Germany, August21, 2000, Revised Papers*, volume 1972 de *Lecture Notes in Computer Science (LNCS)*, pages 93–105. Springer-Verlag, 2000.

[Ricordel et Demazeau, 2002] P.M. Ricordel et Y. Demazeau, Volcano, a Vowels-Oriented Multi-agent Platform. *Dans* B. DUNIN-KEPLICZ et E. NAWARECKI, éditeurs : *From Theory to Practice in Multi-Agent Systems, Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, CEEMAS 2001 Cracow, Poland, September 26-29, 2001, Revised Papers*, volume 2296 de *Lecture Notes in Computer Science (LNCS)*, pages 253–262. Springer-Verlag, 2002.

[Rimassa *et al.*, 2000] G. Rimassa, A. Poggi et P. Turci, *An Object-Oriented Framework to Realize Agent Systems*, WOA2000 Workshop, p. 52-57, Parma, Mai, 2000.

[Rowe, 1965] W. Rowe, Why System Science and Cybernetics? IEEE Transactions on Systems and Cybernetics, 1:2-3, 1965.

[RMIa] RMIT *Prometheus*, http: //www.cs.rmit.edu.au/agents/SAC2/methodology.html. University, Melbourne, AUSTRALIA,

[RMIb] , *Prometheus*, http://www.cs.rmit.edu.au/agents/ prometheus/, 2012.

[Robie, 2007] J. Robie, "*Xml processing and data integration with x query*", IEEE Internet Computing, vol. 11, no. 4, pp. 62-67, 2007.

[Robinson, 2006] S. Robinson, Conceptual modeling for simulation : Issues and research requirements. *In Proceedings of the Winter Simulation Conference 2006 (WSC 06)*, pages 792–800, Dec, 2006.

[Rudnick et Gaspari, 2004] J. Rudnick et G. Gaspari, *Elements of the Random Walk - An introduction for Advanced Students and Researchers*. Cambridge University Press, 2004.

[Rumbaugh *et al.*, 1997] J. Rumbaugh, I. Rumbaugh et G. Booch, *Unified Modelling Language*. Reference Manual. ed. AddisonWesley, 1997.

[Rus *et al.*, 1997] D.Rus, R.Gray et D.Kotz, Transportable Information Agent. *Journal f Intelligent Information systems.* 9, 3, pp. 215-238, 1997.

[Russell et Norvig, 1995] S.J. Russell et P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 1995.

[Russell et Norvig , 2003] S.J. Russell et P. Norvig : *Artificial Intelligence : A Modern Approach*. 2edn. Pearson Education, 2003.

[Sargent, 1998] R. G. Sargent, Verification and validation of simulation models. *In Proceedings of the30th Winter Simulation Conference (WSC '98),* IEEE Computer Society Press. Pages 121–130, Los Alamitos, CA, USA, 1998.

[Sarjoughian et Zeigler, 1998] H. Sarjoughian et B. Zeigler, DEVSJava : Basis for a DEVS-based collaborative ms environment. *Dans Actes de la conférence 1998 Society of Computer Simulation International Conference on Web-Based Modelling and Simulation*, volume 5, pages 29–36, San Diego. SCS The Society for Modelling and Simulation International, 1998.

[Sarjoughian et Zeigler, 1999] S.H. Sarjoughian et B.P. Zeigler, The role of collaborative Devs Modeler in federation development. *In Proceedings of the 99 System Interoperability Workshop*, 1999.

[Seddari *et al.*, 2012] N. Seddari, M. Redjimi et L. Benoudina, ‘ Modélisation multi agents d’un processus industriel (processus de génération de vapeur avec le poste d’eau, *Journée nationale d’études doctorales en informatique (JNEDI 2012)*, pp 21-24, Université 20 Aout 1955, Skikda, Algeria, 2012.

[Seddari et Redjimi, 2013] Noureddine Seddari and Mohammed Redjimi. Multi-agent modeling of a complex system. In Information Technology and e-Services (ICITeS), 2013 3rd International Conferenceon, pages 1–6. IEEE, 2013.

[Seddari *et al.*, 2013] Noureddine Seddari, Mohammed Redjimi, and Lazhar. Benoudina. Operational approach for modeling and simulation of an industrial process. In Computer Applications Technology (ICCAT), 2013 International Conference on, pages 1–6. IEEE, 2013.

[Seddari *et al.*, 2014] Noureddine Seddari, Mohammed Redjimi, and Sofiane Boukelkoul. Using of devs and mas tools for modeling and simulation of an industrial steam generator. Journal of computing and information technology, 22(3):171–189, 2014.

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

[Silaghi *et al.*, 2005] R. Silaghi, F. Fondement et A. Strohmeier, "*Weaving mtl model transformations*", pp. 123-138, 2005.

[Simon, 1969] H.A. Simon, *The sciences of the artificial*, Cambridge (MA). MIT Press,1969.

[Smaili, 1994] M. Smaili, *Modélisation à retards flous de circuits logiques en vue de leur simulation*. PhD thesis, 1994.

[SmallDEVS , 2003] SmallDEVS, *Disponible sur* http://perchta.fit.vutbr.cz:8000/projekty/10.

[Smith, 1988] R.G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *Readingsin Distributed Artificial Intelligence, Bond A. H. and GasserL.(Eds.)*, pages 357–366, 1988.

[Searle, 1969] J.R. Searle, Speech acts. *Cambridge University Press, UK*, 1969.

[Softeam, 2009] *Softeam Objecteering Software*, http://www.objecterring.com, 2009.

[Soulie *et al.*, 1998] J.C. Soulie, P. Marcenac, S. Calderoni, et R. Courdier, Geamas v2.0 : an object oriented platform for complex systems simulations. *In Technology of Object-Oriented Languages*, pages 230–242, Aug, 1998.

[Steels, 1994] L. Steels, Building agents with autonomous behavior systems, in L. Steels & R. Brooks, eds, `The artificial life' route to `artificial intelligence'. Building situated embodied agents.', *Lawrence Erlbaum Associates*, New Haven, 1994.

[Steiniger *et al.*, 2012] A. Steiniger, F. Krüger et A. M. Uhrmacher, Modeling Agents and Their Environment in Multi-level-DEVS, *in Proceedings of the Winter Simulation Conference*, 2012.

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

[Sycara *et al.*, 2001] K. Sycara, M. Paolucci, M. V. Velsen et J. Giampapa, *The RETSINA MAS Infrastructure*. *Autonomous Agents and MAS*, Volume 7, Nos. 1 and 2, 2001.

[Taentzer, 2003] G. Taentzer, AGG : A Graph Transformation Environment for Modeling and Validation of Software. *In Proceedings of the 2nd International Workshop Applications of Graph Transformations with Industrial Relevance, AGTIVE 2003*, pages 446-453, Charlottesville, VA, USA, 2003.

[Taillandier *et al.*, 2010] P. Taillandier, A. Drogoul, D. Vo et E. Amouroux, Gama: a simulation platform that integrates geographical information data, agent based modeling and multi-scale control. *In PRIMA*, pp. 78–98, 2010.

[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.

[Tranvouez et Espinasse, 1999] E.Tranvouez et B.Espinasse, Protocoles de coopération pour le réordonnancement d'atelier. Ingénierie des Systèmes Multi-Agent (*Actes des Journées Francophones de l'Intelligence Artificielle Distribuée et des Systèmes Multi-Agents*), *(Novembre 1999 ; Saint Gilles), JFIADSMA'99*, Ed. par Marie ierre Gleizes, Paris : Hermes, 1999.

[Tratt] L. Tratt, QVTEclipse, *Electronic Source*: http://qvtp.org/downloads/qvtp-eclipse/

[Troccoli et Wainer , 2003] A. Troccoli et G. Wainer, Implementing parallel cell-DEVS. *In IEEE, Proceedings of the 36th Annual Simulation Symposium*, 2003.

[Uhrmacher et Shattenberg, 1998] A.M. Uhrmacher et B. Shattenberg, Agents in Discrete Event Simulation, *in Proceedings of ESS98*, 1998.

[Uhrmacher, 2001] A. M. Uhrmacher, Dynamic structures in modeling and simulation: a reflective approach. *ACM Trans. Model. Comput. Simul*., 11 :206–232, ISSN 1049-3301. April, 2001. (Cité page 193.)

[UMT-QVT, 2005] UMT-QVT, "*UML Model Based Tool*", http://umtqvt.sourceforge.net/, 2005.

[Vangheluwe, 2000] H. Vangheluwe, DEVS as a common denominator for multiformalism hybrid systems modelling. *Actes de la conférence IEEE International Symposiumon Computer-Aided Control System Design*, 1(1) :129–134, 2000.

[Vangheluwe, 2001] H. Vangheluwe, *The Discrete EVent System specification DEVS Formalism*. Technical report, http ://moncs.cs.mcgill.ca/, 2001.

[Vangheluwe, 2004] H. Vangheluwe, *The discrete event system specification (DEVS) formalism*. Technical report, 2004.

[Varró *et al.*, 2002] D. Varró, G. Varró et A. Pataricza, Designing the automatic transformation of visual languages. *Science of Computer Programming Journal*, 44(2) :205-227, 2002.

[Vercouter, 2004] L. Vercouter, MAST : Un modèle de composants pour la conception de SMA. *In 1ère Journée Multi-Agents et Composants (JMAC)*, Paris, France, Novembre, 2004.

[Vernadet et Azena, 1992] F. Vernadet et P. Azena, « Prototypage de systemes d'agents communicants ». *Journée Systèmes Multi-Agents PRC-GRD. Intelligence Artificielle*, Nancy, Décembre, 1992.

[Versmisse , 2008] D. Versmisse, *Gestion de la complexité formelle et opérationnelle des systèmes complexes 'Application aux anthroposystèmes marins'*, Thèse Doctorat, Université du Littoral Côte d'Opale, 2008.

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

[Wainer et Giambiasi, 2001a] G. Wainer et N. Giambiasi, Timed cell-DEVS: modeling and simulation of cell spaces. *In Discrete Event Modeling and Simulation: Enabling Future.* Technologies, Springer, 2001.

[Wainer et Giambiasi, 2001b] G. A. Wainer et N. Giambiasi, Application of the cell-DEVS paradigm for cell spaces modeling and simulation. *Simulation*, 76(1):22-39, 2001.

[Wainer, 2002] G. Wainer, CD++: a toolkit to develop DEVS models. *Software – Practice and Experience*, 32:1261–1306, 2002.

[Weiss, 1999] G. Weiss, *Multi-agent Systems. A modern approach to distributed artificial intelligence*. MIT Press, 1999.

[Weyns *et al.*, 2005a] D.Weyns, H. Van Dyke Panurak, F. Michel, T. Holvoet et J. Ferber, :Environments for Multiagent Systems State of the art and Research Challenges. *In Environments for Multi-agent Systems, edited by D .weyns, H. Dyke Panurak, and F. Michel , Vol 3374 of lecture Notes in Computer Science*, 1-47. Berlin, Heidelberg : Springer Berlin Heidelberg., 2005.

[Weyns *et al.*, 2005b] D. Weyns, M. Schumacher, A. Ricci, M. Viroli et T. Holvoet, Environments in Multi-agent Systems. *The Knowledge Engineering Review*, 20 (02), 2005.

[Wilensky, 2004] U. Wilensky, NetLogo, *Center for Connected Learning and Computer-Based Modeling*, Northwestern University. Evanston, IL, 2004.

[Willink, 2003] E.D.Willink, Umlx : *A graphical transformation language for mda*. http ://www.eclipse.org/gmt, September, 2003.

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

[Wooldridge et Jennings, 1995] M. Wooldridge et N.R. Jennings, Agent theories, architectures, and languages : a survey. *Dans Proceedings of theWorkshop on Agent Theories, Architectures, and Languages on Intelligent Agents (ATAL'95)*, pages 1–39. Springer-Verlag, 1995.

[Wooldridge, 2009] M. Wooldridge, An introduction to Multi-agent Systems. *2nd ed. Chistester*, UK Jhon Wiley & Sons, 2009.

[Wooldridge et Ciancarini, 2001] M. Wooldridge et P. Ciancarini, « *Agent-oriented software engineering: the state of the art* ». Editeurs: P. Ciancarini and M. Wooldridge, Agent-Oriented Software Engineering. Springer- Verlag Lecture Notes dans AI Volume, Janvier, 2001.

[Wooldridge, 2000] M. Wooldridge," *Intelligent Agents ". a modern approach to distributed artificial intelligence*. MIT Press., 2000.

[Wooldridge *et al.*, 2000] M. Wooldridge, N. R. Jennings et D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[Zambonelli *et al.*, 2000] F. Zambonelli, N. R. Jennings et M.Wooldridge, «Organisational Abstrations for the Analysis and Design of Multi-Agent Systems» *Proc. 1st Int. Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, 127-141. 2000.

[Zeigler, 1976] B. Zeigler, *Theory of Modeling and Simulation*. Academic Press, 1976.

[Zeigler, 1984] B. Zeigler, *Multifaceted modelling and discrete event simulation*. Academic Press, 1984.

[Zeigler, 1990] B. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models*. Academic Press, 1990.

[Zeigler *et al.*, 1996] B. P. Zeigler, Y. Moon, D. Kim et J. G. Kim, DEVSC++: A high performance modelling and simulation environment. *Dans Hawaii International Conference on System Sciences*, volume 1, pages 350–359, 1996.

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

[Zeigler *et al.*, 2000b] B.P. Zeigler, D. Kim et H. Praehofer, *Theory of modeling and simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.

[Zeigler et Sarjoughian, 2005] B. P. Zeigler et H.S. Sarjoughian, Introduction to DEVS modeling and simulation with JAVA. *Developing component-based simulation models*. Technical report, 2005.

[Zunino et Amandi, 2000] A. Zunino et A. Amandi, Brainstorm/J : a Java Framework for Intelligent Agents, *Argentinina Symposium on Artificial Intelligence*, 2000.

ANNEXES

Annexe 1

Spécifications DEVS atomiques du système

S = {" *actif* "," *passif* "} $\times R_0^+$ et (phase, σ) = σ pour tous les modèles atomiques du système.

APM	***GUS***
InPorts={"r1", "r2", "r3", "r4", "r5", r6", "r7", "r8", "r9", "r10"} *X = {(r1, d_e_c), (r2,* n_b*), (r3, d_e_b), (r4, n_c), (r5, c_g), (r6, d_v),(r7, n_b), (r8, d_e), (r9,d_g), (r10, d_a)}* *OutPorts = {"r11", "r12", "r13", "r14", "r15"}* *Y={(r11, d_e_c), (r12, d_e), (r13, o_s), (r14, d_v), (r15, c_g)}* *δext (phase, σ, e, X) =* *("actif",temps_régulation) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ", ∞) si APM.X.V = Ø* *("actif ", temps_régulation) sinon* *λ ("actif ") = d_e_c, d_e, d_v, c_g et o_s*	*InPorts={"n1"}* *X = {(n1, c_g)}* *OutPorts ={"n2"}* *Y={(n2, c_g)}* *δext (phase, σ, e, X) =* *(" actif ", temps_perturbation) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si GUS. X.V = Ø* *("actif ", temps_perturbation) sinon* *λ ("actif ") = c_g*
15LIC05	***15B01***
InPorts={"a1", "a2","a3"} *X ={(a1,* n_b*), (a2, d_e), (a3, d_e_c)}* *OutPorts = {"a4"}* *Y={(a4, o_al)}* *δext (phase, σ, e, X) =* *("actif ",temps_alimentation) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ", ∞) si 15LIC05. X.V = Ø* *("actif ", temps_alimentation) sinon* *λ ("actif ") = o_al*	*InPorts={"c1", "c2"}* *X ={(c1, d_e), (c2, d_e_c) }* *OutPorts = {"c3"}* *Y={(c3,* n_b*)}* *δext (phase, σ, e, X) =* *("actif ", temps_calcule n_b) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ", ∞) si 15B01. X.V = Ø* *("actif ",temps_calcule n_b) sinon* *λ ("actif ") = n_b*

15FVC	**15LIC12**
InPorts={"b1", "b2"} *X ={(b1, o_s), (b2, o_al)}* *OutPorts = {"b3"}* *Y={(b3, d_e_c) }* *δext (phase, σ, e, X) =* *("actif ", temps_calcule d_e_c) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif",∞) si 15LIC12. X.V = Ø* *("actif", temps_calcule d_e_c) sinon* *λ ("actif ") = d_e_c*	*InPorts={"d1", "d2","d3"}* *X ={(d1, d_e_b), (d2, n_c),(d3, d_e_c)}* *OutPorts = {"d4"}* *Y={(d4, o_cn)}* *δext (phase, σ, e, X) =* *("actif ",temps_condensation) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15LIC12. X.V = Ø* *("actif ", temps_condensation) sinon* *λ ("actif ") = o_cn*
15E01	**15FVA**
InPorts={"e1", "e2} *X ={(e1, d_e_c), (e2, d_e_b)}* *OutPorts = {"e3"}* *Y={(e3, n_c)}* *δext (phase, σ, e, X) =* *("actif ", temps_calcule n_c) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15E01. X.V = Ø* *("actif ", temps_calcule n_c) sinon* *λ ("actif ") = n_c*	*InPorts={" f1"," f2"}* *X ={(f1,o_s), (f2,o_cn)}* *OutPorts = {"f3"}* *Y={(f3, d_e_b)}* *δext (phase, σ, e, X) =* *("actif ", temps_calcule d_e_b) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15FVA. X.V = Ø* *("actif", temps_calcule d_e_b) sinon* *λ ("actif ") = d_e_b*
15HIC01	**15KT01**
InPorts={" j1", " j2", " j3"} *X ={(j1, d_a), (j2, c_g), (j3, d_g)}* *OutPorts = {" j4 ", " j5"}* *Y={(j4, d_v), (j5,o_cm)}* *δext (phase, σ, e, X) =* *("actif ", temps_combustion) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15HIC01. X.V = Ø* *("actif", temps_combustion) sinon* *λ ("actif ") = d_v et o_cm*	*InPorts={" m1", " m2"}* *X ={(m1,o_s), (m2,o_cm)}* *OutPorts = {"m3"}* *Y={(m3, d_a) }* *δext (phase, σ, e, X) =* *("actif ", temps_calcule d_a) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ", ∞) si 15KT01. X.V= Ø* *("actif ", temps_calcule d_a) sinon* *λ ("actif ") = d_a*

15FVG	***15LIC01A***
InPorts={" k1", " k2"} *X ={(k1,o_cm), (k2,o_s)}* *OutPorts = {"k3"}* *Y={(k3, d_g)}* *δext (phase, σ, e, X) =* *("actif ", temps_calcule d_g) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15FVG. X.V = Ø* *("actif ", temps_calcule d_g) sinon* *λ ("actif ") = d_g*	*InPorts={"g1", "g2","g3"}* *X ={(g1, d_e), (g2, d_v), (g3, n_b)}* *OutPorts = {" g4"}* *Y={(g4, o_hy)}* *δext (phase, σ, e, X) =* *("actif ", temps_hydraulique) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ", ∞) si 15LIC01A. X.V = Ø* *("actif ", temps_hydraulique) sinon* *λ ("actif ") = o_hy*
15FV03	***15FV03***
InPorts={" i1", " i2"} *X ={(i1, o_s), (i2, o_hy)}* *OutPorts = {" i3 "}* *Y={(i3, d_e) }* *δext (phase, σ, e, X) =* *("actif ", temps_calcule d_e) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15FV03. X.V = Ø* *("actif ", temps_calcule d_e) sinon* *λ ("actif ") = d_e*	*InPorts={"h1", "h2"}* *X ={(h1, d_v), (h2, d_e) }* *OutPorts = {"h3"}* *Y={(h3, n_b)}* *δext (phase, σ, e, X) =* *("actif ", temps_calcule n_b) si phase = "passif"* *(phase, σ-e) sinon* *δint (phase, σ) =* *("passif ",∞) si 15FV03. X.V = Ø* *("actif ", temps_calcule n_b) sinon* *λ ("actif ") = n_b*

Annexe 2

Spécifications DEVS couplé du système

SIMULATEUR(SIM)	***PERTURBATION(PER)***
InPorts={"s1"} *X ={(s1, c_g)}* *OutPorts= {"s2"}* *Y={(s2, d_v)}* *D ={ALI, PER, CON, COM, HYD}* *MALI =ALIMENTATION* *MPER =PERTURBAT ION* *MCON =CONDENSATION* *MCOM =COMBUSTION* *MHYD =HYDRAULIQUE* *EIC ={(SIM , s1),(PER, z5)}* *EOC ={(COM, v3), (SIM , s2)}* *IC={((ALI,x3),(PER,z3)),((ALI,x4),(PER,z2)), ((CON,y3),(PER,z4)),((CON,y4),(PER,z1)), ((HYD,u3),(PER,z7)),((HYD,u4),(PER,z6)), ((COM,v3),(PER,z9)),((COM,v4),(PER,z10)), ((COM,v5),(PER,z8)),((PER,z11),(CON,y2)),((PER,z12), (ALI,x1)),((PER,z13),(CON,y1)),((PER,z13), (COM,v2)),((PER,z13),(HYD,u1)),((PER,z13),(ALI,x2)), ((PER,z14), (HYD,u2)),((PER,z15),(COM,v1))}*	*InPorts = {"z1", "z2","z3", "z4", "z5","z6", "z7", "z8", "z9", "z10"}* *X ={(z1, n_c),(z2, d_e_c), (z3, n_b), (z4, d_e_b), (z5, c_g), (z6, n_b),(z7, d_e),(z8, d_g), (z9, d_v), (z10, d_a)}* *OutPorts={"z11", "z12", "z13", "z14", "z15"}* *Y={(z11, d_e_c), (z12, d_e), (z13, o_s),(z14, d_v), (z15, c_g)}* *D= {GUS, APM }* *MGUS= PERTURBATEUR* *MAPM= CONTROLEUR* *EIC= {((PER, z1), (APM,r4)), ((PER,z2), (APM,r1)), ((PER,z3), (APM,r2)),((PER,z4), (APM,r3)),((PER,z5), (GUS,n1)),((PER,z6), (APM,r7)),((PER,z7), (APM,r8)), ((PER,z8), (APM,r9)), ((PER,z9), (APM,r6)), ((PER, z10), (APM, r10)) }* *EOC= {((APM, r11),(PER, z11)), ((APM, r12), (PER, z12)), ((APM, r13), (PER, z13)), ((APM, r14), (PER, z14)), ((APM, r15),(PER,z15)) }* *IC ={(GUS,n2), (APM,r5) }*
ALIMENTATION (ALI)	***CONDENSATION (CON)***
InPorts={"x1", "x2"} *X ={(x1, c_g), (x2, o_s)}* *OutPorts={"x3","x4"}* *Y={(x3,* n_b*), (x4, d_e_c)}* *D= {15LIC05,15B01, 15FVC }* *M15LIC05=Régulateur* *M15B01=Bâche* *M15FVC=Vanne Eau* *EIC = {((ALI , x1), (15LIC05,a2)),((ALI, x2), (15FVC, b1)), ((ALI, x1),(15B01,c1))}* *EOC ={((15FVC, b3), (ALI,x4)), ((15B01, c3), (ALI, x3))}* *IC ={((15LIC05,a4), (15FVC,b2)), ((15B01,c3), (15LIC05,a1)),((15FVC,b3),(15B01,c2)), ((15FVC,b3), (15LIC05,a3)) }*	*InPorts= {"y1", "y2"}* *X ={(y1, o_s), (y2, d_e_c)}* *OutPorts={"y3", "y4"}* *Y={(y3, d_e_b), (y4, n_c)}* *D={15LIC12,15E01,15FVA }* *M15LIC12= Régulateur* *M15E01= Condenseur* *M15FVA = Vanne Eau* *EIC ={((CON , y2),(15LIC12,d3)), ((CON, y2), (15E01,e1)),((CON, y1),(15FVA,f1))}* *EOC ={((15E01, e3), (CON, y4)), ((15FVA , f3), (CON , y3))}* *IC= {((15LIC12,d4), (15FVA,f2)), ((15FVA,f3), (15LIC12,d1)), ((15FVA,f3), (15E01,e2)), ((15E01,e3), (15LIC12,d2)) }*

COMBUSTION(COM)	***HYDRAULIQUE(HYD)***
InPorts= {"v1", "v2"} *X ={(v1, c_g), (v2, o_s)}* *OutPorts={"v3", "v4","v5"}* *Y={(v3, d_v), (v4,d_a), (v5, d_g)}* *D={15HIC01,15KT01,15FVG }* *M15HIC01= Régulateur* *M15KT01=Turbo-ventilateur* *M15FVG =Vanne Gaz* *EIC = {((COM , v1), (15HIC01, j2), ((COM, v2),(15KT01,m1)), ((COM, v2),(15FVG,k2))}* *EOC={((15HIC01, j4), (COM , v3)), ((15KT01,m3), (COM , v4)), ((15FVG, k3), (COM , v5))}* *IC={((15HIC01, j5), (15KT01,m2)), ((15HIC01,j5), (15FVG,k1)), ((15KT01,m3), (15HIC01,j1)), ((15FVG,k3), (15HIC01,j3)) }*	*InPorts= {"u1", "u2"}* *X ={(u1, o_s), (u2, d_v)}* *OutPorts={"u3", "u4"}* *Y={(u3, d_e), (u4, n_b)}* *D={15LIC01A,15FV03, 15H01 }* *M15LIC01A = Régulateur* *M15FV03= Vanne Eau* *M15H01 =Ballon* *EIC ={((HYD , u2), (15LIC01A, g3)), ((HYD, u2),(15H01,h1)),((HYD, u1),(15FV03,i1))}* *EOC = {((15FV03, i3), (HYD, u3)), ((15H01, h3), (HYD , u4))}* *IC={((15LIC01A,g4), (15FV03,i2)), ((15FV03, i3),(15LIC01A,g1)),((15FV03,i3), (15H01,h2)), ((15H01,h3), (15LIC01A,g3))}*