THÈSE

présentée en vue de l'obtention du titre de

DOCTEUR
de
l'École Nationale Supérieure
des Mines de Saint-Étienne et de
l'Université Jean Monnet
(Arrêté ministériel du 30 mars 1992)

Spécialité : Informatique

par

Stéphane GALLAND

Approche multi-agents pour la conception et la construction d'un environnement de simulation en vue de l'évaluation des performances des ateliers multi-sites

Soutenue le 20 Décembre 2001 devant le jury :

Président :	Pr. Bernard GRABOT	ENI Tarbes
Rapporteurs :	Pr. François VERNADAT	Université de Metz
	Pr. René MANDIAU	Université de Valenciennes
Examinateurs :	Pr. Jean-Pierre MÜLLER	CIRAD Montpellier
	Pr. Jean-Pierre CAMPAGNE	ENSM Saint-Étienne
	Ma. Frédéric GRIMAUD	ENSM Saint-Étienne
	Ma. Philippe BEAUNE	ENSM Saint-Étienne

Thèse préparée au sein des laboratoires MSGI et SMA de l'ENSM/SE

À mon frère
À mes parents
À ma sœur
À mes deux âmes sœurs

Remerciements

Je souhaite remercier Monsieur Jean-Pierre Campagne pour avoir accepté de m'accueillir au sein de son équipe de recherche. Ses conseils et son aide ont largement contribué à la réalisation de mes travaux de recherche.

Je remercie également Messieurs Frédéric Grimaud et Philippe Beaune d'avoir encadré ma thèse. Ils m'ont apporté leur expérience et leurs conseils. Ce travail ne serait pas tel qu'il est sans leurs points de vue critiques sur mes travaux, et sans leur soutien humain par leurs encouragements.

Je remercie Messieurs François Vernadat et René Mandiau pour avoir accepté de rapporter cette thèse. Leur examen et leurs remarques ont contribué à l'amélioration de ce document. Je remercie également Messieurs Jean-Pierre Müller et Bernard Grabot d'avoir accepté de participer à mon jury.

Je tiens à remercier :

Monsieur Olivier Boissier pour sa disponibilité, ainsi que pour ses remarques et ses critiques éclairées.

Tout ceux qui m'ont accompagné tout au long de ces trois années, et plus particulièrement Saliha, Pascal, Mahdi, Laurent, Hubert, Thibaut, Franck, Gildas, Lyes et Amel.

Les membres des centres SIMMO et SITE de l'École des Mines pour leur sympathie et l'ambiance chaleureuse qu'ils m'ont réservés.

Laurent pour son soutien constant, sa bonne humeur perpétuelle, et son amitié indeffectible.

La douce Victoria : tu fus pour moi le soleil de mes nuits par ton souvenir et ta gaîté.

La tendre Christine. Je n'oublierai pas ce qui nous attache l'un à l'autre, et ce que cela m'a permis de réaliser pendant ces trois années.

Merci à mes parents et à ma sœur qui m'ont apporté leur soutien inconditionnel et constant. Je les remercie également de tout ce qu'ils ont fait pour moi durant toutes ces années.

Résumé

Nous nous situons dans le contexte de la simulation de systèmes industriels complexes et distribués en termes opérationnel, informationnel et décisionnel. Nous considérons plus particulièrement les problèmes de formalisation, de modularité, de centralisation et de mise en évidence des flux et sous-systèmes. En effet, l'évolution du contexte industriel pousse les entreprises à évoluer vers des systèmes de plus en plus décentralisés (entreprises virtuelles, groupement d'entreprises, décentralisation, ...). Les méthodes et les outils de simulation existants ne supportent pas de manière optimale ces nouveaux systèmes. En effet, il reste aujourd'hui très difficile de modéliser et simuler le comportement de systèmes tels que les groupements d'entreprises. Après avoir mis en évidence cette problématique, nous proposons dans le cadre de nos travaux de recherche une approche méthodologique adaptée aux systèmes industriels fortement distribués. Cette approche est basée sur les systèmes multi-agents et reste indépendante de toute plateforme ou outil informatique. Nous proposons un cycle de vie et une première définition des phases les plus importantes : spécification d'un modèle de simulation, conception d'un modèle multi-agents et implantation d'un modèle informatique. Les concepts que nous mettons en œuvre (systèmes multi-agents, systémique, ...) nous permettent de répondre aux différents problèmes posés par les systèmes de production complexes et distribués.

Mots clés : Simulation de systèmes industriels, Systèmes multi-agents, Méthodologie, Distribution, Systémique

Abstract

We are located in the context of simulation of industrial systems, whish are complex and distributed in operational, informational and decisional terms. In this context, we consider the problems of formalization, modularity, centralization and highlighting of flows and subsystems. Indeed, the evolution of the industrial context pushes the companies to evolve to more and more decentralized systems (virtual enterprises, consortium, decentralization...). Existing methodologies and simulation tools make difficult to have an optimal support of these new systems. Indeed, there remains today very difficult to model and simulate the behavior of systems such as the consortia. After highlighted these problems, we propose within the framework of our research a methodological approach adapted to the strongly distributed industrial systems. This approach is based on the multi-agent systems but remains independent of any platform or simulation tool. We propose a life cycle and a first definition of the most significant phases: specification of a simulation model, design of a multi-agent model and implementation of a data-processing model. The concepts which we use (multi-agent systems, systemic...) enable us to answer the various problems arising from the complex and distributed systems of production.

Keywords: Industrial system simulation, Multi-agent systems, Methodology, Distribution, Systemic

Table des matières

Table des figures

Liste des tableaux

Liste des définitions

Première partie

Introduction

Chapitre 1

Introduction générale

L'étude des systèmes industriels a fortement changé au cours de ces dernières années tant du point de vue de la modélisation que de la simulation. Plus particulièrement, l'évolution du contexte économique a donné naissance à de nouvelles organisations d'entreprises. Ainsi, pour faire face aux nouvelles contraintes de leur environnement, les entreprises ont dû s'allier et se grouper au sein d'entités collectives. Les systèmes industriels distribués sont apparus sous plusieurs formes (groupement d'entreprises, entreprises virtuelles, ...). Face à ce contexte, la modélisation et la simulation de tels systèmes montrent de réelles faiblesses. Par exemple, les méthodes et les outils de simulation actuels ne supportent pas les concepts de groupement d'entreprises. Certes, il est déjà possible de réaliser des modèles de simulation de ces groupements, mais c'est sans tenir compte de l'ensemble des contraintes propres à ce type d'organisation (confidentialité, dynamique de composition, ...). D'autre part, la simulation pose de nombreux problèmes méthodologiques. Il reste toujours difficile de savoir quoi modéliser et comment. Ainsi, la qualité des modèles de simulation est trop souvent assujettie aux compétences et à l'expérience du modélisateur. De plus, nous pensons qu'il y a de grands avantages à modéliser un système industriel selon une approche systémique (augmentation de la modularité et facilitation de la compréhension).

Dans ce contexte, nous proposons une approche méthodologique permettant de répondre aux problèmes de modélisation pour la simulation de systèmes industriels distribués. Cette approche méthodologique se base sur les travaux déjà existants et les étend afin de mieux supporter les nouvelles organisations d'entreprises. Ainsi, nous proposons de créer des modèles de simulation de haut niveau grâce à un langage décrit par un métamodèle Uml. Ensuite, nous considérons que les systèmes multi-agents apportent une solution au support de la simulation des trois sous-systèmes composant une entreprise (sous-systèmes opérationnel, informationnel et décisionnel). Nous proposons de transformer le modèle de simulation de haut niveau en un modèle de système multi-agents. Cette opération est réalisée durant la

phase de conception de notre approche méthodologique. Finalement, nous proposons une architecture logicielle capable de mettre en œuvre l'ensemble des modèles précédemment définis.

Ce mémoire est articulé autour de trois grandes parties :

- La partie I est consacrée à la définition du contexte de recherche dans lequel nous nous situons. Nous y définissons les concepts de système industriel distribué, de simulation de systèmes industriels. Nous y exposons les grandes lignes des problématiques que nous voulons aborder. Nous sommes conscients que l'étude et la mise en œuvre d'une méthodologie forment un travail de longue haleine. Par conséquent, nous limitons nos travaux à certains aspects particuliers. Notre objectif principal est de fournir les bases d'un environnement adapté à la modélisation et à la simulation de systèmes industriels distribués.
- Dans la partie II, nous présentons un tour d'horizon des travaux scientifiques déjà réalisés. Cet exposé est divisé en trois parties. Le chapitre 4 est consacré à la modélisation en entreprise et plus particulièrement aux méthodologies et aux outils qui nous paraissent intéressants dans le cadre de notre problématique. Dans le chapitre 5, nous exposons les concepts de base de la simulation, ainsi que deux méthodologies de modélisation spécifiques. Enfin, nous exposons les concepts des systèmes multi-agents dans le chapitre 6. De plus, tout au long de cette partie, nous présentons les raisons des choix que nous effectuons pour répondre à la problématique initiale, et qui nous ont amenés à proposer une approche méthodologique nommée $\mathcal{M}_{\mathcal{A}}\mathcal{M}\mathcal{A}$-$\mathcal{S}$.
- La partie III est consacrée à la présentation de nos propositions. Plus particulièrement, nous y exposons le cycle de vie de notre approche méthodologique dans le chapitre 8, puis les phases de spécification, conception, et enfin l'implantation d'un environnement de modélisation et de simulation.

Chapitre 2

Contexte de recherche

Ce chapitre nous permet d'expliquer l'objet de notre recherche. Notamment, nous présentons successivement notre domaine de recherche, et les problématiques qui nous intéressent au sein de ce même contexte. Enfin, nous concluons ce chapitre par un bref exposé de nos propositions.

2.1 Domaine de recherche

Dans cette section nous définissons le cadre de nos travaux de recherche : la *simulation à évènements discrets de systèmes manufacturiers complexes et distribués.*

2.1.1 Systémique

Depuis plusieurs dizaines d'années a émergé un nouveau concept capable d'aider à la résolution de problèmes complexes dans des domaines divers. Cette notion moderne de *système* (ou approche systémique) succède à l'approche rationaliste classique décrite dans le *Discours de la méthode* de Descartes [Descartes, 1637]. À l'origine mise en évidence par des hommes tels que Von Bertalanffy, Wiener, Shannon ou Forrester [Durand, 1975], nous utilisons la définition de [De Rosnay, 1975] :

— Définition 2.1 : Système —

Un système est un ensemble d'éléments en interaction, organisés en fonction d'un but.

Les systèmes s'organisent essentiellement autour de quatre axes :

– **l'interaction :** deux éléments d'un système possèdent une relation causale bijective c'est-à-dire que l'élément A influence l'élément B et inversement. La rétroaction est une forme particulière d'interaction.

– **l'organisation :** les éléments d'un système sont organisés afin de composer des entités au comportement propre.

– **la globalité :** selon VON BERTALANFFY, un système est un ensemble non réductible à ses éléments. Un système possède des qualités que n'ont pas ses éléments.

– **la complexité :** la complexité d'un système est définie par le nombre et les caractéristiques de ses éléments, les incertitudes, les aléas propres à l'environnement du système et les relations entre les notions de déterminisme et d'indéterminisme.

D'un point de vue fonctionnel, un système se caractérise par un ensemble de *flux* (information, produits, ...), de *centres de décisions* traitant les informations et agissant sur les flux, de *boucles de rétroaction* pour avoir des informations sur ce qui se passe en aval, de *délais de réponse* et d'*entrées/sorties* représentant les interactions du système avec son environnement.

D'un point de vue structurel, un système est principalement l'agrégation de *ressources* et de *relations* qui permettent de le distinguer de son environnement.

Un système doit réagir en permanence avec les informations provenant de son environnement. Pour répondre à ce besoin plus ou moins important d'interaction, on distingue théoriquement les systèmes ouverts pour lesquels l'environnement peut agir sur le comportement du système, et les systèmes fermés qui sont repliés sur eux-même. Pratiquement, seuls existent des systèmes ouverts répondant plus ou moins efficacement à leurs environnements. De plus, il est nécessaire pour un système de s'assurer de sa stabilité face à son comportement interne mais aussi face à son environnement. Enfin un système peut être organisé en *niveaux hiérarchiques* composés de *sous-systèmes* spécialisés.

Dans la section suivante, nous présentons les systèmes qui nous intéressent dans le cadre de ce mémoire : les systèmes industriels de production.

2.1.2 Système industriel de production

Nous nous intéressons à une classe particulière de système : les systèmes industriels de production à flux discrets et à partage de ressources. Nous pouvons les définir plus formellement grâce aux définitions suivantes :

— Définition 2.2 : Système de production manufacturier —

« *Un système de production est un ensemble de ressources qui permettent (de transformer des matières premières et/ou des composants en produits finis). Dans cet ensemble, on distingue essentiellement quatre types de ressources : des équipements (machines, outils, moyens de transport, moyens informatiques, ...), des moyens humains qui permettent le bon déroulement du processus de transformation, des produits à différentes étapes de fabrication (matière premières, produits semi-finis, produits finis, ...), des entrepôts de matières et des aires de stockage.* » [Ye, *1994*]
« *Un système de production manufacturier est un centre de transformation de matérieux bruts en produits finis destinés à alimenter des clients en mobilisant des ressources (machines, opérateurs, matières premières ou composants, zones de stockage, ...).* » [Dupont, *1998*]

Un système de production doit remplir les fonctions décrites dans le tableau 2.1. Il s'agit des fonctions principales de fabrication, de transport et de stockage.

Fonction	Action	
Fabrication	Fabriquer	usiner, emboutir, ...
	Contrôler	vérification de la conformité de la production aux critères de qualité
Transport	Manipuler	placer les pièces en position de travail adéquate et les replacer sur le système de manutention
	Manutentionner	déplacement d'une pièce d'un poste de travail vers un autre
Stockage	Stocker	placer les pièces sur une aire de stockage

Tab. 2.1 – Fonctions principales d'un système de production

2.1.2.1 Ressources

Pour réaliser ses tâches de fabrication et de transport, le système de production a besoin de ressources. Les définitions 2.3 et 2.4 décrivent les deux types de ressources existantes [Grimaud, 1996].

— DÉFINITION 2.3 : RESSOURCE ACTIVE —

Ce sont des ressources qui possèdent une certaine autonomie vis-à-vis du système auquel elles appartiennent, c'est à dire que leur état peut évoluer indépendamment des autres entités qui composent le système. Ce sont par exemple des machines, des ressources humaines. Ces ressources sont généralement structurées à l'aide d'une hiérarchie. Pour que les ressources actives puissent réaliser leurs opérations, il est indispensable d'utiliser des ressources passives.

— DÉFINITION 2.4 : RESSOURCE PASSIVE —

Ce sont des ressources qui ne participent pas directement à l'élaboration ou au transport d'une opération. Ce sont par exemple, les supports servant pour le transport de pièces comme les palettes, les rouleaux, les chariots, ...

Certaines ressources peuvent être critiques. Par exemple, un nombre limité de palettes peut être considéré comme une ressource critique à un moment donné du processus de production. De même, un opérateur humain réalisant la maintenance de plusieurs machines se trouvant à des endroits éloignés peut être considéré comme une ressource critique.

2.1.2.2 Flux

Un flux représente un ensemble d'entités passives qui parcourent le système de production. Les flux peuvent être de types différents : les *matières premières*, ou les produits finis ou semi-finis. On trouve également des flux d'*informations*.
Les entités composant les flux sont la source des activités des ressources actives du système. Elles constituent la *charge du système*.
La formalisation de la charge du système passe en général par l'utilisation des *gammes* et des *nomenclatures* [GRIMAUD, 1996].

— DÉFINITION 2.5 : GAMME —

C'est l'ensemble des actions permettant d'atteindre un objectif de fabrication, de contrôle, de transport, de stockage ou de maintenance. Une gamme peut comporter des gammes intermédiaires et/ou des cycles séquentiels ou parallèles.

— Définition 2.6 : Nomenclature —

C'est l'ensemble des composants et des composés intermédiaires qui entrent dans la fabrication d'un produit. Une nomenclature est représentée par une arborescence qui permet de connaître les composants et composés intermédiaires.

Un même composant peut apparaître dans plusieurs gammes et dans plusieurs nomenclatures. De plus, tout élément de flux suit une gamme connue a priori ou non et appartient à une nomenclature.

2.1.2.3 Gestion

Les systèmes de production sont gérés à l'aide de règles qui concourent à son bon fonctionnement. Elles sont généralement mises en œuvre grâce à des capteurs et des actionneurs appartenant aux ressources des systèmes.

— Définition 2.7 : Capteur —

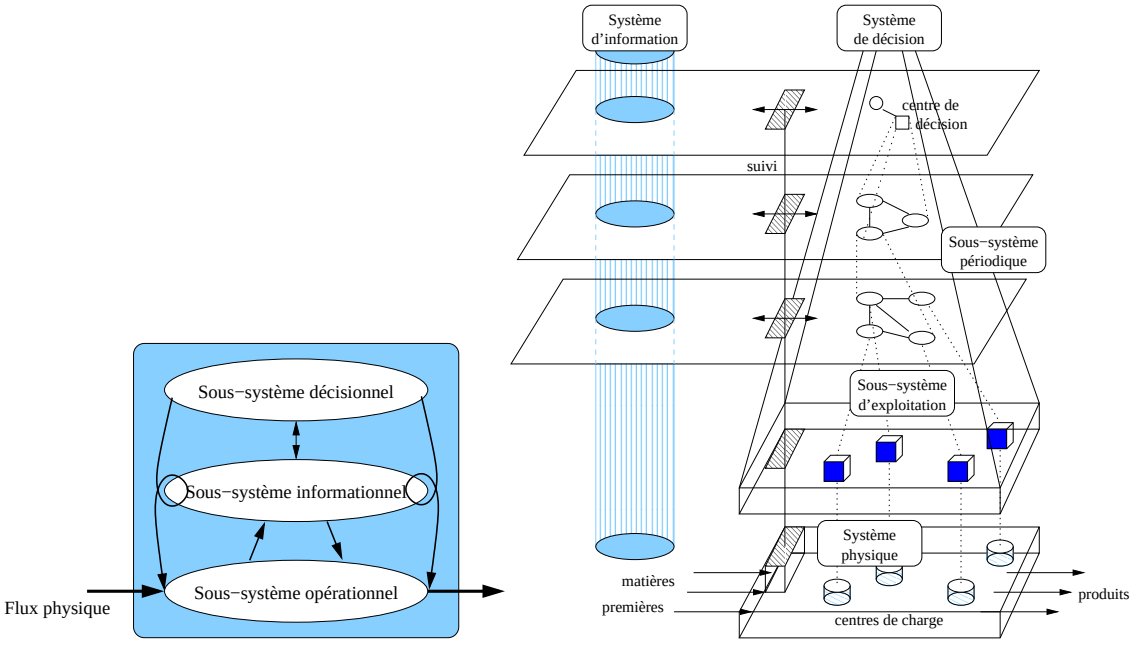
Il s'agit d'un élément physique qui collecte un flux informationnel. La valeur donnée par le capteur est une composante du vecteur d'état du système. Il permet d'obtenir par exemple des informations sur des entités physiques ou sur le nombre de composants assemblés à un instant donné.

— Définition 2.8 : Actionneur —

Il permet de mettre en œuvre des actions sur le système, décidées par des règles, pour modifier le comportement de ce système. Des règles de complexité moyenne peuvent être utilisées pour déterminer le choix d'une ressource parmi plusieurs.

2.1.2.4 Une approche systémique

Cette définition de système de production manufacturier peut être rapprochée de celle de [Le Moigne, 1992]. En effet, comme l'illustre la figure figure 2.1(a) page suivante, il considère qu'un système de production est composé de trois sous-systèmes. Le sous-système opérationnel représente l'ensemble des infrastructures physiques du système de production (machines, ressources passives, ...). Le sous-système décisionnel correspond à l'ensemble des centres de prise de décisions. Ils



(a) LE MOIGNE [LE MOIGNE, 1992] (b) GRAI [BOUTIN, 2001]

FIG. 2.1 – Approche systémique

gèrent les règles de gestion du système comme par exemple la génération d'ordres ou la gestion d'allocation des ressources. Le sous-système informationnel est essentiellement un vecteur de communication entre les deux autres sous-systèmes. Par exemple, les gammes et les nomenclatures définies dans la partie informationnelle du système permettent aux centres de décisions de gérer les trajets des produits au sein du système.

De plus, il existe d'autres approches systémiques. Citons, par exemple, la vision proposée par GRAI [MARCOTTE, 1995]. Cette méthode, basée sur la théorie des systèmes de [LE MOIGNE, 1977 ; LE MOIGNE, 1992] et sur la théorie des systèmes hiérarchisés [MEZAROVIC et al., 1970], propose une vision illustrée par la figure 2.1(b). Contrairement à l'approche de Jean-Louis LE MOIGNE, le système d'information est placé parallèlement au système de décision et au système physique.

2.1.2.5 Problèmes posés par un système de production

Les systèmes de production posent lors de leur conception et lors de leur exploitation, un certain nombre de problèmes structurels, fonctionnels et organisationnels. On peut les regrouper en six groupes [GRIMAUD, 1996] :

– **les problèmes de dimensionnement :** Il s'agit de déterminer de la meilleure façon possible les capacités des machines, le nombre de ressources, les capacités

des stocks, ...

– **les problèmes de fonctionnement :** La résolution de ces problèmes permettent une compréhension plus précise du système. On essaye par exemple de répondre à des questions comme : « Existe-t'il des goulets d'étranglement ? Peut-on prendre en compte des demandes prioritaires sans induire de trop fortes perturbations ? ... »

– **les problèmes de productivité :** Peut-on être sûr que les ressources sont utilisées au mieux ? Peut-on minimiser le nombre de ressources actives dans un atelier ? Faut-il produire de manière plus régulière pour augmenter la production ? ...

– **les problèmes de maintenance :** Doit-on arrêter une machine pour une maintenance préventive ou attendre qu'elle tombe en panne pour agir ? ...

– **les problèmes d'aléas ou de pannes des ressources :** Quel sera le comportement du système s'il fonctionne en mode dégradé ? À quel taux de panne, telle ressource devient-elle critique pour l'atelier ? ...

– **les problèmes d'ordonnancement :** Peut-on améliorer le rendement d'un système en proposant un meilleur ordonnancement de la production ? Peut-on supprimer une machine en réorganisant la production différemment ? ...

2.1.3 Systèmes distribués de production

Conscientes de l'impossibilité de maîtriser tous leurs métiers, les entreprises d'aujourd'hui ont tendance à se transformer, à se réorganiser et à profiter des opportunités que leur offre leur environnement en terme de compétences. On assiste à un recentrage de certaines entreprises sur les métiers dans lesquels elles possèdent un avantage compétitif, et à une externalisation des métiers annexes considérés comme les métiers dans lesquels elles ne peuvent exceller.

Ainsi, « les entreprises sont lancées dans le changement, elles adoptent une philosophie caractérisée par une vision globale, intuitive, et réactive et une démarche de l'entreprise vers l'intelligence : l'entreprise est devenue un système d'apprentissage collectif où les vraies richesses sont le savoir et les compétences qu'elle accumule. L'organisation apprend et se transforme en permanence. L'entreprise n'a plus les frontières aussi nettes, elle intègre ses clients, ses fournisseurs et ses partenaires dans sa structure, qui s'étend ou se rétracte au rythme des alliances qu'elle passe ou des projets qu'elle lance. » [Couture et Loussararian, 1999]

Dans ce contexte ont émergé de nouvelles formes d'organisation d'entreprises.

2.1.3.1 Formes majeures

Les formes les plus courantes de ces nouveaux types d'entreprises sont l'*entreprise réseau*, l'*entreprise virtuelle*, l'*entreprise étendue* ou les *réseaux d'entreprises*.

La forme la plus connue est l'entreprise réseau. Nous avons choisi les deux définitions suivantes :

— DÉFINITION 2.9 : ENTREPRISE RÉSEAU —

« C'est un ensemble d'entreprises liées les unes aux autres par un cycle de production. Le lien n'est ni juridique, ni structurel ; il revêt souvent la forme de simples accords. Ces entreprises ont en commun un puissant système de coopération fonctionnelle. » [BUTERA, 1991]
« L'entreprise réseau est une entreprise qui a choisi d'étendre son action pour partager les défis d'environnement économique dont elle maîtrise d'autant mieux la complexité qu'elle est outillée des compétences de ses partenaires. Des partenaires convaincus comme elle-même des multiples avantages d'une alliance. » [POULIN et al., 1994]

La communauté scientifique utilise aussi les termes « filières » (BELLON), « constellation d'entreprises » (LORENZONI) ou « entreprises modulaires » (BRILMAN) pour désigner une entreprise réseau. Cette même communauté considère trois types d'oganisation d'entreprise réseau :

a) l'entreprise à *base hiérarchique* qui désigne une organisation composée de relations quasi hiérarchiques entre les donneurs d'ordres et les entreprises ;

b) l'entreprise réseau à *centre de gravité centré* qui possède une firme pivot qui coordonne les différents participants à l'entreprise réseau ;

c) l'entreprise réseau *coopérative* qui ne possède aucun centre de gravité et met en œuvre des relations d'interdépendances entre les partenaires.

La seconde forme d'entreprise distribuée est l'entreprise virtuelle. Il s'agit d'un partenariat défini comme suit :

— Définition 2.10 : Entreprise virtuelle —

« *L'entreprise virtuelle peut être définie comme étant une agrégation temporelle de compétences et de ressources qui collaborent ensemble pour un besoin spécifique tel une opportunité d'affaire.* » [Goranson *et al., 1997*]
« *Quel que soit le type d'entreprise virtuelle, elle est formée d'un groupe de partenaires ayant des compétences respectives. Chaque partenaire est un spécialiste reconnu par ses pairs contribuant à la formation de l'équipe qui l'entoure.* » [Ettinghoffer, *1992*]

Toutes ces définitions d'entreprise virtuelle mettent en avant l'aspect temporel. Ainsi, elle peut être dissoute dès que le projet ayant mené à sa constitution est réalisé. Il s'agit d'une forme de partenariat plus opportuniste que les autres (entreprise réseau, ...), car il se fonde sur l'exploitation momentanée du marché [Couture et Loussararian, 1999]
Le troisième type d'entreprise distribuée nous semblant intéressante est l'entreprise étendue.

— Définition 2.11 : Entreprise étendue —

« *Le concept d'entreprise étendue est principalement fondée sur l'idée de considérer l'entreprise manufacturière traditionnelle et d'y incorporer les entreprises avec lesquelles elle entretient des relations dans le cadre de sa production.* » [Browne *et al., 1995*]
« *Le système manufacturier d'une entreprise est vu dans un contexte plus global qui englobe les relations de l'entreprise avec ses clients et fournisseurs.* » [Cloutier, *1999*]

L'entreprise étendue est la résultante de l'ensemble des systèmes d'information qui assurent les relations entre une entreprise et ses succursales, ou entre ses services répartis dans des zones géographiques éloignées, ou encore entre l'entreprise et ses sous-traitants.
Enfin, nous concluons la description des formes majeures d'entreprise distribuées par le réseau d'entreprises.

— Définition 2.12 : Réseau d'entreprise —

Ce sont des constructions coopératives à moyen et long terme qui, dans leur forme la plus achevée s'appuient sur l'intérêt mutuel et réciproque des partenaires en présence. [Nunes, *1994*]

[NUNES, 1994] fait la distinction entre deux types d'organisation pour les réseaux d'entreprises :

a) les réseaux centrés qui sont mis en place et dirigés par une firme centrale désirant mettre en œuvre des relations privilégiées avec ses fournisseurs et ses clients fidélisés ;

b) les réseaux fédérés qui ne possèdent aucune firme centrale soit parce qu'il y a instabilité d'un éventuel leader sur le long terme, soit parce qu'il y a un cadre institutionnel spécifique. Prenons comme exemple les groupements de petites et moyennes entreprises sous forme de groupements d'intérêt économique.

La notion de réseau d'entreprise peut être vue comme une généralisation du concept d'entreprise réseau défini ci-dessus.

En plus des systèmes de production classiques (*cf.* section 2.1.2), nous nous intéressons à ces évolutions des organisations d'entreprises. Nous verrons dans les sections suivantes quels sont les points qui attirent notre intérêt. Dans la suite de ce mémoire nous considérons que ces différentes formes d'entreprises appliquées au domaine manufacturier sont d'une manière générale des *systèmes distribués de production.*

2.1.3.2 Problèmes posés par les systèmes distribués

Les problèmes posés par les systèmes distribués de production, en plus de ceux hérités des systèmes classiques décrits dans la section 2.1.2.5, sont essentiellement la gestion de la confidentialité des informations. En effet dans le cadre d'entreprises réseaux ou de tout autre système distribué, les différents partenaires veulent et doivent pouvoir interdire la diffusion d'informations qui peuvent s'avérer vitales pour elles. Ce problème est un sous-ensemble d'un problème plus important qui est la définition de protocoles de coopération et d'échanges d'informations entre les différents membres du système distribué.

2.1.4 Simulation de systèmes manufacturiers

L'un des moyens que possède les gestionnaires des systèmes de production pour répondre aux problèmes posés par ceux-ci (*cf.* section 2.1.2.5) est l'utilisation de la simulation.

— Définition 2.13 : Simulation —

« *La simulation est un moyen explicatif pour définir un système, un vecteur d'analyse pour déterminer des résultats critiques, un évaluateur de conception pour analyser et évaluer des solutions proposées, ...* » *[* Law *et* Kelton*, 1991]*
« *La simulation est l'imitation dans le temps des opérations d'un processus ou d'un système réel. La simulation implique la génération d'une évolution "artificielle" du système, et l'observation de cette évolution pour réaliser des déductions sur les caractéristiques opérationnelles du système réel représenté.* » *[* Banks*, 1999]*
La simulation est une collection de méthodes et d'outils permettant et reproduire totalement ou partiellement le comportement de systèmes réels. Cette technique peut être utilisée par l'intermédiaire de logiciels appropriés (réseaux de Pétri, chaînes de Markov, Arena® *[* Kelton *et al., 1998],* Simple++®*, ...).*

La simulation est l'un des outils les plus efficaces à la disposition des concepteurs et des gestionnaires des systèmes industriels. Elle consiste à construire un modèle du système réel et à conduire des expériences sur ce modèle afin de comprendre son comportement et d'en améliorer les performances.
Il existe différents types de modèles. Les modèles *physiques* sont ceux dans lesquels le système réel est représenté par une réplique ou maquette, à une échelle différente et éventuellement à l'aide de matériaux différents. Les modèles *symboliques* sont une abstraction mathématisée de la réalité. Ils sont en général exécutés sur un calculateur, qu'il soit analogique ou digital.
Une autre distinction concerne la prise en compte des aléas dans le modèle. Dans certains cas, qualifiés de *déterministes*, leur influence est considérée comme négligeable. Le plus souvent, ils doivent être représentés car ils jouent un rôle significatif (pannes, ...). On a alors affaire à des modèles *stochastiques*.
Une troisième dichotomie sépare les modèles *statiques* et les modèles *dynamiques*. Dans les premiers, le temps n'intervient pas comme par exemple dans le cas d'un modèle comptable permettant de calculer un produit en fin d'exercice à l'aide d'un tableur. Dans les seconds, il est un facteur essentiel du comportement et de l'état du système (réacteur chimique régi par des équations différentielles, ...).
Enfin dans les modèles dynamiques, on distingue les modèles *discrets*, dans lesquels l'état du système change qu'à certaines dates (exemple d'une file d'attente devant une caisse), et les modèles *continus* ou ce changement est permanent comme par exemple dans le cas d'un réacteur chimique. Un modèle qui contient à la fois des composantes discrètes et continues est dit *mixte*.

Le choix de la méthode de modélisation dépend fortement de la nature du système réel et de sa représentation. La table 2.2 illustre la nature de ce choix.

Système	Modèle	Méthode
déterministe	déterministe	analyse numérique
stochastique	déterministe	calcul des probabilités
déterministe	stochastique	Monte Carlo
stochastique	stochastique	simulation

TAB. 2.2 – Choix d'une méthode de modélisation et d'exploitation

Pour traiter les modèles des systèmes industriels, nous nous situons dans le cadre de la modélisation [LAW et KELTON, 1991] :
- symbolique : nous désirons réaliser des modèles exécutables sur ordinateurs ;
- stochastique : la prise en compte de phénomènes aléatoires au sein d'un système de production est nécessaire (pannes, taux d'occupation, ...) ;
- dynamique : le temps est un facteur important pour la simulation d'un système de production. En effet, le comportement de ce dernier peut évoluer en fonction ou durant le déroulement du temps ;
- discrète : nous considérons que le système d'exploitation doit faire face à des évènements qui ont la caractéristique de se produire à n'importe quel moment et de changer l'état du système modélisé (arrivée d'un ordre de fabrication, arrivée de matière première, fin de fabrication d'un produit fini, ...)

Dans la suite de ce mémoire nous utiliserons abusivement le terme « simulation » comme un synonyme de *simulation à évènements discrets*.

2.1.5 Simulation de systèmes distribués

Face aux problèmes d'adaptation de la simulation aux systèmes parallèles et aux réseaux informatiques, le concept de simulation distribuée ou parallèle a émergé [FUJIMOTO, 1990a ; FUJIMOTO, 1993 ; LIN et FISHWICK, 1996] :

— DÉFINITION 2.14 : SIMULATION DISTRIBUÉE OU PARALLÈLE —

Ce type de simulation fait référence à l'exécution de programmes de simulation à évènements discrets sur des systèmes multi-processeurs ou sur des réseaux de stations informatiques. Son premier but est d'augmenter les performances des systèmes simulatoires via une exécution parallèle. Parfois la nature même de la simulation est distribuée comme, par exemple, pour les simulateurs militaires.

Il existe une différence sémantique entre la simulation distribuée et la simulation parallèle. Dans le premier, le but est de réaliser le processus de simulation au sein d'un réseau de stations informatiques. Dans le second cas il s'agit d'utiliser une machine parallèle (hyper-cube, machine multi-processeurs, ...). Dans la suite de ce mémoire nous nous concentrerons sur le premier type d'approche.

2.2 Problématique

Dans cette section nous présentons la problématique à laquelle nous nous attachons, c'est-à-dire la simulation distribuée de systèmes industriels complexes. Nous considérons quatre grandes catégories de problèmes rencontrés à la fois lors de la modélisation mais aussi lors du processus de simulation.

2.2.1 Formalisation

L'un des premiers problèmes rencontrés durant la phase de modélisation pour la simulation d'un système manufacturier est le manque de définition formelle des éléments constituant ces derniers.
Ainsi, les modèles de base de la simulation tels que les réseaux de Pétri [David et Alla, 1992 ; Wang, 1998] ou la théorie des files d'attentes [Knuth, 1998] permettent de modéliser convenablement un système industriel [Zimmermann, 1994]. Malheureusement, ils restent d'une grande complexité à maîtriser et à utiliser. Créer un modèle de simulation sur la base de ces concepts et de ces outils est du ressort de spécialistes. Du point de vue d'un industriel, l'investissement en moyens humains et financiers peut s'avérer considérable et irréalisable.
Pour résoudre ce problème essentiel, la communauté scientifique et des sociétés commerciales ont développé des outils de simulation dédiés aux systèmes industriels. Nous citerons par exemple les langages SIMAN [Pegden et al., 1995] ou QNAP [Veran et Potier, 1984], et les outils comme Arena® [Kelton et al., 1998] ou Simple++® [Tecnomatix®, 2001]. Tous ces outils offrent la possibilité de modéliser un système industriel à l'aide d'objets facilement appréhendables par les industriels (files d'attentes, zones de stockage, unités de traitements, ...)
Malheureusement ces outils de simulation influencent fortement la vision des concepteurs. Par exemple, les outils Arena® et Simple++® n'offrent pas la même vision quant à la modélisation : les éléments de modélisation sont différents. Ainsi, l'utilisation d'un outil ou d'un autre nécessite souvent la réécriture totale du modèle de simulation.
De plus la qualité des modèles de simulation (rapport entre la réponse aux besoins en simulation d'un modèle et sa complexité « syntaxique ») est fortement dépendante des compétences des concepteurs. Rares sont les règles définissant les

organisations structurelles (artefacts de modélisation représentant les éléments composant un système industriel) et sémantiques (règles de bonne construction d'un modèle de système industriel) acceptables pour les modèles de simulation. L'existence de ces règles de construction seraient utiles pour aider les concepteurs à définir et vérifier leurs modèles.
Le problème de la formalisation est partiellement résolu par l'existence de méthodologies et de langages de modélisation de systèmes industriels comme ASCI [KELLERT et FORCE, 1998a ; KELLERT et RUCH, 1998b], CM [NANCE, 1981], IDEF [US AIR FORCE, 1993a], UEML [VERNADAT, 2001], ... Mais elles permettent la modélisation de systèmes industriels qui ne considèrent que rarement les nouveaux types d'organisations. De ce fait, les modèles créés avec ces méthodes ne peuvent généralement pas être transposés en modèles de simulation distribués.

2.2.2 Mise en évidence des flux et sous-systèmes

Ce second problème peut se rapprocher de celui de la formalisation. Prenons l'exemple d'un modèle de simulation dont le mode de gestion est défini en flux poussé. Si le concepteur désire réaliser un modèle du même système mais cette fois en utilisant un mode de gestion en flux tiré, il devra réécrire totalement le modèle de simulation. Hors la description de l'infrastructure physique et informationnelle reste en majeure partie la même dans les deux modèles de simulation. Seule la spécification de la gestion du système a changé.
Ainsi, un système de production est parcouru par les flux d'entités physiques et les flux d'informations, et comprend un sous-système décrivant la gestion du système industriel. Ces flux et sous-système sont distincts même s'il existe de très fortes relations entre eux. Actuellement, les modèles de simulation incluent ces différentes parties sans toutefois les mettre en évidence. Cette vision du système de production ne va pas sans poser quelques problèmes. La compréhension du modèle de simulation reste difficile car il faut faire un effort souvent conséquent pour les différencier.
Une approche systémique comme celle proposée par [LE MOIGNE, 1977 ; LE MOIGNE, 1992] permettrait une modélisation plus modulaire. Malheureusement, les outils de simulation dédiés aux systèmes industriels ne prennent en compte que rarement une telle approche.

2.2.3 Décentralisation

Un problème majeur rencontré à la fois par les méthodologies et les outils de simulation est la centralisation des modèles et des processus de simulation.
Les méthodologies supportant la modélisation de systèmes industriels distribués restent encore rares et en forte évolution. En effet, la communauté scientifique est très

active dans ce domaine de recherche. Ainsi des méthodes comme EML [Vernadat, 1997], CIMOSA [AMICE, 1993] ou ARIS permettent une modélisation quasi complète d'une entreprise de type hybride (entreprises virtuelles ou groupements d'entreprises).
Malheureusement, les outils de simulation supportant ce type de système restent rares. Nous pouvons mentionner les travaux réalisés dans le domaine de la simulation distribuée comme HLA [US Department of Defense, 1996] ou ARéVi [Duval et al., 1997] ou des outils permettant la communication interprocessus comme CORBA [Mowbray et Zahavi, 1995 ; Siegel, 1996]. Malheureusement ces outils sont soit trop génériques (HLA, CORBA, ...), soit trop spécifiques (ARéVi est centré sur la réalité virtuelle). D'une part, il reste toujours difficile de mettre en œuvre ces outils. D'autre part, des problèmes plus spécifiques apparaissent dans le cadre de la simulation de systèmes distribués. Prenons l'exemple d'un groupement d'entreprises. Chaque membre, tout en voulant participer à la vie du consortium, désire ne pas diffuser l'ensemble des informations et des données qui font sa spécificité et sa puissance commerciale. Ainsi, ce problème de confidentialité doit absolument être supporté par les outils de simulation. De plus, les systèmes comme les entreprises virtuelles sont en perpétuelle évolution. Ainsi le groupe peut être dissout dès le projet terminé, ou un membre peut quitter ou intégrer le groupe au cours d'un projet. Ce dynamisme des structures et des relations entre les différentes parties d'un système distribué doit aussi être parfaitement supporté par les outils de simulation. Même si la communauté scientifique a déjà travaillé sur des problèmes de distribution comme la synchronisation des modèles [Filloque, 1992], les problèmes générés spécifiquement par les systèmes industriels distribués sont encore mal gérés.

2.2.4 Réutilisation

Un autre problème rencontré actuellement par les concepteurs de modèles de simulation est la faible modularité de ces derniers. En effet, même si les concepts de modèle et de sous-modèle sont souvent présents dans les outils de simulation, il reste souvent difficile de construire des modèles totalement modulaires. Par exemple, l'utilisation d'un sous-modèle existant impose souvent de réaliser une copie de celui-ci et de l'incorporer dans le nouveau modèle de simulation. Cette duplication, quoiqu'utile, ne permet pas de répercuter automatiquement une modification sur toutes les instances du sous-modèle. Il est nécessaire de modifier manuellement tous les modèles de simulation utilisant le sous-modèle. L'idéal serait dans ce cas de modifier une fois pour toute le sous-modèle, ces changements étant immédiatement et automatiquement pris en compte par l'ensemble des modèles utilisant ce sous-modèle. D'autre part, les sous-modèles créés sont souvent écrits en fonction des modèles qui les incluent. Ainsi les sous-modèles sont souvent dépendants de leur utilisation. D'un point de

vue industriel, une approche plus modulaire serait plus efficace. Ainsi des travaux comme ceux de [CHEN et SZYMANSKI, 2001] permettent de mettre en œuvre des modèles de simulation sur les bases d'une conception par composants. Il reste toutefois à vérifier si l'approche déjà proposée est adaptable à la simulation distribuée de systèmes industriels ou s'il est nécessaire d'en réaliser une nouvelle.

2.2.5 Synthèse

L'ensemble des problèmes mentionnés ci-dessus ont été soulevés afin de mettre en évidence les différents axes de recherche possibles. Notre but est de permettre la simulation de systèmes manufacturiers complexes tels que les réseaux d'entreprises, la modélisation de relations entre des donneurs d'ordres et des fournisseurs ou la simulation d'une entreprise fortement décentralisée. On remarquera que les cas qui nous intéressent incluent toujours un aspect de décentralisation ou de distribution. Nous noterons toutefois que nous ne nous limitons pas à l'aspect physique (la distribution informatique des modèles de simulation). En effet, nous considérons que la distribution de l'information et des processus de prise de décision sont deux aspects eux aussi importants.

2.3 Propositions

Pour résoudre les quatre problématiques exposées dans la section précédente, nous proposons de concevoir une approche méthodologique basée sur UML et les systèmes multi-agents [GALLAND et al., 1999]. Comme nous l'avons indiqué dans la section 2.2, de nombreux travaux tendent à résoudre partiellement les problèmes qui nous intéressent. Du point de vue de la simulation distribuée, la communauté scientifique a déjà donné naissance à des outils comme HLA [US DEPARTMENT OF DEFENSE, 1996] qui permet de faire communiquer plusieurs outils de simulation au sein d'un réseau informatique. Cette approche résout notamment une grande partie des problèmes inhérents à la simulation distribuée comme la synchronisation des modèles [FILLOQUE, 1992]. Du point de vue de la modélisation des systèmes distribués, de nombreux travaux existent comme par exemple la représentation des firmes dans un contexte international proposée par [BURLAT, 1996]. De plus, les outils existants sont fortement dépendants d'un domaine. Par exemple, ARÉVI se concentre sur la représentation virtuelle des systèmes [DUVAL et al., 1997 ; CHEVAILLIER et al., 1997] et SWARM est un environnement de simulation adapté aux systèmes de vie artificielle [BURKHART, 1994]. Enfin, certains outils ne tiennent compte que d'un aspect de la distribution des modèles de simulation. Prenons pour exemple HLA qui est une architecture permettant de faire communiquer des modèles de simulation distants mais, afin d'être utilisable dans la majeure partie des cas, se

limite exclusivement aux communications inter-modèles.

Notre approche méthodologique doit :

- proposer des éléments de modélisation adaptés aux systèmes industriels distribués et à leur simulation (machine, file d'attente, gamme, centre de décision, ...) ;
- permettre une modélisation efficace par le biais d'une approche systémique [Le Moigne, 1992] ;
- proposer un guide de modélisation et de simulation ;
- utiliser un environnement de modélisation autorisant éventuellement la modélisation distribuée ;
- utiliser un environnement de simulation acceptant la distribution informatique des modèles.

La conception d'une méthodologie restant un projet de grand envergure, nous nous proposons d'utiliser une approche itérative de développement. La figure 2.2 illustre ce point de vue en schématisant les deux premières étapes du développement de notre approche méthodologique. En effet, nous étoffons progressivement chaque axe majeur en étudiant un sous-problème à la fois. Les travaux qui sont exposés dans ce mémoire représentent la synthèse des deux premiers cycles de développement (*cf.* partie III).

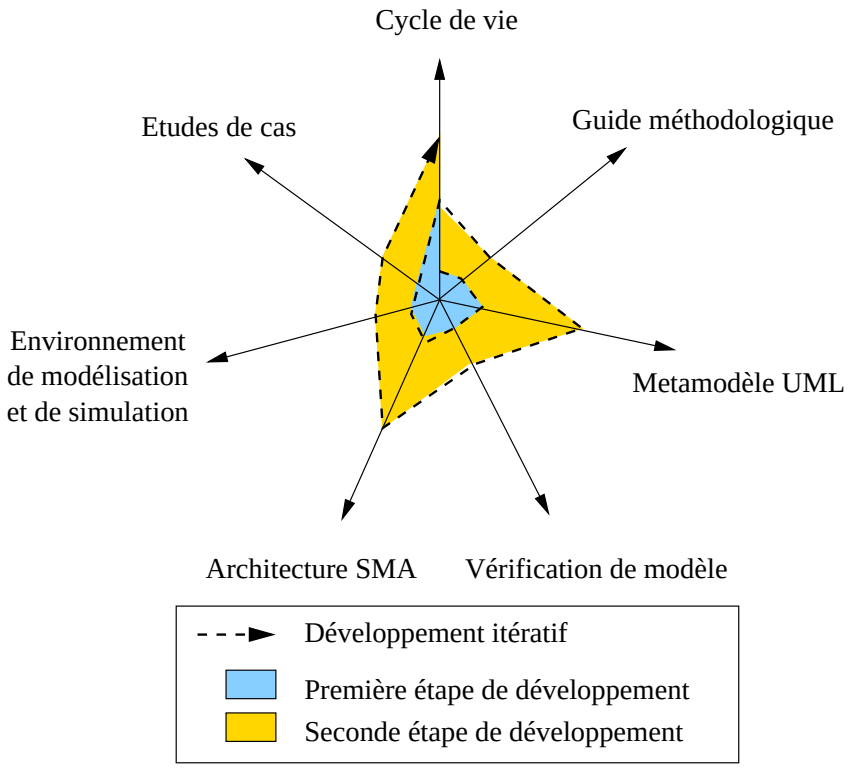


Fig. 2.2 – Développement itératif de $\mathcal{MAMA}$-$\mathcal{S}$

Dans les sections suivantes, nous présentons les choix de réalisation dégagés a priori

(avant une étude de domaine) et validés a posteriori (après une étude de domaine).

2.3.1 Cycle de vie des modèles

Le développement d'une approche méthodologique nécessite la mise en œuvre d'un cycle de vie pour les modèles de simulation. Nous nous proposons dans le cadre de nos travaux de choisir et d'étendre un cycle de vie existant afin qu'il supporte la modélisation et la simulation de systèmes industriels distribués. Dans la suite de ce mémoire nous exposons précisément l'ensemble de nos apports dans ce cadre.

2.3.2 « Unified Modelling Language »

Nous proposons l'utilisation de métamodèles UML (« Unified Modelling Language ») [MULLER, 1997 ; BOOCH et al., 1997]. Ces derniers permettent de définir les structures pouvant composer un modèle. Dans notre cas, nous définissons un métamodèle UML représentant les systèmes industriels distribués, c'est-à-dire une structure orienté-objet définissant les objets pouvant composer un modèle de système industriel distribué, ainsi que les relations et les associations entre ces objets. Cette approche possède les avantages d'une formalisation simple et claire des éléments de modélisation. Elle permet aussi, au travers des mécanismes de génération de code d'UML, la génération quasi-automatique de modèles de simulation exécutables. D'autre part, la définition sous forme de diagrammes d'objets et de contraintes d'intégrité permet une vérification simple et efficace de la cohérence des modèles de simulation. Enfin le dernier avantage d'UML est qu'il existe un grand nombre d'outils de modélisation supportant la métamodélisation UML. Ces outils permettent à partir d'un métamodèle quelconque de pouvoir éditer et modifier les modèles correspondant (Rational Rose®[1] , ARGO/UML [2] , ARAKHNÊ [3] , ...).

2.3.3 Systèmes multi-agents

Le support de la distribution au sein de notre approche méthodologique est réalisée pour une part essentielle par des systèmes multi-agents [FERBER, 1995]. Nous considérons que les agents cognitifs (agents ayant des capacités de réflexion et de planification) sont fortement adaptés à nos problématiques. En effet, si nous considérons les trois sous-systèmes proposés par [LE MOIGNE, 1992], les systèmes multi-agents (SMA) permettent :

– **pour le sous-système opérationnel :** le support de la décentralisation géographique des infrastructures physiques. Les agents échangent des lots de fabrication ;

[1] http://www.rational.com/products/rose/index.jsp
[2] http://argouml.tigris.org/
[3] http://arakhne.org/

– **pour le sous-système informationnel :** de distribuer l'information (gammes, nomenclatures, ...) partiellement ou totalement au sein d'un ensemble de modèles de simulation. Ceci est réalisé grâce aux capacités interactionnelles et cognitives des agents ;

– **pour le sous-système décisionnel :** de modéliser les processus de prise de décision d'un système industriel. En effet, comme l'ont illustré [Burlat, 1996] et [Kabachi, 1999], les agents peuvent être utilisés pour modéliser les différents acteurs au sein d'un processus de prise de décision et cela quelque soit leur niveau (tactique, stratégique ou opérationnel).

De plus les capacités organisationnelles des systèmes multi-agents nous permettent de faire face à la dynamique de création et de destruction des acteurs au sein d'un système industriel distribué.

2.3.4 Autre contrainte de réalisation

Notre approche méthodologique étant destinée à une utilisation industrielle, nous voulons tenir compte d'une particularité qui est la possession d'outils de simulation, ainsi que de l'expérience associée. Ainsi notre approche permet d'intégrer des outils de simulation comme Arena®. La figure 2.3 illustre un modèle d'un groupement de trois entreprises. Deux possèdent déjà un modèle de simulation Arena®, alors que le troisième est simulé, par exemple, via une société d'agents comme SWARM [Burkhart, 1994] ces trois modèles de simulation distincts peuvent communiquer grâce à une architecture multi-agents que nous proposons au sein de notre approche méthodologique.

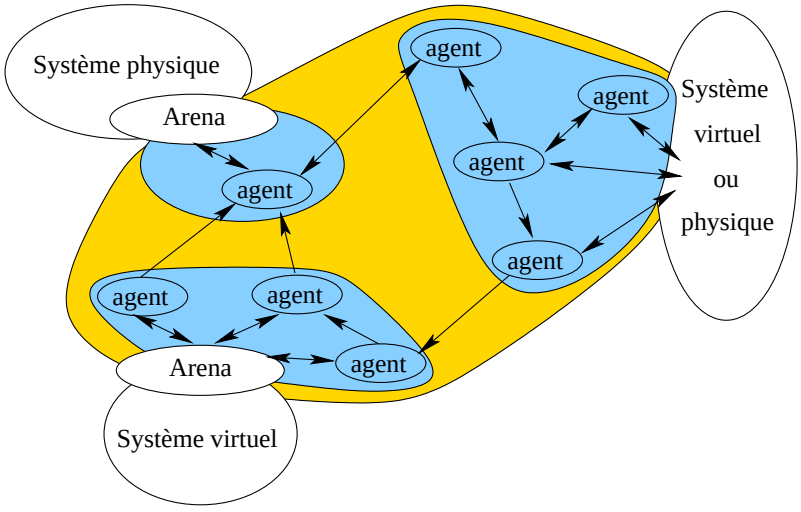


Fig. 2.3 – Exemple de modèle de simulation

2.4 Conclusion

Face à l'évolution des systèmes industriels vers plus de plus en plus de décentralisation, il est nécessaire que les outils et les méthodes pour les modéliser et pour simuler leurs comportements évoluent à leur tour. Face à ce problème, la communauté scientifique a réalisé des travaux intéressants tant du point de vue de la modélisation [VERNADAT, 2001 ; BURLAT, 1996 ; KABACHI, 1999], que du point de vue de la simulation [US DEPARTMENT OF DEFENSE, 1996]. Malheureusement, l'ensemble de ces travaux sont soit inadaptés aux systèmes industriels distribués, soit traitent un problème particulier de ces derniers. D'autre part, les outils de simulation existants supportent encore rarement la distribution informatique d'un modèle de simulation.
Pour résoudre les problèmes de formalisation (unification des paradigmes de modélisation) des systèmes industriels distribués, de modularité, de décentralisation et de réutilisation des modèles, nous proposons une approche méthodologique basée sur une modélisation systémique et l'utilisation de la métamodélisation UML ainsi que des systèmes multi-agents.
Si nous considérons les points de vue de la distribution et de la centralisation pour les aspects méthodologiques et simulatoire, nous pouvons décrire les apports de nos travaux vis-à-vis de l'existant.

- **simulation et conception centralisées :** Il s'agit de l'approche actuelle la plus courante. Elle consiste à centraliser les informations, les compétences ou les sous-modèles nécessaires pour la construction d'un modèle de simulation. Le processus de simulation est réalisé avec un seul outil de simulation tel qu'ARENA®, SIMPLE++® ou SIMAN. Notre méthodologie permettra une plus grande souplesse quant à l'utilisation de sous-modèles et au support de la simulation à base de composants. Elle permettra aussi une meilleure compréhension du modèle de simulation en permettant de distinguer clairement les différents flux composant le système manufacturier. Cette approche systémique nous permettra une plus grande modularité et réutilisabilité des modèles.
- **simulation centralisée, conception distribuée :** Il s'agit d'une approche semblable à la précédente, mais ici la phase de conception est répartie entre plusieurs concepteurs. En plus des problématiques citées précédemment s'ajoutent les difficultés inhérantes à tout travail collectif. Notre méthodologie pourrait apporter un support méthodologique quant à la mise en commun des modèles de simulation. Plus en aval, elle pourrait présenter un cadre formel quant aux possibilités de communication entre les différents concepteurs (uniquement d'un point de vue informatique).
- **simulation distribuée, conception centralisée :** Cette approche est utilisée

par des outils comme SWARM [Burkhart, 1994 ; Santa Fee Institute, 1994] ou ARéVi [Duval et al., 1997]. Elle consiste à créer un modèle décrivant le système de production. Ce modèle est composé d'entités distribuables. Les deux exemples cités ont la particularité d'utiliser des technologies à agents pour réaliser le processus de simulation. Notre approche méthodologique peut apporter un cadre formel permettant de faire communiquer des outils qui ne sont pas destinés à dialoguer ensemble. Cette approche est semblable à la vision développée par HLA [US Department of Defense, 1996].

– **simulation et conception distribuées :** Cette dernière approche est celle que nous désirons étudier. Elle consiste à rassembler tous les avantages de la modélisation collaborative et de la simulation distribuée. Les différents apports de notre méthodologie cités ci-dessus se retrouvent ici.

Dans la suite de ce mémoire nous présentons les points forts mais aussi les faiblesses des méthodes et des outils pour la modélisation et la simulation de systèmes industriels distribués. Nous commençons la partie II par la description des méthodes de modélisation d'entreprise (UEML, ...) en mettant surtout en avant leurs capacités à supporter la modélisation de systèmes industriels distribués. Dans la suite de cette même partie, nous décrivons les outils de simulation adaptés à notre problématique de simulatio. Enfin, après un bref exposé de la genèse des systèmes multi-agents, nous présentons une approche de modélisation appelée « Voyelles ».
La partie III contient les détails de nos propositions. Nous appelons notre approche $\mathcal{M}_{\mathcal{A}}\mathcal{M}\mathcal{A}$-$\mathcal{S}$ (« Multi-Agent Methodological Approach for Simulation »). Ainsi, nous présentons le cycle de vie d'un modèle de simulation qui est basé sur l'extension des approches plus classiques issues du génie logiciel et de méthodes de construction des modèles de simulation. Ensuite, nous spécifions les quatre étapes majeures de ce cycle en décrivant les objectifs de chacune d'entre elles, ainsi que la description pseudo-formelle sous forme de métamodèles Uml. De plus, nous décrivons l'architecture d'un système multi-agents capable de supporter la simulation de systèmes industriels distribués.

Deuxième partie

État de l'art

Chapitre 3

Introduction

Face à l'évolution de la complexité des organisations des entreprises, les outils et les méthodes existants de modélisation et de simulation de systèmes industriels montrent leurs faiblesses. Ainsi, des systèmes tels que les réseaux d'entreprises ou les entreprises virtuelles restent difficilement modélisables et simulables et posent de nombreux problèmes de conception, de modularité, et d'exécution de leurs modèles. Prenons l'exemple d'un groupement d'entreprises. Il est possible de réaliser un modèle d'un tel système avec les outils et les méthodes existants. Mais les contraintes imposées par cette conception sont parfois trop importantes. Le plus souvent, il est nécessaire de centraliser les informations pour construire un modèle unique. Ceci va à l'encontre du droit à la confidentialité des membres d'un groupement. D'autre part, les concepts et les artefacts de modélisation existants permettent parfois difficilement de réaliser le modèle des nouvelles organisations d'entreprises. Par exemple, la notion de modèle de simulation distant (c-à-d, le modèle d'un autre membre du groupement) n'est supportée par aucun artefact. Ainsi seuls les concepts « classiques » de la simulation sont supportés (machine, file d'attente, ...). Les méthodes de conception des modèles de simulation n'intègrent pas une approche distribuée de modélisation. En effet, quoi de plus naturel pour un membre d'un groupement de concevoir localement un modèle de son propre système et, par la suite, de l'intégrer dans le modèle du groupement. Ce dernier problème illustre aussi ceux d'interopérabilité pouvant exister entre les différents modèles des membres du groupement d'entreprises. Face à l'hétérogénéité des outils de simulation et de leurs modèles respectifs, il est difficile de mettre en place des protocoles de communication et d'échange d'informations. Comment s'assurer que les concepts manipulés par un modèle sont les mêmes que ceux utilisés par un autre modèle ?

Maintenant, supposons que les problèmes de conception ont tous été réglés. Le groupement d'entreprises est donc modélisé par un ensemble de modèles pouvant interagir et communiquer. Il reste toutefois à régler les problèmes spécifiques à l'exécution du processus de simulation. Tout d'abord, les problèmes de synchronisation des modèles.

En effet, il est nécessaire que les messages échangés par les divers modèles de simulation n'arrivent ni trop tôt, ni trop tard. D'autre part, ces messages doivent avoir la même sémantique pour l'émetteur et pour le récepteur. Enfin, un groupement d'entreprises est susceptible de fortement évoluer dans le temps. Il faut donc garantir la cohérence du modèle du groupement alors que certains de ses membres peuvent quitter ou intégrer à tout moment le groupement. Cette cohérence peut être gérée à la fois statiquement (les changements ne se déroulent qu'entre deux processus de simulation) et dynamiquement (les changements peuvent intervenir durant un processus de simulation).

Ainsi, nous venons de préciser un certain nombre de problèmes posés par la modélisation et la simulation de systèmes industriels distribués. Cette liste n'est évidemment pas exhaustive, mais fait mention de ce qui nous intéressent plus particulièrement dans la cadre de la problématique présentée dans le chapitre 2.

La modélisation en entreprise constitue une première réponse à plusieurs de ces problèmes. Ayant pour objectif l'étude et la modélisation d'entreprises, elle ne se focalise pas sur une partie ou une fonction de ces dernières. Ainsi elle intègre des connaissances provenant de domaines aussi variés que l'économie, la gestion de production ou la gestion des ressources humaines. De nombreuses méthodes ont vu le jour à la suite de travaux réalisés dans cette discipline. Parmi les méthodes les plus connues, citons CIMOSA [VERNADAT, 1998 ; AMICE, 1993] ou GRAI [DOUMEINGTS, 1984]. Malheureusement, la modélisation en entreprise doit faire face à pléthore de méthodes qui, bien trop souvent, utilisent des concepts identiques sous des termes différents, ou des concepts différents sous des termes identiques. Face à cette difficulté de constitution d'une base conceptuelle commune, le projet UEML [VERNADAT, 2001] propose de définir une ontologie et un langage de modélisation pouvant être utilisés par l'ensemble des méthodes et des outils de modélisation en entreprise.

Quand à la simulation, elle permet d'évaluer le comportement dynamique des entreprises (et plus particulièrement des systèmes industriels) à partir de modèles. Cette discipline est un complément idéal de la modélisation en entreprise. En effet, cette dernière ne permet pas d'avoir une évaluation fine des comportements des systèmes, elle doit faire appel à des techniques de recherche opérationnelle (programmation linéaire, ...), ou de simulation. Au sein de la discipline de la simulation, nous pouvons citer la simulation distribuée [DAMANI et GARG, 1998 ; LIN et FISHWICK, 1996] qui s'intéresse plus particulièrement à la distribution des modèles. Le problème de synchronisation de ces derniers a fait notamment l'objet de nombreux travaux [FUJII et al., 1999 ; FILLOQUE, 1992]. D'autre part, certains travaux récents s'intéressent à l'interopérabilité des modèles de simulation [US DEPARTMENT OF DEFENSE, 1996]. De nombreuses méthodes de modélisation pour la simulation de

systèmes industriels ont été proposées, comme par exemple la « Conical Methodology » [Nance, 1981 ; Nance, 1994a] ou « Analyse Spécification Conception Implantation » [Kellert et Ruch, 1998b ; Kellert et Force, 1998a]. Enfin, la simulation étant une technique très prisée par les entreprises, de nombreux outils ou langages ont vu le jour : Arena® [Kelton et al., 1998], Simple++® [Tecnomatix®, 2001], QNAP [Veran et Potier, 1984], ... Malheureusement, aucun de ceux-ci n'a fait l'objet du développement d'une véritable méthodologie.
Face aux travaux issus de la modélisation en entreprise et de la simulation, nous avons constaté qu'il reste très difficile de mettre en œuvre la conception et la simulation d'un modèle de système industriel distribué. En effet, si chaque méthode ou outil proposé répond à un ou plusieurs problèmes, chacun ou chacune ne peut être utilisé seul. Il s'avère donc nécessaire, de concevoir un environnement de modélisation et de simulation intégrant les méthodes et les outils existants, et supportant les nouvelles organisations d'entreprises. Dans les chapitres suivants, nous présentons la démarche qui nous a poussé à cette conclusion. Ainsi nous avons choisi des méthodes issues de la modélisation en entreprise (*cf.* chapitre 4) et de la simulation (*cf.* chapitre 5) qui nous permettra de répondre partiellement à notre problématique. Enfin, avant de conclure cette partie, nous présentons les systèmes multi-agents [Ferber, 1995 ; Weiss, 1999]. En effet, nous montrerons qu'ils sont très adaptés au support de la simulation de systèmes industriels distribués. Nous présentons notamment l'approche de modélisation « Voyelles » [Demazeau, 1997 ; Demazeau, 1995 ; Boissier, 1999] que nous utilisons pour concevoir les systèmes multi-agents que nous proposons dans la partie III.

Chapitre 4

Modélisation en entreprise

Face à l'évolution de la complexité des organisations des entreprises, la communauté scientifique propose un ensemble de travaux permettant de modéliser et de comprendre le fonctionnement de ces systèmes. Ces diverses techniques permettent ainsi aux décideurs industriels de choisir des politiques de gestion adaptées à l'optimisation des critères de performance qu'ils désirent mettre en avant (optimisation des coûts de production, réduction des délais de fabrication, ...).
Dans ce chapitre, après une vue d'ensemble, nous présentons certaines des méthodes qui nous paraîssent les plus intéressantes concernant la modélisation en entreprise. Toutefois il ne s'agit pas d'un état de l'art exhaustif, et nous nous limitons exclusivement aux méthodes qui intègrent une partie de notre problématique ou qui sont fortement utilisées dans le cadre de nos travaux de recherche. Mais avant d'aborder ces méthodes nous présentons, dans une première section, une vue d'ensemble de la modélisation en entreprise.

4.1 Vue d'ensemble

La modélisation en entreprise est une discipline encore jeune qui regroupe l'ensemble des expériences et des connaissances, informelles ou non, sur la modélisation en entreprise. Son objectif est de permettre une meilleure compréhension des mécanismes participant au fonctionnement de ces systèmes. Cette amélioration des connaissances permettra de réaliser des systèmes de pilotage capables de gérer des systèmes de plus en plus complexes, mais aussi de prévoir leurs évolutions au cours du temps.
Le concept d'entreprise ayant déjà fait l'objet de nombreux travaux dans de nombreux domaines, la modélisation en entreprise possède une nature pluridisciplinaire. Les concepts manipulés proviennent d'un ensemble très vaste au sein duquel nous pouvons citer la gestion de production, les ressources humaines, l'économie, le droit, ... Toutefois, un ensemble de concepts semble être manipulé par tous dans la modélisation en entreprise (activité, processus, tâches, ...). Issus de la modélisation

fonctionnelle et des systèmes à évènements discrets, ils sont à la base de la plupart des formalismes et des langages utilisés pour la modélisation en entreprises [GRP GT 5, 1998]. Toutefois, même si ces concepts sont couramment acceptés, tous n'en ont pas la même définition. Pour répondre à ce problème, un groupe de travail veut réaliser les spécifications d'une norme définissant l'ensemble des concepts. Ainsi « les prénormes CEN ENV 40 003 et ENV 12 204 précisent la terminologie et énoncent les principes fondamentaux sous-jacents au domaine de la modélisation en entreprise » [VERNADAT, 1998].

Nous utilisons aussi la définition de [GRP GT 5, 2000] pour représenter le concept de méthodologie :

DÉFINITION 4.1 : MÉTHODOLOGIE

Une méthodologie est composée d'une méthode de modélisation et de modèles qui s'appuient sur des outils de représentation.

La notion de processus possède un certain nombre de définitions. Elles s'accordent toutes sur le fait qu'un processus est un ensemble de phases définies généralement comme des activités :

DÉFINITION 4.2 : PROCESSUS

« *Un processus est une séquence partiellement ordonnée d'étapes (sous-processus ou activités), déclenchée par un évènement pour atteindre un but fixé.* » *[* VERNADAT, *1999]*

« *Le processus est une combinaison d'activités, mobilisant des savoir-faires multiples se déroulant dans le temps, et finalisé par un objectif.* » *[* EL MHAMEDI *et al., 1997]*

« *Un processus est une succession d'activités qui produisent une valeur pour le client. En fait, il s'agit de ce que le client voit, de ce qu'il perçoit, de ce qu'il juge.* » *[* JACOB, *1995]*

Les activités entrent dans une grande part dans la constitution d'un processus. Elles correspondent à une action de transformation au sein de l'entreprise. Elles sont définies comme suit :

— Définition 4.3 : Activité —

« Une activité transforme un état d'entrée (objet physique ou informationnel) en un état de sortie, sous l'influence d'objets de contrôle. Cettre transformation est susceptible de concerner les caractéristiques physiques (fabriquer), spatiales (transporter), temporelles (stocker) de l'état d'entrée. L'activité mobilise des ressources ou moyens de production (homme ou machine) qui ne subissent pas cette transformation. » [El Mhamedi *et al., 1997]*
« Une activité est l'accomplissement d'une tâche. Il s'agit en général d'une séquence d'opérations devant être exécutée en totalité par une ou plusieurs ressources et ceci dans un temps donné pour réaliser la tâche spécifiée. L'activité est une étape élémentaire d'un processus. » [Vernadat*, 1999]*

Comme le mentionne François Vernadat, une tâche entre dans la définition d'une activité. Nous utilisons la définition de [GRP GT 5, 2000].

— Définition 4.4 : Tâche —

La tâche est un but donné dans des conditions déterminées. Elle indique ce qui est à faire, (alors que) l'activité (décrit) ce qui se fait. La notion de tâche véhicule avec elle l'idée de prescription, sinon d'obligation.

À partir de ce tour d'horizon, nous pouvons présenter un ensemble de méthodes et de langages couramment utilisés pour la modélisation en entreprise [Boutin, 2001].
Leur intérêt est essentiellement dû à la possible intégration de certains de leurs concepts au sein de notre démarche de modélisation. Ainsi, nous rappelons que nous désirons pouvoir modéliser un système industriel distribué, et cela en utilisant une approche systémique. Cet objectif nous permet de mettre en avant certains des langages et des méthodes de modélisation en entreprise.
Les langages IDEF sont très utilisés au sein du monde industriel. Ainsi, IDEF0 propose une approche de modélisation des activités de l'entreprise. IDEF1 est un langage intéressant par son objectif de modéliser le sous-système informationnel de l'entreprise. Enfin IDEF3 est un langage de modélisation d'un niveau supérieur à IDEF0 (niveau processus).
Certaines méthodes de modélisation informatiques sont utilisées dans le cadre de la modélisation en entreprise. Elles sont intéressantes, car elles offrent un grand nombre d'artefacts de modélisation quant aux sous-systèmes informationnels ou à la description des activités d'une entreprise. Ces méthodes se limitent toutefois au niveau informatique.

Ensuite nous présentons des méthodes spécialisées dans la modélisation en entreprise : GRAI, CIMOSA et PERA. Elles s'attachent à décrire l'entreprise dans sa globalité tout en offrant des points de vue différents.

Enfin, avant de conclure ce chapitre, nous présentons UEML. Il s'agit d'un effort de normalisation des concepts et des artefacts de modélisation des entreprises. Il est très intéressant de par sa nature normative qui peut se rapprocher de nos objectifs quant à la modélisation pour la simulation.

4.2 Langages IDEF

IDEF (« Integrated computer aided manufacturing DEFinition language ») est une technologie standardisée développée par l'U.S. Air Force [US AIR FORCE, 1993a]. Elle permet de définir l'architecture de systèmes industriels, et est couramment utilisée par les industriels dans le cadre de l'analyse d'entreprise, de la définition de processus, et de la modélisation de processus ou d'activités. Le Département de la Défense des États-Unis (DoD) utilise particulièrement les méthodes IDEF durant les initiatives de re-modélisation des processus industriels et de leur validation.

IDEF est un ensemble de méthodes composé entre autre de :
- IDEF0 : modélisation basée sur les activités,
- IDEF1 : modèles informationnels,
- IDEF1x : modélisation de structures de données,
- IDEF2 : modèles pour la simulation,
- IDEF3 : saisie de descriptions de processus,
- IDEF4 : conception orientée objet,
- IDEF5 : saisie de descriptions d'ontologies,
- IDEF6 : saisie de rationalités conceptuelles,
- IDEF7 : méthode d'audit pour les systèmes d'information,
- IDEF8 : modélisation d'interfaces utilisateurs,
- IDEF9 : spécifications de la conception dirigées par scénarios des systèmes d'information,
- IDEF10 : modélisation d'architectures d'implantation,
- IDEF11 : modélisation d'artefacts informationnels,
- IDEF12 : modélisation d'organisations,
- IDEF13 : conception de formalismes tri-schémas,
- IDEF14 : conception de réseaux.

Les langages les plus connus sont IDEF0, IDEF1 et IDEF3.

4.2.1 IDEF0

IDEF0 [US Air Force, 1993b] est une méthode basée sur « Structured Analysis and Design Technique® » [Lissandre, 1990]. Son objectif est de construire un modèle des activités de l'entreprise. Elle comporte non seulement la définition d'un formalisme graphique mais aussi la description d'une méthodologie de développement des modèles.

L'application d'IDEF0 consiste en la construction d'une série hiérarchique de diagrammes, de textes et de glossaires par l'intermédiaire de deux composants : les fonctions (représentées par des boites) et les données et objets reliant les fonctions (représentées par des flèches).

Le langage de modélisation d'IDEF0 possède les caractéristiques suivantes :

- il est expressif, compréhensif et capable de représenter une grande variété d'opérations commerciales ou manufacturières.
- c'est un langage cohérent et simple permettant une expression rigoureuse, précise et non ambiguë.
- il permet une meilleure communication entre les différents acteurs (analystes, concepteurs et utilisateurs) en limitant les besoins d'apprentissage à un seul langage et en utilisant une présentation hiérarchique des niveaux de détail.
- il a fait l'objet de validations complètes.
- ce langage peut être généré automatique par un outil de modélisation graphique.

En plus d'un langage, IDEF0 préconise l'utilisation d'une méthodologie de construction et d'interprétation des modèles.

IDEF0 est une technique de modélisation basée sur une combinaison de graphiques et de textes. Elle permet la spécification d'un cahier des charges, supporte l'intégration des activités et fournit les outils nécessaires aux changements potentiels des systèmes modélisés. Un modèle IDEF est composé d'une suite hiérarchique de diagrammes permettant la description par niveau de détail croissant des systèmes et de leurs interfaces dans le contexte du système modélisé.

IDEF0 est composé de trois types de diagrammes :

- le diagramme *graphique* définit les fonctions et les relations fonctionnelles en utilisant une syntaxe composée de boîtes et de flèches ainsi que la sémantique associée ;
- le *texte* et le *glossaire* fournissent des informations complémentaires concernant la représentation graphique.

Lorsqu'elle est utilisée de manière systématique, IDEF0 est une approche d'ingénierie des systèmes qui :

- met en place des mécanismes d'analyse et de conception durant tous les niveaux du développement ;
- produit de la documentation servant à l'intégration de nouveaux systèmes ou à la validation de ceux existant ;

– peut servir de vecteur de communication entre les analystes, les concepteurs, les utilisateurs et les gestionnaires ;
– permet l'archivage des consensus d'équipes de modélisation ;
– produit des outils de gestion de projets grands et complexes, en utilisant une mesure des progrès qualitative ;
– fournit une architecture de référence pour l'analyse d'entreprise et permettant la gestion des ressources et des informations.

Les composants acceptés par la syntaxe sont les boîtes, les flèches, les règles et les diagrammes. Les boîtes représentent les fonctions, les activités, les processus ou les transformations. Les flèches représentent les interactions de la fonction avec son environnement.

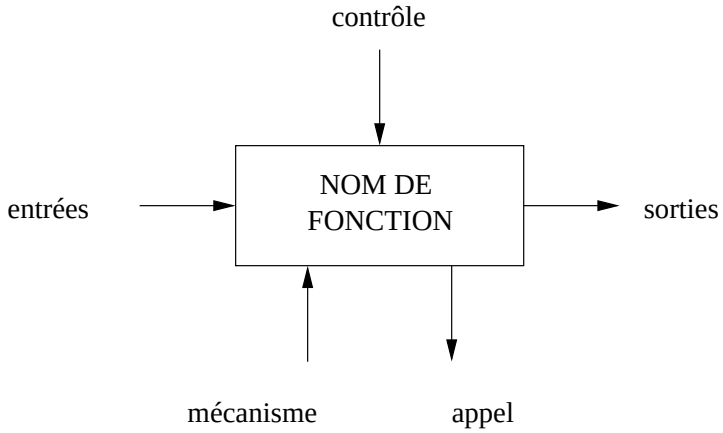


FIG. 4.1 – Une activité dans IDEF0

Comme l'illustre la figure 4.1, chaque côté d'une boîte possède une signification particulière dans le cadre des relations entre les boîtes et les flèches. Les flèches arrivant sur le côté gauche représentent les entrées. Ces dernières sont utilisées ou modifiées par la fonction pour produire les sorties (flèches sortant à droite). Le côté supérieur correspond aux conditions de contrôle nécessaires pour que la fonction réalise correctement son rôle. La partie inférieure d'une boîte est associée aux mécanismes (moyens pour réaliser l'activité).

4.2.2 IDEF1

IDEF1 [MAYER, 1992] est une méthode développée initialement dans le cadre du projet « Integrated Computer Aided Manufacturing » de l'U.S. Air Force. IDEF0, IDEF1 et IDEF2 forment la base d'une « architecture » de modèles permettant l'accomplissement des objectifs de production de systèmes ou environnements industriels.

Alors qu'IDEF0 et IDEF2 se focalisent respectivement sur les aspects fonction-

nels et temporels (simulation), IDEF1 a pour rôle la modélisation informationnelle nécessaire pour le bon déroulement des fonctions du système industriel.
La méthodologie de modélisation IDEF1 incorpore les principes de base permettant de produire des modèles informationnels. Ses objectifs sont de construire un modèle d'information intégré, concevoir une base de données pour ce dernier, et implanter et installer la base de données, les fonctions associées et les composants procéduraux.
IDEF1 définit un ensemble de procédures et de règles permettant de créer le modèle informationnel. Celles-ci utilisent des formalismes graphiques et textuels et le concept de formulaire. IDEF1 fournit les outils nécessaires au suivi et au contrôle du développement du modèle.
Parce que la modélisation induit un processus d'évolution, la méthode IDEF1 est organisée en un ensemble d'étapes ayant des résultats mesurables. Elle fournit la possibilité d'utiliser le concept de modularité qui permet de limiter les incomplétudes, les imprécisions et les inconsistances.
Un modèle informationnel est composé de deux éléments fondamentaux :

– **les diagrammes :** contiennent les caractéristiques structurelles du modèle présenté en accord avec l'ensemble des règles et des procédures de construction d'une représentation de l'information. La figure 4.2 illustre le diagramme décrivant les relations entre les entités composant le système d'information modélisé ;
– **les dictionnaires :** correspondent à la signification de chaque élément du modèle décrit par l'intermédiaire d'un texte condensé et d'indices qui définissent clairement les informations représentées par le modèle.

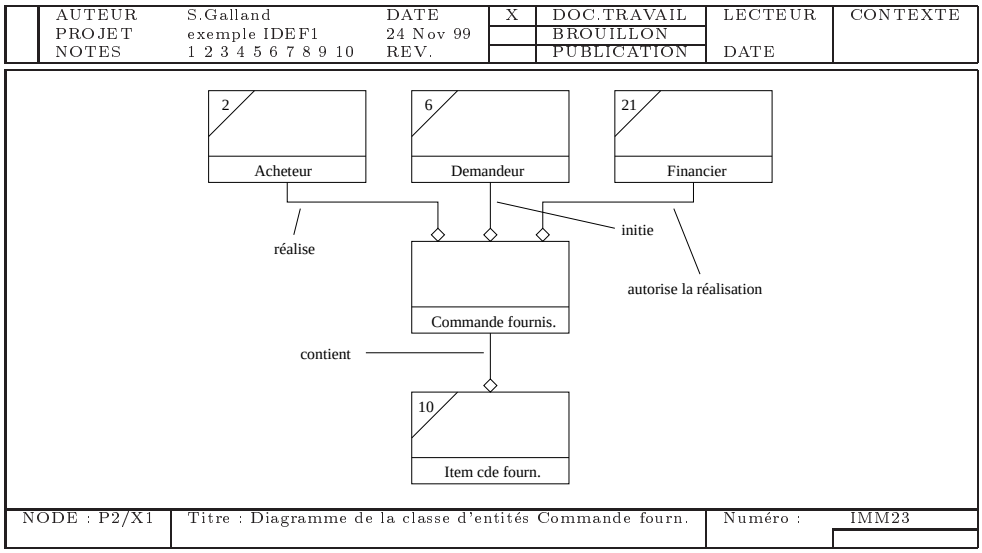


Fig. 4.2 – IDEF1 – Diagramme de classes d'entités

Un modèle d'information IDEF1 est un reflet d'une entreprise manufacturière et

fournit une définition de base des besoins informationnels de cette organisation. Elle assure la possibilité de distribution de l'information et l'intégration du système informationnel dans l'entreprise.
Le développement du modèle informationnel est composé de cinq phases :

– **phase 0 :** durant cette phase, le domaine et les objectifs du modèle sont dégagés ;
– **phase 1 :** l'objectif est de définir les entités apparentes ;
– **phase 2 :** les relations entre les entités sont dégagées ;
– **phase 3 :** le but est de définir les clés pour chaque entité ainsi que leurs attributs ;
– **phase 4 :** les attributs non-clés sont définis et sont associés à des entités.

Le processus de développement d'un modèle informationnel est, par nature, itératif. Le modèle évolue progressivement grâce à la succession des cinq étapes.
Il existe trois types évidents d'activités cycliques dans la méthode IDEF1 : cycle de collecte des données, cycle de validation, cycle d'acceptation par des experts. Chacun d'entre eux peut exister plusieurs fois durant la vie du projet de développement.
Le cycle de collecte des données est initié durant la phase 0. Son objectif est l'établissement d'une base documentaire à partir de laquelle toutes les informations pour la construction du modèle pourront être tirées. Il n'est pas improbable que les modélisateurs reviennent durant les phases suivantes sur les sources de cette documentation afin d'éclaircir des points particuliers.
Durant les différentes phases de la modélisation, les auteurs ou modélisateurs doivent faire face à plusieurs revues de la part des lecteurs. Les commentaires rédigés par ces derniers doivent être incorporés dans le modèle. Ce processus est répété tant que le résultat escompté n'est pas atteint. Ces itérations sont le fondement du cycle de validation (appelée aussi « IDEF Kit Cycle »).
Le cycle d'acceptation est le moment où un ensemble d'experts évalue le modèle d'information (final ou en évolution). Il détermine s'il correspond aux objectifs attendus. Typiquement, le cycle d'acceptation se produit plusieurs fois durant le déroulement du projet. Il est réalisé en général à la fin d'une phase mais plus rarement à la fin de chaque étape. Quoiqu'il arrive, une revue d'acceptation doit avoir lieu à la fin de la phase 4.

4.2.3 IDEF3

IDEF3 [US AIR FORCE, 1993c] est une méthode de modélisation graphique basée sur la description de processus. Cette description est réalisée par des diagrammes de flux, complétés par des documents d'information. Les diagrammes de flux sont composés à partir de quatre éléments :

– les *unités de comportement* (UDC) qui représentent toute entité ou artefact pouvant être produit par le système [SANDOVAL, 1994]. Elles sont représentées par

un rectangle divisé en trois zones (*cf.* figure 4.3) : son nom, son niveau de détail dans la décomposition hiérarchique, et le numéro d'une éventuelle activité IDEF0 associée ;
- les liens utilisés pour relier les UDC (*cf.* figure 4.3) ;
- les connecteurs logiques (ET, OU, et OU exclusif) ;
- les références qui est « un terme propre à IDEF3 pour permettre de faire référence à une partie du modèle (ou à un autre modèle). » [Vernadat, 1999].

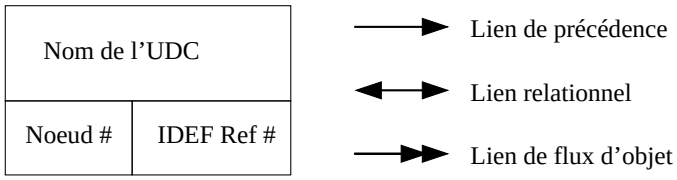


Fig. 4.3 – IDEF3 - Constructions de base

Même si les concepts de base d'IDEF3 proposent un grand pouvoir expressif, les modèles obtenus sont de nature qualitative et ne permettent donc pas l'analyse et l'optimisation des processus modélisés. Pour ce faire, il faut transformer les modèles IDEF3 en modèles plus formels comme les réseaux de Pétri [El Mhamedi et al., 1997].

4.3 Méthodes informatiques

La discipline informatique du Génie Logiciel et de la conception de systèmes d'information (SI) a donné naissance à un grand nombre de concepts et de méthodes. Même si à l'origine celles-ci ont été conçues dans un cadre uniquement informatique, leur utilisation a été étendu à la modélisation en entreprise.

4.3.1 La méthode Merise

Tout d'abord, prenons l'exemple de Merise (Méthode d'Étude et de Réalisation Informatique pour les Systèmes d'Entreprise) [Tardieu et al., 1985]. Cette méthode, créée en 1978 sous l'impulsion du ministère de l'industrie français, s'inspire des méthodes existantes sur la conception de systèmes de base de données, la conduite de projet, la programmation structurée et l'analyse modulaire des systèmes. « Son objectif est de fournir, à la fois, une philosophie, une démarche, des modèles, des formalismes et des normes pour concevoir et réaliser un système d'information » [Pierreval, 1990].
Les avantages de Merise sont multiples et dépendants du point de vue selon lequel nous nous plaçons [Gabay, 1998].

Selon le point de vue des méthodes de conception de SI, « Merise est une approche globale du SI menée parallèlement et simultanément sur les données et les traitements ». Elle correspond à « une description du SI par niveaux (conceptuel, organisationnel et opérationnel) qui constituent le *cycle d'abstraction* ». Merise est « une description du SI utilisant un formalisme de représentation précis, simple et rigoureux, pour la description des données (...) : le modèle *entité-relation* ».
Selon le second point de vue des démarches méthodologiques de développement de systèmes d'information, Merise propose « un découpage du processus de développement en quatre étapes : étude préalable, étude détaillée, réalisation et enfin mise en œuvre. Ce découpage, repris et normalisé par l'AFNOR (norme Z67-101 : recommandations pour la conduite de projets informatiques), correspond au *cycle de vie* d'un système d'information, et l'ensemble des résultats produits à chaque étape constitue le *cycle de décision* ». Merise est aussi « une description détaillée de la structure de travail à mettre en place pour mener à bien le développement du système d'information ».
Nous ne décrirons pas les détails de l'approche Merise qui sont plus largement abordés dans des ouvrages comme [TARDIEU et al., 1985].

4.3.2 « Unified Modeling Language »

À la fin de 1994, Jim RUMBAUGH (méthode OMT) et Grady BOOCH (méthode BOOCH), constatant que l'évolution parallèle des méthodes objets ne faisait plus progresser la technologie, décident d'unifier leurs travaux au sein d'une méthode unique : la méthode unifiée (« the Unified Method » ou UM). Une année plus tard, Ivar JACOBSON (créateur des cas d'utilisation ou « use cases ») rejoint ce groupe.
La première version de la méthode unifiée a permis de recueillir plus d'un millier de commentaires détaillés de la part de la communauté des utilisateurs. La principale demande est la mise à disposition d'artefacts de modélisation objet et non pas de processus de modélisation de ceux-ci. En 1996, la méthode unifiée devient UML (« Unified Modeling Language »), un langage de modélisation objet. En 1997, sous l'impulsion de grandes entreprises (DEC, HP, i-Logix, ...), la version 1.0 d'UML a fait l'objet d'une procédure de standardisation auprès de l'OMG (« Object Management Group ») [BOOCH et al., 1997]. Depuis lors UML est considéré comme un standard dans le domaine des langages de modélisation et de spécification orienté-objet.
La notation UML se concentre sur la description des artefacts du développement de logiciels plutôt que sur la formalisation du processus de développement lui-même. Cette notation est générique, extensible et configurable par l'utilisateur. L'une des caractéristiques de la définition d'UML est l'utilisation récursive de la notation UML pour décrire les concepts utilisés par ce langage. Ce choix pratique pose cependant le problème de « l'oeuf et de la poule ». C'est pourquoi des concepts de base permet-

tant de comprendre la spécification d'Uml sont définis. Ces concepts sont issus des technologies à objets : classes, objets, interfaces, associations, héritage, agrégation, ...
Le métamodèle Uml, c'est-à-dire le modèle des modèles, permet de décrire les contraintes structurelles et sémantiques des modèles développés par les utilisateurs. Uml propose par défaut un ensemble de métamodèles qui définissent ses différents langages (diagrammes de classes, cas d'utilisation, ...). Ce principe est très intéressant, car il permet de spécifier relativement facilement un nouveau langage de modélisation. D'autre part, en plus de l'extension du métamodèle à proprement parlé, Uml propose un ensemble de mécanismes capables d'étendre la notation Uml au niveau des modèles mêmes (stéréotypes, ...).

4.4 La méthode GRAI

La méthode GRAI [Marcotte, 1995 ; Doumeingts, 1984] est basée sur la théorie des systèmes de [Le Moigne, 1977 ; Le Moigne, 1992] et sur la théorie des systèmes hiérarchisés [Mezarovic et al., 1970]. Cette méthode propose un ensemble de modèles ou formalismes : le modèle conceptuel GRAI, la grille GRAI et les réseaux GRAI.

4.4.1 Le modèle conceptuel GRAI

La méthode GRAI permet de représenter le système de production par un modèle conceptuel de référence. Ce modèle est composé de deux modèles : le modèle conceptuel de référence du système de production, et le modèle conceptuel de référence d'un centre de décision décrivant la structure décisionnelle du système de production [Doumeingts, 1984].

4.4.1.1 Modèle conceptuel de référence du système de production

Ce système se décompose en deux sous-systèmes : le sous-système physique de production et le sous-système de gestion de production. Ce dernier est à son tour décomposé en un sous-système décisionnel et un sous-système informationnel. La figure 2.1(b) page 10 illustre la structure et les relations possibles entre ces différents sous-systèmes.

4.4.1.2 Modèle conceptuel de référence d'un centre de décision

Le modèle conceptuel de référence d'un centre de décision correspond à la vision macroscopique de la structure décisionnelle d'un système de production. Il décrit l'ensemble des relations existantes entre un centre de décision (CD) et d'autres CD.

Le centre de décision possède des connaissances sur *ce qu'il doit faire*, *sur quoi* il peut agir pour atteindre son objectif, ainsi que *jusqu'où* et *comment* il peut agir. Ainsi, à partir des informations lui provenant du sous-système physique, il met en œuvre un processus de décision et émet une décision vers les niveaux inférieurs.
La grille GRAI et les réseaux GRAI [DOUMEINGTS, 1984] permettent de représenter le système décisionnel GRAI. Ces formalismes ont pour objectif de représenter les concepts contenus dans le modèle conceptuel de référence [MARCOTTE, 1995]. Nous présentons ces deux formalismes dans les sections suivantes.

4.4.2 La grille GRAI

La grille GRAI est la représentation graphique de la structure décisionnelle périodique du système de pilotage. Elle permet de mettre en évidence les centres de décision et leurs interactions, leurs structures, les liens décisionnels et informationnels entre les centres, et les informations externes et internes.
Elle est constituée de fonctions qui regroupent les centres de décision pilotant des activités de même nature, et est décomposée en niveaux décisionnels définis par un couple ⟨`horizon`,`période`⟩, et correspondant aux niveaux stratégique, tactique et opérationnel.
La grille GRAI propose trois fonctions de base :

- **planifier :** elle vise à déterminer le plan de réalisation des produits demandés compte tenu des matières approvisionnées et des ressources affectées à la fabrication, et elle assure la synchronisation et la coordination des différentes fonctions ;
- **gérer les produits :** cette fonction regroupe les activités de gestion interne des pièces et d'achat externe de matières et de composants ;
- **gérer les ressources :** elle optimise l'utilisation des compétences des personnels et des machines, conformément aux stratégies de l'entreprise, et dans un objectif général de maîtrise des coûts.

4.4.3 Les réseaux GRAI

Les réseaux GRAI ont pour objectif la description détaillée des activités d'un centre de décision identifié dans la grille GRAI. L'élément de base d'un réseau est l'activité. Une activité est un processus de transformation réalisé avec un certain nombre de supports, un ou plusieurs déclencheurs et produisant un résultat. Il existe deux types d'activités : les activités de décision et d'exécution.
L'activité de décision est caractérisée par ses objectifs, ses variables de décision, ses contraintes et ses critères. L'objectif est le résultat à atteindre par le système. Les variables de décision sont les éléments mis en œuvre pour atteindre les objectifs. Les

contraintes représentent les limites de fonctionnement des variables de décision. Les critères correspondent aux fonctions à optimiser et permettent de choisir entre les différentes variables de décision pour atteindre les objectifs [Marcotte, 1995].

4.5 CIMOSA

CIMOSA (Computer Integrated Manufacturing - Open System Architecture) est une architecture de conception des systèmes intégrés de production. Elle a été développée dans le cadre du projet ESPRIT par le consortium AMICE [AMICE, 1993].

4.5.1 L'architecture CIMOSA

Cette architecture comprend un cadre de modélisation, une plate-forme d'intégration et une méthodologie d'intervention [Vernadat, 1998 ; Vernadat, 1999]. Le cadre de modélisation, appelé « cube CIMOSA » est illustré par la figure 4.4 [Boutin, 2001].
Il s'articule autour de trois principes fondamentaux et orthogonaux. Les trois axes du cube sont :

- **axe de généricité :** il se compose de trois niveaux :
  - un niveau *générique* qui correspond à la définition des primitives de base du langage de modélisation ou « constructs »,
  - un niveau *partiel* qui comprend les structures prédéfinies et réutilisables pour un domaine d'application,
  - un niveau *particulier* qui inclut les modèles spécifiques de l'entreprise.
- **axe des modèles :** il est aussi appelé axe de dérivation. Il définit trois niveaux de modélisation :
  - un niveau de *définition des besoins* durant lequel un cahier des charges est rédigé,
  - un niveau de *spécification de conception* qui est l'analyse conceptuelle des solutions aux besoins exprimés,
  - un niveau de *description de l'implantation* qui correspond à la description précise de la solution retenue.
- **axe des vues :** ou axe de génération qui définit l'entreprise selon quatre vues :
  - les *fonctions* qui décrivent les fonctionnalités et le comportement de l'entreprise en termes de processus, d'activités et d'opérations,
  - les *informations*,
  - les *ressources* qui décrivent les moyens à mettre en œuvre pour réaliser les fonctions de l'entreprise,
  - l'*organisation* qui correspond à la description des responsabilités et de l'autorité dans la prise de décision.

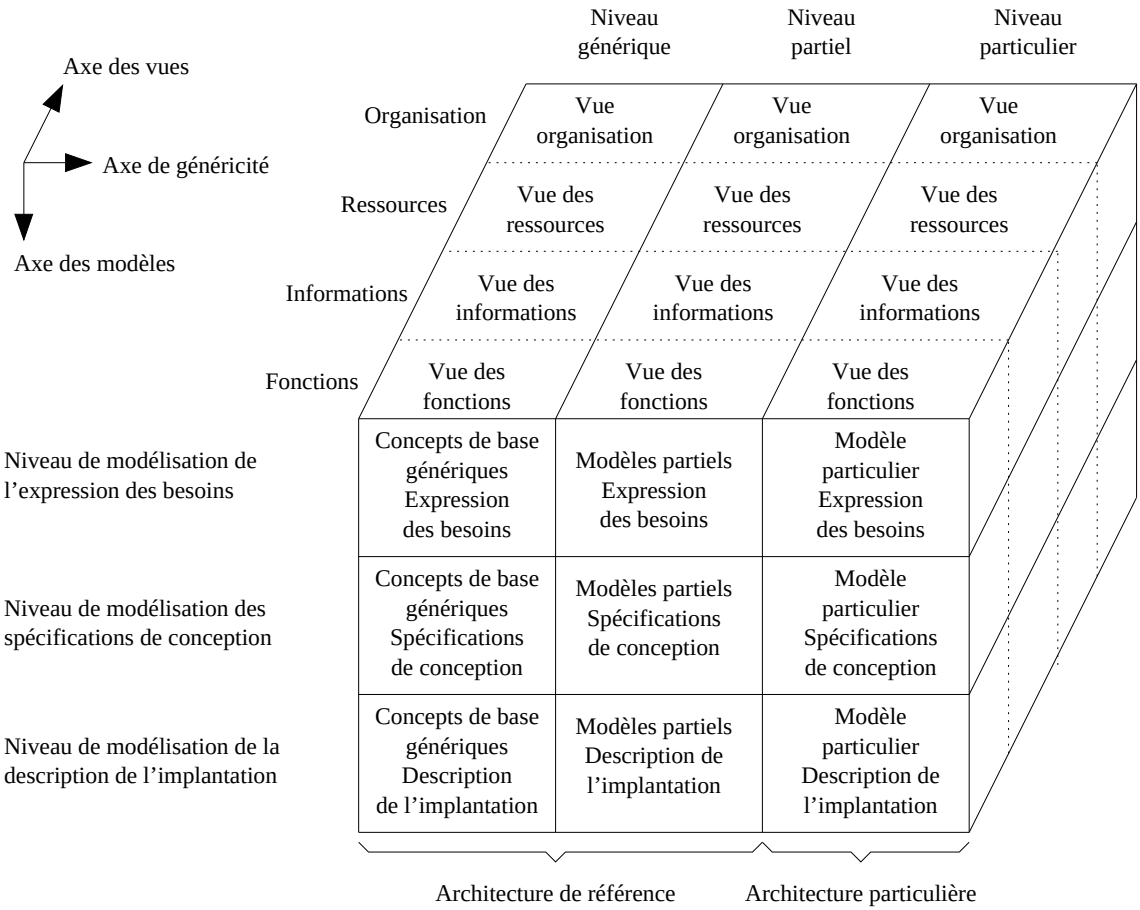


FIG. 4.4 – Le « cube » CIMOSA

Chacune des vues de CIMOSA n'est pas indépendante des autres. Elles sont des filtres de lecture pour les informations contenues dans le modèle.

Les modèles CIMOSA prennent en compte le temps par le biais des dates d'occurences des événements et des durées d'exécution des activités. L'indéterminisme est aussi introduit par la gestion des événements et la gestion des exceptions. Ils permettent de représenter à la fois des aspects statiques (via les « constructs ») et des aspects dynamiques (via les « workflows » temporisés) qui permettent de mettre en œuvre la simulation et la représentation des processus concourants et coopératifs. Selon [EL MHAMEDI et al., 1997], « CIMOSA est à ce jour la seule méthode qui modélise le flux de matière, d'informations et de contrôle dans un même formalisme unifié ».

Contrairement aux autres méthodes de modélisation d'entreprise, CIMOSA ne préconise pas de formalisme graphique. En effet, [VERNADAT, 1998] considère que « la modélisation graphique des processus (est) par nature ambiguë ». Malheureusement, ce manque de représentation graphique n'a pas permis à CIMOSA de pénétrer dans le milieu industriel de la même manière qu'IDEF0 ou Merise.

4.5.2 Démarche de modélisation

La démarche de modélisation passe par les étapes suivantes [Vernadat, 1998] :
- une analyse des domaines fonctionnels de l'entreprise, ainsi que de leurs relations ;
- l'identification des processus maîtres à modéliser ;
- l'analyse détaillée des processus maîtres selon les principaux « constructs » définis dans la méthode ;
- la consolidation du modèle au niveau spécification de conception ;
- l'adaptation du modèle aux contraintes d'implantation ;
- la traduction du modèle dans le langage des systèmes utilisés.

Cette démarche s'inscrit dans une méthodologie plus complète, qui prend en compte l'ensemble du cycle de vie du système. La méthodologie retenue est PERA, que nous allons présenter dans la section suivante.

4.6 La méthode PERA

PERA (« Purdue Enterprise Reference Architecture ») est une méthodologie d'ingénierie des environnements industriels, développée par [Williams, 1994]. L'architecture est décomposée en cinq étapes [Boutin, 2001] :
- une phase de *conceptualisation* qui est composée de l'identification (définition de l'étendue de l'étude) et de la conception (description de la mission, de la vision et des valeurs de l'entreprise) ;
- une phase de *définition* durant laquelle sont définis les besoins de mise en œuvre, les tâches, les modules et macro-fonctions nécessaires pour les besoins, et enfin les diagrammes de connexion entre les tâches, les modules et les macro-fonctions ;
- une phase de *conception* qui est décomposée en une phase de conception fonctionnelle (spécification des choix initiaux de l'architecture du système d'information, de l'organisation humaine et de la partie opérative) et d'une phase de conception détaillée (description des détails correspondant aux informations renseignées dans la conception fonctionnelle) ;
- une phase d'*installation* et de *construction* qui consiste à mettre en œuvre les décisions prises durant les phases précédentes en termes d'installation et de test des bases de données et des programmes, de formation du personnel, et de l'installation des équipements ;
- une phase *opérationnelle* et de *maintenance* qui correspondent à l'installation effective du système et à son évolution.

4.7 « Unified Enterprise Modelling Language »

Actuellement, il est nécessaire aux entreprises de bien comprendre et maîtriser leur manière de fonctionner. Elles doivent régulièrement modifier leur structure organisationnelle pour faire face aux changements imposés par leur environnement afin d'améliorer leurs critères de satisfaction économiques (coûts, ...), qualitatifs (robustesse des produits, ...) ou temporels (délais de livraison, ...).
Ainsi, les entreprises doivent mettre en œuvre des modèles leur permettant de comprendre, analyser et optimiser leurs comportements. Cette construction, aussi appelée *modélisation en entreprise*, fait appel à diverses notations, formalismes, méthodes et outils pour mettre en avant les différentes facettes du système. Selon [VERNADAT, 2000], la modélisation en entreprise peut être définie comme l'art d'externaliser la connaissance en entreprise qui augmente la valeur de l'entreprise ou des besoins à pourvoir. L'objectif premier est de permettre la description de la notion en entreprise par le biais des aspects fonctionnels, comportementaux, informationnels, de gestion des ressources, organisationnels ou économiques. Elle doit tenir compte des structures en entreprises seules ou de réseaux en entreprises. Ainsi, l'objectif est de réaliser un modèle qui permette de comprendre, structurer, évaluer, optimiser et contrôler l'ensemble des opérations composant l'entreprise [ROLSTADAS, 1995 ; VERNADAT, 1996]. L'un des moyens pour mettre en œuvre cet objectif est d'utiliser ces modèles au sein d'outils de simulation.
La modélisation en entreprise prend ses bases dans la modélisation fonctionnelle (SADT, ...), la modélisation de systèmes d'informations (modèles entité-relations, ...) et les diagrammes de flux de données. Ces principes ont émergé durant les années 70, principalement dans les domaines de l'analyse des systèmes d'information et du génie logiciel. Durant la décennie suivante, des méthodes spécifiques ont été proposées pour modéliser et concevoir des systèmes manufacturiers de grande taille. Ces nouvelles approches proposent un lien avec les processus de simulation (ensemble des méthodes IDEF, GRAI, ...). Ainsi, la modélisation en entreprise est passée d'un point de vue fonctionnel à un point de vue centré plus spécifiquement sur les processus opérationnels [ZELM et al., 1995]. Durant les années 90, la modélisation en entreprise a adopté successivement une approche orientée objet [MERTINS et al., 1995] et une approche ontologique [FOX et GRUNINGER, 1995]. Cette dernière permet de prendre en compte une plus grande part des concepts inhérents à la modélisation des processus opérationnels au sein d'une entreprise, et supporte plus efficacement les échanges inter-systèmes (informations, ...) en utilisant des formats comme « Process Interchange Format » [LEE et al., 1998] et « Process Specification Language » [NIST, 2001].
Après l'effort de développement du projet de pré-normalisation CIMOSA [HEULUY et VERNADAT, 1997], qui voulait répondre aux problèmes d'intégration et de

modélisation en entreprises (« Enterprise Modeling and Integration »), de nombreux langages (IDEF3, IEM, DEM, ...) et outils (ARIS toolset, FirstSTEP, PROPLAN, ...) sont apparus. Selon [Vernadat, 2001], cette situation correspond à une « tour de Babel » pour les utilisateurs de ces outils et de ces langages. En effet, lorsqu'une personne veut utiliser un outil, il doit impérativement apprendre un nouveau langage alors qu'ils utilisent les mêmes concepts pour décrire l'entreprise (activités, ...). Cette situation est d'autant plus critique que tous ces outils ne peuvent échanger les modèles ou communiquer entre eux.
C'est en 1997 que cette problématique est reconnue par l'ensemble de la communauté, et qu'est née l'idée de concevoir une approche normative universelle [Kosanke et Nell, 1997]. Ainsi, UEML (« Unified Enterprise Modelling Language ») a pour objectif de proposer un ensemble clair de syntaxes et de sémantiques pour la modélisation en entreprise [Vernadat, 2001].

4.7.1 Principes

L'idée principale est de proposer un langage unifié pour la modélisation d'entreprises et non pas un langage capable de remplacer tous ceux préexistants.
Sur les bases d'une étude comparative réalisée par le groupe de travail CEN ENV 12204 [CEN, 1995], un ensemble de constructions a été dégagé pour couvrir les besoins en modélisation d'une entreprise. La figure 4.5 résume l'ensemble des principes déjà mis en avant par ce groupe et qui sont la base de la réflexion sur UEML.

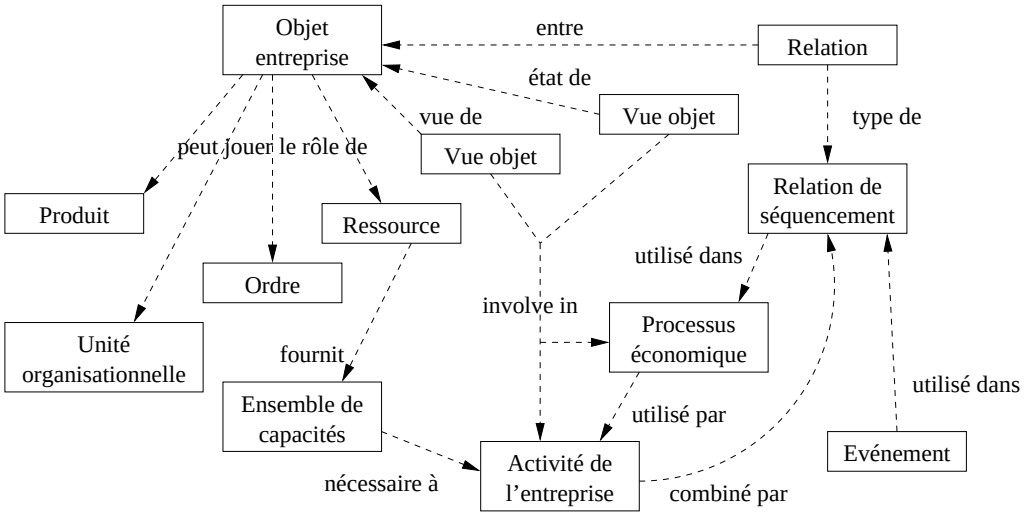


Fig. 4.5 – Constructions du CEN ENV 12204 [Vernadat, 2001]

Dans un premier temps, UEML est destiné à être un langage facile à apprendre et à utiliser, et qui serait une interface standardisée d'utilisation pour l'ensemble des outils existants. Ainsi, les utilisateurs ont besoin de connaître un seul langage pour

accéder à un grand nombre de systèmes. Dans un second temps, UEML permet l'échange entre différents systèmes de modélisation d'entreprise. Ainsi, les modèles d'entreprises pourront être réutilisés quelque soit l'outil utilisé.
UEML propose un consensus à la communauté scientifique tant au niveau de la terminologie que des structures conceptuelles utilisables pour représenter une entreprise. Il est principalement basé sur un métamodèle (et son ontologie associée) qui a été accepté par les utilisateurs finaux et par les concepteurs d'outils de modélisation. Cet effort d'unification étant en cours de réalisation, la première étape du développement d'UEML consiste en la comparaison des métamodèles dans l'ensemble des outils et des langages existants.
Basé sur les compréhension et l'utilisation de la modélisation d'entreprise, UEML doit tenir compte des principes suivants [VERNADAT, 2001] :

a) le langage doit définir un ensemble fini d'éléments de modélisation.

b) le principe de séparation des processus et des ressources : le langage doit supporter que les entités économiques puissent être vues soit comme une grande collection de processus concurrents, soit comme une grande collection de ressources communiquantes avec une séparation claire entre les ressources et les processus.

c) le principe de séparation entre le comportement et les fonctionnalités de l'entreprise : les fonctionnalités font référence à ce qui peut être fait, alors que le comportement indique comment le faire. Des constructions séparées doivent être fournies pour les fonctionnalités et le comportement. Ainsi, la gestion des modifications du système sera beaucoup plus flexible.

d) le principe de séparation des ressources et des unités organisationnelles : les unités organisationnelles (« ceux qui décident ») doivent être distinguées des ressources (« ceux qui font »).

4.7.2 Élements de modélisation

Tout comme « PSL » [NIST, 2001], UEML propose un ensemble d'éléments principaux de modélisation et un ensemble d'éléments complémentaires. Les éléments principaux correspondent aux divers concepts nécessaires à la construction du modèle UEML. Alors que ces éléments sont communs à toute description UEML, les éléments complémentaires sont dédiés à un secteur d'activité ou à des applications particulières (services médicaux, industrie électronique, ...). Pour être compatible avec UEML, un environnement de modélisation doit impérativement supporter les éléments principaux et éventuellement certains des éléments complémentaires.
La figure 4.6 illustre, selon un formalisme UML, les principaux éléments proposés par UEML :

– **Événement :** (ou condition d'activation de processus)

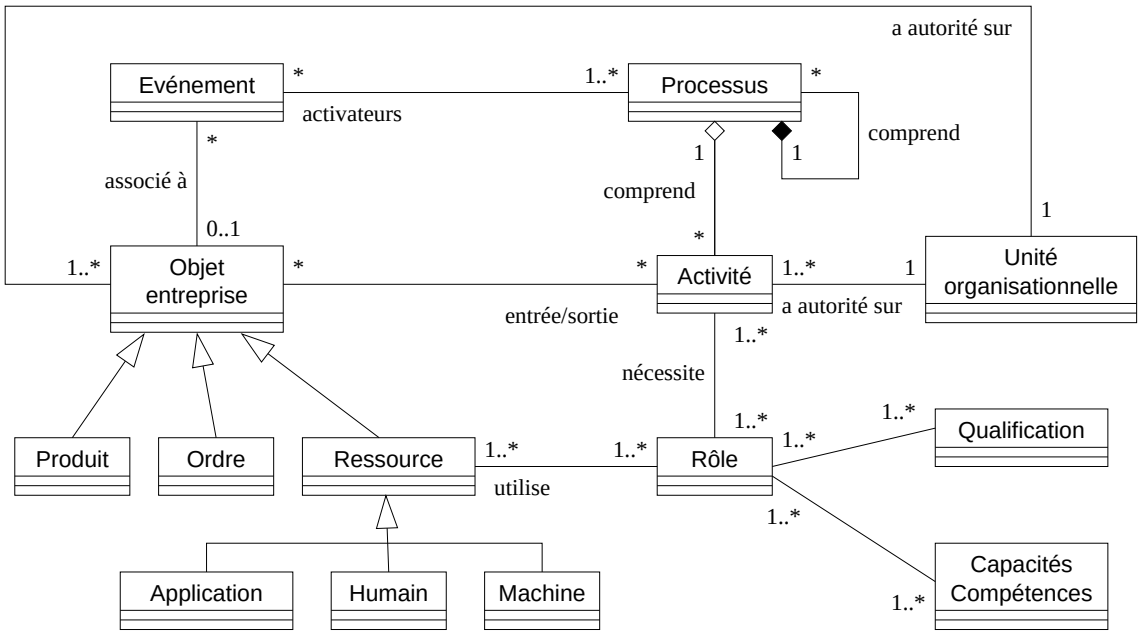


Fig. 4.6 – Élements principaux de modélisation d'UEML [Vernadat, 2001]

Un événement représente un changement de l'état du système. Il correspond à un événement sollicité ou non qui déclenche l'exécution d'un processus.

– **Processus :**

Un processus est un ensemble partiellement ordonné d'activités ou de sous-processus. Chaque processus est déclenché par l'occurence d'au moins un événement.

– **Activité :**

Une activité est une action de transformation d'une entrée en une sortie. Cette transformation est réalisée durant un temps déterminé et en utilisant un ensemble de ressources.

– **Objet d'entreprise :**

Un objet d'entreprise est une entité qui est utilisée, transformée ou créée par l'activité durant les opérations réalisées par l'entreprise. Ces objets ainsi que leurs états sont utilisés pour les entrées et les sorties des activités. Ils sont définis par leurs propriétés c'est-à-dire des attributs pour les propriétés statiques et des méthodes pour les propriétés comportementales. Les objets entreprises peuvent avoir des relations entre eux : une relation de généralisation (« is-a »), une relation d'agrégation (« part-of »), une relation définie par l'utilisateur (« association »). Comme le suggère « Integrated Enterprise Modelling approach » [Mertins et al., 1995 ; Spur et al., 1996], les objets d'entreprise peuvent être spécialisés en trois sous-catégories : produits, ordres, ressources.

– **Ressource :**

Il s'agit d'une spécialisation d'objet d'entreprise. Elle est utilisée pour permettre l'exécution des activités. En plus des propriétés héritées de la classe des objets d'entreprise, la ressource définit un ensemble de rôles qui l'utilisent, un calendrier indiquant les périodes d'activité de la ressource, et les capacités de cette ressource. Un coût d'utilisation peut aussi être défini dans cette classe d'objet. Comme cela est suggéré par CIMOSA, les ressources peuvent être spécialisées en trois catégories : les applications, les ressources humaines et les machines.

– **Unité organisationnelle :**
Une unité organisationnelle définit un élément d'une structure organisationnelle. Cet élément possède une autorité et la responsabilité d'un ensemble d'activités et d'objets de l'entreprise. Elle définit un centre de prise de décision à un niveau particulier (atelier, département, ...). Ce niveau est caractérisé par un horizon de planification incluant une période de révision comme cela est suggéré par GRAI.

4.8 Conclusion

La modélisation en entreprise est un domaine scientifique récent. Elle a pour objectif de permettre de modéliser une entreprise dans sa globalité (économique, logistique, ...). Ainsi par l'intermédiaire de langages ou de méthodes comme la suite IDEF [US AIR FORCE, 1993a] ou comme CIMOSA [VERNADAT, 1998], les entreprises peuvent être modélisées afin de mieux analyser et comprendre leur fonctionnement. Dans un premier temps, nombre de méthodes se sont spécialisées dans un aspect particulier : IDEF0 pour la modélisation des activités, ou Merise pour la modélisation des systèmes d'information. Dans un second temps, de nouvelles méthodologies sont apparues. Elles permettent de modéliser l'entreprise d'un point de vue plus global. Par exemple, GRAI [DOUMEINGTS, 1984] prend en compte les aspects physiques du système de production, les aspects informationnels et les moyens de pilotage et de prise de décision au sein de l'entreprise. Toutefois, chacune de ces méthodes présente un pouvoir d'expression qui lui est propre et ne modélise pas, en général, la totalité de l'entreprise. D'autre part, chaque outil ou méthode possède ses propres concepts et ses propres définitions. Des travaux récents comme ceux qui ont donné naissance à UEML [VERNADAT, 2001], veulent proposer une ontologie commune à l'ensemble du domaine de la modélisation en entreprise.
L'ensemble de ces travaux est très intéressant. En effet, même si elle reste jeune, la discipline de la modélisation en entreprise possède déjà de nombreux outils utilisables par les industriels (suite IDEF, GRAI, CIMOSA, ...). Les concepts mis en œuvre permettent de dégager l'ensemble des artefacts de modélisation nécessaire pour

représenter un système de production. Toutefois, nous pouvons émettre deux commentaires. La modélisation en entreprise ne tient pas encore compte de systèmes de production distribués (entreprises virtuelles, ...). Même s'il est possible de modéliser ce type de système avec les méthodes existantes, aucune d'entre elles ne propose d'artefact spécialisé. Ainsi, un problème comme la confidentialité des informations au sein d'un tel système n'est pas pris en compte. D'autre part, la modélisation en entreprise n'est pas suffisante pour comprendre le fonctionnement d'une entreprise. En effet, il est nécessaire d'utiliser des outils complémentaires, tels que la simulation, pour l'évaluation des performances. Ces derniers permettront de mettre en œuvre le comportement de l'entreprise et ainsi de mettre en évidence l'évolution des critères d'évaluation de l'entreprise. D'autre part, le point de vue adopté par la modélisation en entreprise reste bien souvent trop élevé. Cette dernière remarque trouve toutefois une réponse dans l'utilisation de méthodes de modélisation pour la simulation (décrites dans le chapitre suivant).

Nous pouvons retenir un certain nombre de concepts utilisés par la modélisation en entreprise. Ces divers « points forts » nous semblent très intéressants dans la cadre de notre problématique de mise en œuvre d'un environnement pour l'évaluation des systèmes industriels distribués.

Le premier « point fort » est l'approche de modélisation basée en général sur une vision systémique de l'entreprise. Ainsi, une représentation comme celle de [Le Moigne, 1992] ou comme celle empruntée par GRAI met en avant les différents sous-systèmes d'une entreprise. Cette dernière est souvent considérée comme composée des sous-systèmes physique, informationnel et décisionnel. Cette vision systémique peut être utilisée dans le cadre de la modélisation de systèmes distribués. En effet, elle permet de modulariser les modèles de simulation et ainsi de faciliter la modélisation, la compréhension du système, et la maintenance de ces modèles.

D'autre part, en plus de l'approche systémique proposée, les méthodes de modélisation en entreprise proposent souvent des concepts et des artefacts de modélisation qui peuvent être adaptés pour créer des modèles de simulation. Ainsi, la modélisation du sous-système décisionnel à l'aide de centres de décision, comme cela est spécifié dans GRAI, peut être utilisée pour représenter les centres de pilotages du système industriel. Au niveau du sous-système informationnel, chacune des méthodes propose sa propre vision. Par contre la prise en compte du sous-système physique est souvent trop générique c'est-à-dire que les modèles représentent essentiellement ces systèmes d'un point de vue *activité*. Cette approche est selon nous inadaptée pour comprendre tous les facteurs entrant en compte dans le comportement du sous-système. Toutefois nous verrons dans le chapitre suivant que l'utilisation d'approche de simulation résoud partiellement ce problème. En effet, la simulation

reste un outil nécessaire et complémentaire à la modélisation en entreprise. Elle facilite les mises en évidence et la compréhension des mécanismes dynamiques existants dans une entreprise (pannes, goulets d'étranglement, ...).

Chapitre 5

Méthodologies de simulation

La simulation est une technique d'analyse dynamique des systèmes de plus en plus utilisée depuis quelques dizaines d'années. Dans un cadre industriel, de nombreux outils et méthodes ont vu le jour (Arena®, Simple++®, ...). Ils permettent à la fois de réaliser des modèles de concepts et de systèmes industriels, mais aussi de mettre en œuvre un processus de simulation des comportements dynamiques du système modélisé.
Dans ce chapitre, nous présentons un ensemble d'outils de simulation permettant à la fois de créer des modèles dédiés à la simulation mais aussi de mettre en œuvre le processus de simulation à proprement parler.

5.1 Vue d'ensemble

L'un des moyens que possèdent les gestionnaires des systèmes de production pour répondre aux problèmes posés par ceux-ci (*cf.* section 2.2) est l'utilisation de la simulation.
La simulation est l'un des outils les plus efficaces à la disposition des concepteurs et des gestionnaires des systèmes industriels lorsque l'on souhaite étudier finement le comportement dynamique d'un système de production. Elle consiste à construire un modèle du système réel (ou représentation de ce dernier) et à effectuer des expériences (scénario) sur ce modèle afin de comprendre son comportement et d'en améliorer les performances.
Il existe différents types de modèles. Les modèles *physiques* sont ceux dans lesquels le système réel est représenté par une réplique ou maquette, à une échelle différente et éventuellement à l'aide de matériaux différents. Les modèles *symboliques* sont une abstraction « mathématisée » de la réalité. Ils sont en général exécutés sur un calculateur, qu'il soit analogique ou digital.
Une autre distinction concerne la prise en compte des aléas dans le modèle. Dans certains cas leur influence est considérée comme négligeable. On parlera de modèles

déterministes. Le plus souvent, les aléas doivent être représentés car ils jouent un rôle significatif (pannes, ...). On parle alors de modèles *stochastiques*.
Une troisième dichotomie sépare les modèles *statiques* et les modèles *dynamiques*. Dans les premiers, le temps n'intervient pas comme par exemple dans le cas d'un modèle comptable permettant de calculer un produit en fin d'exercice à l'aide d'un tableur. Dans les seconds, il est un facteur essentiel du comportement et de l'état du système (réacteur chimique régi par des équations différentielles, ...).
Enfin dans les modèles dynamiques, on distingue les modèles *discrets*, dans lesquels l'état du système change qu'à certaines dates (exemple d'une file d'attente devant une caisse), et les modèles *continus* où ce changement est permanent comme par exemple dans le cas d'un réacteur chimique. Un modèle qui contient à la fois des composantes discrètes et continues est dit *mixte*.
Dans le contexte de nos travaux, nous considérons la définition 2.13 page 15 : « la simulation est un moyen explicatif pour définir un système, un vecteur d'analyse pour déterminer des résultats critiques, un évaluateur de conception pour analyser et évaluer des solutions proposées, ... ». Ainsi selon les différentes dichotomies présentées ci-dessus, nous nous situons dans le cadre particulier de modélisation [LAW et KELTON, 1991]. Selon ces auteurs, la simulation de systèmes industriels se situe principalement dans la cadre des modèles :

– **symboliques :** En effet, la plupart des modèles doivent être utilisables par des outils informatiques ;

– **dynamiques :** Le temps est un facteur important pour la simulation de systèmes industriels. Ainsi il est inconcevable de ne pas tenir compte des temps de traitement des diverses machines, des délais de transports, ou des durées des pannes. En effet, tous ces éléments entrent en ligne de compte pour l'évaluation des performances des systèmes ;

– **discrets :** Les systèmes de production sont des systèmes pour lesquels les changements d'état ne sont pas continus dans le temps ou assujettis à des modèles mathématiques ou physiques. Ainsi les systèmes de production sont stables jusqu'à ce que des événements provoquent des changements dans leurs états (arrivée d'un lot de matière première, stockage d'un produit fini, arrivée d'un ordre de production, ...).

Selon [KELTON et al., 1998], la simulation est l'outil le plus populaire pour la recherche opérationnelle. Basée sur la théorie des files d'attente [KNUTH, 1998 ; CHING, 2001 ; TRIVEDI, 2001], la simulation est un outil très apprécié pour la relative facilité de mise en œuvre en comparaison des outils et des méthodes d'analyses des systèmes typiquement basés sur des approches analytiques (analyse statistique, programmation linéaire, ...). D'autre part, les résultats obtenus par le biais de simulations sont souvent plus faciles à interpréter par les industriels (principaux

utilisateurs des outils de simulation [Morgan, 1989 ; Shannon et al., 1980]). Mais la simulation ne permet pas l'optimisation d'un système. C'est donc un outil à utiliser si tous les autres outils fournis par la recherche opérationnelle ne s'appliquent pas sur le système étudié. En effet, il est primordial de faire la différence entre l'évaluation et l'optimisation d'un système. Dans le premier cas, il ne s'agit que de mettre en évidence le comportement d'un système en fonction des facteurs et des paramètres données aux modèles. L'optimisation consiste à trouver les valeurs de ces paramètres afin de maximiser un ou plusieurs critères de satisfaction (coût de production minimum, ...). Ainsi, on voit très bien que la simulation est adaptée à la résolution des problèmes d'évaluation des systèmes de production.

Dans les sections suivantes, nous présentons les méthodes et les outils de modélisation et de simulation qui nous paraissent intéressants dans le cadre de notre problématique de modélisation et de simulation de systèmes industriels distribués.

Dans une première section, nous présentons la « Conical Methodology ». À nos yeux, elle est une représentation intéressante de l'évolution des méthodologies de simulation. D'autre part, les concepts qu'elle met en œuvre nous semblent très intéressants et d'actualité. En effet, certains d'entre eux sont des réponses à notre problématique de simulation (cycle de vie d'un modèle de simulation, phases de modélisation, ...).

Nous consacrons la section suivante à la méthodologie ASCI. Sa vision des systèmes de production nous semble très interessante et adaptable à la représentation que l'on peut faire des systèmes industriels distribués. D'autre part, comme la plupart des méthodologies, elle propose un cycle de vie des modèles. À l'instar de celui proposé par la « Conical Methodology », ce cycle est communément admis par la communauté scientifique.

Enfin, nous présentons « High Level Architecture » (HLA) qui est une architecture particulièrement adaptée à nos besoins de simulation. En effet, elle permet l'interopérabilité des simulations et de leurs modèles. Très proche de notre problématique de distribution des modèles au sein d'un réseau informatique, HLA propose un canevas architectural capable de faire communiquer des outils de simulation hétéroclites. Ainsi, même s'ils ne répondent pas entièrement à notre problématique, les concepts mis en œuvre dans HLA peuvent être réutilisés dans le cadre de la simulation de systèmes industriels distribués.

Nous prenons le parti de ne pas présenter les outils et les langages de simulation « classiques » tels que QNAP, SIMAN, Simple++®, les réseaux de Pétri ou Arena®. En effet, ces divers outils ne proposent pas d'approche directement liée à la résolution de notre problématique. Toutefois, nous les utiliserons comme des briques élémentaires de simulation au sein de l'architecture que nous proposons dans la partie III. De plus, nous les considérons comme des outils nécessaires pour la mise en œuvre de notre méthodologie : notre but n'est pas de proposer un nouvel outil de

simulation, mais d'utiliser les outils existants et de les adaptées aux besoins de notre démarche.

5.2 « Conical Methodology »

La « Conical Methodology » (CM), proposée par [NANCE, 1981], apparaît comme la plus ancienne des approches méthodologiques pour le développement de modèles de simulation. Elle reste toutefois d'actualité aux vues de ses récentes évolutions [NANCE, 1994a ; NANCE, 1994b]. La CM s'articule autour de quatre axes :

- La spécification d'un modèle doit être séparée de son exécution, c'est-à-dire que les détails d'exécution, souvent nécessaires aux langages de simulation, ne doivent pas influencer la conception du modèle ;
- La spécification et la documentation d'un modèle sont inséparables et doivent être indépendantes des solutions utilisées pour l'exécution : il est nécessaire de retarder le plus possible le choix de l'outil d'exécution ;
- La spécification d'un modèle doit pouvoir être réalisée avec une approche hiérarchique par raffinements successifs ;
- Un langage de spécification et de documentation du modèle est nécessaire.

La CM est une méthodologie récursive orientée objet. Le modèle est représenté par un objet qui peut être composé d'un ensemble d'objets et de relations entre ces derniers. La spécification par raffinements successifs permet de contrôler l'exactitude, la complétude, la cohérence et tous les critères nécessaires à l'implantation du modèle.

5.2.1 Cycle de vie d'un modèle

Selon [NANCE, 1994a], le cycle de vie d'un modèle est composé des étapes suivantes (figure 5.1) :

- Le *modèle conceptuel* est la représentation par l'expert du système à modéliser. Elle est fortement influencée par la perception de l'expert vis-à-vis du système, ainsi que par les objectifs à atteindre ;
- Un ou plusieurs *modèles communicatifs* sont élaborés à partir d'un modèle communicatif précédent ou d'un modèle conceptuel. Ces modèles doivent pouvoir être communiqués et compris par les experts qui pourront les utiliser pour les comparer au système et aux objectifs ;
- Un *modèle informatique*, élaboré à partir des précédents modèles, représente un modèle communicatif à partir duquel les résultats expérimentaux sont obtenus. Pour construire ce modèle, il est possible d'utiliser des langages comme SIMSCRIPT [CACI, 2001] ou SIMULA [BIRTWISTLE, 1979 ; MAGNUSSON, 1994] ;

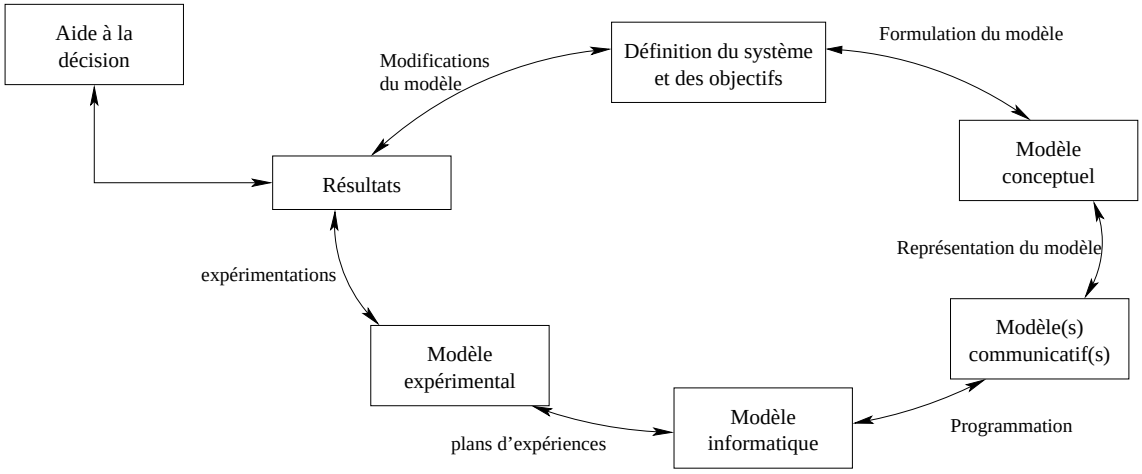


Fig. 5.1 – « Conical Methodology » : cycle de vie d'un modèle

- Un *modèle expérimental* complète le modèle informatique par une description *exécutable* des expériences à réaliser ;
- Les *résultats* sont obtenus par l'exécution du modèle expérimental selon un scénario d'expérimentations. Si plusieurs exécutions sont réalisées, il est nécessaire de mettre en œuvre des changements des données d'entrée, des changements structurels, ... ;
- La phase d'*aide à la décision* permet d'étudier les résultats des différents scénarios pour en extraire les caractéristiques comportementales du modèle. Une analyse complète permet d'extrapoler ou de prédire le résultat d'expériences futures ;
- Les *modifications* du modèle sont engendrées par des redéfinitions des objectifs de l'étude ou par une extension des fonctions du modèle. Comme l'illustre la figure 5.1, les modifications ne sont pas représentées comme une phase mais plutôt comme un concept global permettant de revenir en arrière.

5.2.2 Modélisation avec CM

La CM a pour but d'offrir un fil conducteur pour mettre en œuvre la modélisation et pour permettre de contrôler la complexité des modèles de manière à pouvoir les vérifier et les valider. Elle prend en compte les cinq premières phases du cycle de vie d'un modèle. L'approche proposée par la CM pour spécifier un modèle est décrite dans la table 5.1.

Cette approche montre que le modèle est défini au travers des attributs, des objets, et des relations entre eux. Les étapes 3a et 3b illustrent la décomposition hiérarchique d'un modèle en sous-modèles et de ces derniers en composants de niveau inférieur. L'ordre de définition de ces modèles reste à la charge de l'expert. Ce dernier peut utiliser une approche horizontale : définition de tous les sous-modèles de niveau 1,

1. Énoncé des objectifs de l'étude
   (a) Définitions
   (b) Hypothèses relatives aux objectifs
2. Environnement de modélisation
   (a) Ressources de modélisation
      i. Organisation, dates, ressources humaines ... affectées à l'étude
      ii. Planning prévu et budget alloué
   (b) Hypothèses du modèle
      i. Conditions d'application (limites)
      ii. Interactions avec l'environnement
         A. Description des entrées
         B. Hypothèses sur les rétro-actions entre le modèle et l'environnement
         C. Description des sorties et de leur format
      iii. Définition de l'état initial
      iv. Conditions d'arrêt de la simulation
3. Définition et Spécification du modèle
   (a) Modèle
      i. Ensemble des objets
      ii. Attributs indicatifs
      iii. Attributs relationnels
   (b) Sous-modèles
      i. Sous-modèle au 1^{er} niveau d'abstraction
         A. Ensemble des objets
         B. Attributs indicatifs
         C. Attributs relationnels
         D. Sous-modèle au $2^{ème}$ niveau d'abstraction
         (...(D)...) Sous-modèle au $n^{ème}$ niveau d'abstraction
      ii. Sous-modèle au 1^{er} niveau d'abstraction
      ...
4. Validation du modèle et procédures de vérification
   (a) Tests de validation
   (b) Critères et tests de vérification
5. Expérimentation du modèle
   (a) Hypothèses à tester
   (b) Plan d'expériences
6. Besoins pour l'implantation

TAB. 5.1 – « Conical Methodology » : approche de modélisation

puis ceux de niveau 2, ... Il peut aussi décider de définir les sous-modèles selon une approche verticale : définition d'un sous-modèle de niveau 1, puis sa décomposition, ...

La « Conical Methodology » nous semble très intéressante pour plusieurs raisons. Tout d'abord, elle propose un cycle de vie des modèles qui est reconnu par une grande partie des communautés scientifique et industrielle. Nous pensons utiliser une version modifiée par nos soins de ce cycle de vie. Ces modifications permettront d'intégrer les spécificités dues à la modélisation et la simulation de systèmes industriels distribués. En plus de son cycle, la CM propose des concepts de modélisation très intéressants. Notons, par exemple, la séparation de la spécification d'un modèle de son exécution, ou encore le choix au plus tard des outils d'implantation de la simulation. Ce dernier point illustre de plus l'indépendance de la CM vis-à-vis des outils de simulation. Nous rappelons que nous ne désirons pas concevoir un nouvel outil de simulation, mais proposer une adaptation des outils existants afin de supporter notre problématique. Cette indépendance vis-à-vis des outils nous semble donc primordiale.

5.3 « Analyse Spécification Conception Implantation »

La méthodologie ASCI (« Analyse Spécification Conception Implantation ») proposée par [Kellert et Force, 1998a ; Kellert et Ruch, 1998b] est utilisée pour modéliser les systèmes de production à flux discrets et à partage de ressources [Laizé, 1998]. Elle met en œuvre le processus de modélisation proposé par [Gourgand, 1984] qui préconise la construction d'un modèle de connaissances et d'un modèle d'action. ASCI propose une approche systémique de décomposition d'un système de production [Kellert et Force, 1998a ; Kellert et Ruch, 1998b] ainsi que l'utilisation du paradigme objet. D'autre part, ASCI considère que les utilisateurs n'ont pas besoin de connaître les concepts utilisés par ASCI pour pouvoir l'utiliser efficacement. Un ensemble de documents (guide d'analyse des systèmes de production, glossaire et méthode de spécification des flux physiques et informationnels) est donc proposé.

5.3.1 Concepts fondamentaux

La méthodologie ASCI propose une vision systémique afin de mieux maîtriser la complexité d'un système industriel. Comme l'illustre la figure 5.2, le système de production S, plongé dans un environnement E, est composé de [Laizé, 1998] :
- l'ensemble F des fournisseurs et sous-traitants,
- l'ensemble C des commandes ou des besoins des clients,
- l'ensemble $\mathbb{C}$ des contraintes auxquelles S est soumis,
- l'ensemble P des produits que fabrique S,
- l'ensemble R des relations, connues ou que l'on peut connaître, qui expriment les interactions possibles entre S et E et entre les éléments de E.

L'ensemble C, représenté par R, correspond à la charge principale de S ; ce dernier agit sur son environnement E en fabriquant des produits P_c pour C et en commandant des matières premières à ses fournisseurs ou sous-traitants (ensemble F). Nous allons à présent nous attacher à la description des trois sous-systèmes de S.

5.3.1.1 Le sous-système informationnel

Le sous-système informationnel (aussi appelé logique dans cette méthodologie) est composé des éléments sur lesquels le système doit réaliser des opérations comme transformer, traiter ou transporter. Il prend en compte les notions de :

- **lot :** il permet de décrire de manière hiérarchique la charge du système de production ;
- **gamme :** elle représente l'ensemble des actions permettant d'atteindre un objectif ;

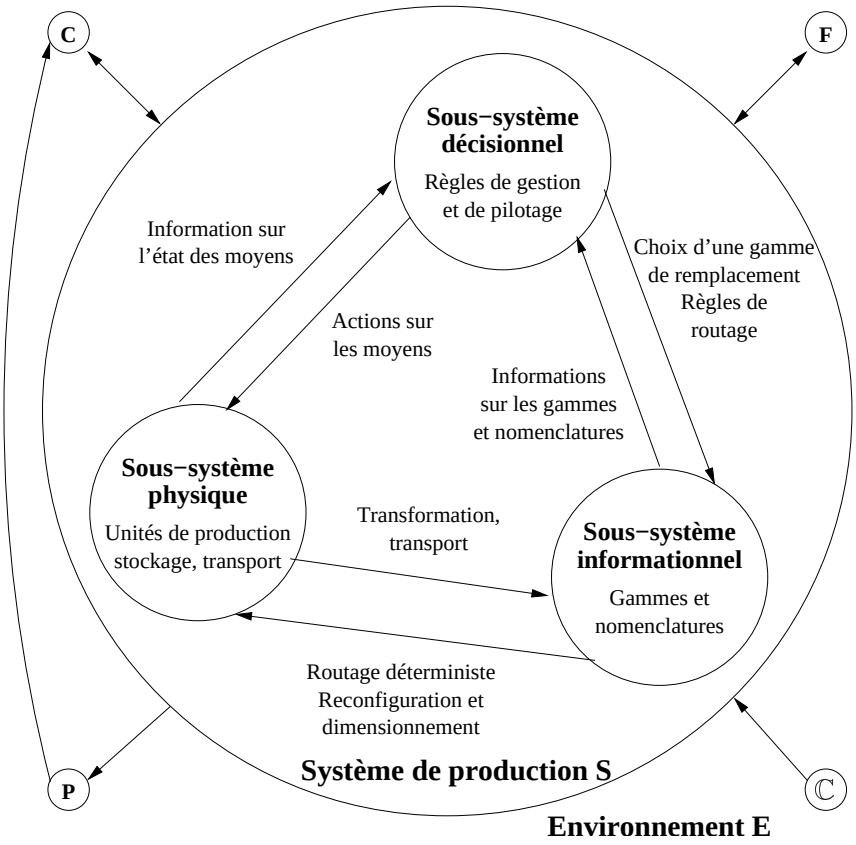


FIG. 5.2 – Système de production S plongé dans son environnement E

– **nomenclature :** elle correspond à l'ensemble des composants et des produits intermédiaires entrant dans la fabrication d'un produit.

5.3.1.2 Le sous-système physique

Le sous-système physique est défini par l'ensemble des moyens de production, de stockage et de transport, leur répartition géographique et leurs interconnexions. Il est décomposé en couches hiérarchiques. Par exemple, un système de production est composé d'ateliers comprenant plusieurs lignes de production. Pour identifier les différentes parties des systèmes, des termes génériques sont proposés : unité de production, de stockage ou de transport, station de travail, transporteur statique ou dynamique, chemin, ressource, capteurs, ... « Unité de production », « unité de stockage » et « unité de transport » sont des entités décomposables. En accord avec les experts industriels, les « ressources » représentent aussi bien les opérateurs que les outils ou les supports de stockage ou de transport. Dans la méthodologie ASCI une attention particulière a été accordée aux chemins utilisés par les transporteurs. Une décomposition en tronçons et carrefours a été proposée :

– un tronçon est une partie du chemin durant laquelle se déroule un transport pur.

Un seul type de transport (dynamique ou statique) peut emprunter un tronçon mais ce dernier peut appartenir à plusieurs chemins ;
- un carrefour est une zone dans laquelle se croisent plusieurs chemins. Il est considéré comme une section critique (risques de collision, ...) ;
- la notion de capteur permet d'introduire la récolte d'informations destinées au sous-système décisionnel qui peut alors agir sur le mécanisme de transport à l'aide d'actionneurs.

5.3.1.3 Le sous-système décisionnel

Le sous-système décisionnel ou sous-système de gestion et de pilotage est composé d'un système d'information et d'un système de contrôle. Il contient l'ensemble des règles de fonctionnement du système ainsi que les contraintes qui doivent être satisfaites pour être cohérent avec l'environnement. La modélisation de ces règles est réalisée à travers un ensemble de centres de décision qui connaissent tout ou partie du système de production. Ils émettent des flux informationnels (ordres d'approvisionnement, de fabrication ou de livraison, ...) qui vont influencer les flux des sous-systèmes physique et informationnel.
La vision d'un système de production en une décomposition en trois sous-systèmes complémentaires et communicants permet alors la modification d'une partie de l'un tout en gardant une cohérence avec les autres. Une justification de cette décomposition se trouve dans [Kellert et Force, 1998a].

5.3.2 Modélisation

Le paradigme objet est une extension naturelle de l'approche systémique de par ses concepts d'encapsulation, d'héritage, et d'agrégation ou de composition [Booch, 1994 ; Rumbaugh et al., 1991]. Ils autorisent à un nombre potentiellement élevé d'individus d'appréhender et de mettre en œuvre aisément, et de manière fiable, une approche systémique. Utilisée tout au long de la modélisation, l'approche orienté-objet assure une démarche totalement cohérente. [Kellert et Force, 1998a] précise l'utilisation d'une approche hiérarchique qui permet de mener l'étude avec, à la fois, une analyse descendante (passage d'une vision globale à une description détaillée du système) et une analyse ascendante (localisation d'un élément du système dans le contexte industriel).
Les principales phases composant la méthodologie ASCI sont :

- **Analyse du domaine :** Son objectif est de formaliser un modèle de connaissances du système de production qui soit à la fois générique et orienté-objet. Le formalisme Entité-Association-Attribut [Chen, 1976 ; Tardieu et al., 1985] est préconisé par [Kellert, 1992].

– **Spécification :** Cette seconde phase consiste à passer d'une vision Entité-Association-Attribut (EAA) à une vision objet du domaine.

– **Conception et Implantation :** Ces deux étapes ont pour objectif la réalisation de l'implantation informatique de l'environnement de modélisation.

Nous nous intéressons à ASCI pour plusieurs raisons. Tout d'abord, il s'agit d'une approche récente qui intègre des travaux récents de modélisation en entreprise. Toutefois, cette approche méthodologique ne supporte pas naturellement les nouvelles organisations d'entreprises (réseaux, entreprise virtuelle, ...). Un autre attrait d'ASCI est son approche méthodologique basée sur un cycle de vie des modèles qui est très proche de celui proposé par la « Conical Methodology ». D'autre part, les concepts manipulables durant la conception sont intéressants et illustrent les artefacts nécessaires à la modélisation de systèmes industriels (lot, gammes, systèmes de transport, ressources humaines, ...). Enfin, ce qui nous semble le plus intéressant dans l'approche proposée par ASCI est sa vision systémique de la décomposition d'un système industriel. Rappelons que cette décomposition nous semble très intéressante pour améliorer la conception, la compréhension et la modularité des modèles de simulation. Pour conclure cette section, nous pensons que la méthode ASCI possède toutes les bases nécessaires à la résolution d'une partie de notre problématique. Ce qu'il lui manque tient essentiellement à son manque de support de la notion de distribution du point de vue conceptuel et de l'implantation.

5.4 « High Level Architecture »

La simulation est un domaine pour lequel il existe un très grand nombre de classes (systèmes industriels, phénomènes physiques, ...). Face à cette problématique, il est très difficile de faire communiquer ces différentes approches de la simulation. L'interopérabilité entre les modèles de simulation est une caractéristique rarement supportée. Pour répondre à ce problème, l'université de Californie a proposé, sous la tutelle du Département de la Défense (DoD), une architecture supportant l'interopérabilité des modèles de simulation [US DEPARTMENT OF DEFENSE, 1996]. HLA (« High Level Architecture ») est un ensemble de règles et de canevas dont l'objectif est de proposer une architecture commune à tous les outils de simulation. Ainsi, grâce à HLA, des informations et des modèles peuvent être échangés entre diverses applications. D'autre part, à l'instar de CORBA [MOWBRAY et ZAHAVI, 1995 ; SIEGEL, 1996], HLA propose des mécanismes de communication au sein d'un réseau informatique.
La terminologie de HLA est basée sur deux concepts :

1. un « fédéré » est un membre d'une fédération HLA. Il s'agit d'une application qui participe à la vie d'une fédération. Dans la réalité, il peut prendre la forme

d'un gestionnaire de fédérés, d'un collecteur d'information, de simulations en cours de réalisation ou encore de visualisateurs passifs.

2. une « fédération » est un ensemble d'objets fédérés.

La définition formelle de HLA par le groupe « Modeling and Simulation » (M&S) comprend trois composants principaux : les règles HLA, la spécification de l'interface des fédérés HLA, et un canevas de modèle orienté-objet HLA (« Object Model Template » ou OMT).
HLA est l'une des infrastructures proposées par le M&S qui, lorsqu'elle est utilisée dans un cadre technique défini, fournit des mécanismes puissants pour l'interopérabilité et la réutilisabilité des modèles de simulation. Mais, si HLA est nécessaire pour supporter l'interopérabilité, il n'est pas suffisant en soi. En effet, HLA propose une architecture de communication entre les modèles de simulation. Mais, la modélisation et la compréhension des informations transitant par les mécanismes d'HLA sont du ressort des modèles de simulation et non pas d'HLA. Pour avoir une interopérabilité totale, il est donc nécessaire d'utiliser HLA comme vecteur de communication et d'implanter dans les objets fédérés des mécanismes permettant d'utiliser les données échangées.
Dans les sections suivantes, nous présentons les trois composants de HLA proposés dans les spécification du M&S.

5.4.1 Règles HLA

HLA propose un certain nombre de règles pour la construction d'une fédération :

- **Règle 1 :** Les fédérations doivent avoir un modèle de fédération (« Federation Object Model » ou FOM) ayant une documentation respectant l'OMT.
- **Règle 2 :** Dans une fédération, toutes les instances des objets associés à une simulation doivent appartenir à un objet fédéré, et pas à l'infrastructure d'exécution (« RunTime Infrastructure » ou RTI).
- **Règle 3 :** Pendant l'exécution d'une fédération, tous les échanges de données entre les objets fédérés doivent passer par la RTI.
- **Règle 4 :** Pendant l'exécution de la fédération, tous les objets fédérés doivent interagir avec l'infrastructure d'exécution selon les spécification d'interface de HLA.
- **Règle 5 :** Pendant l'exécution de la fédération, un attribut instancié doit appartenir à un seul objet fédéré à un instant donné.

HLA propose aussi un certain nombre de règles pour la construction des objets fédérés :

- **Règle 6 :** Les objets fédérés doivent contenir un modèle de simulation (« Simulation Object Model » ou SOM) documenté selon les spécifications de l'OMT.
- **Règle 7 :** Les objets fédérés doivent pouvoir modifier ou propager tout attribut, et envoyer ou recevoir toute interaction, comme cela est décrit dans leurs spécifications SOM.
- **Règle 8 :** Les objets fédérés doivent pouvoir transférer ou accepter la propriété d'attribut durant l'exécution de la fédération, comme cela est spécifié dans les structures SOM.
- **Règle 9 :** Les objets fédérés doivent promouvoir la variation des conditions qui leur permettent de fournir les services de modification des attributs.
- **Règle 10 :** Les objets fédérés doivent autoriser la gestion locale du temps de manière à permettre la coordination des échanges de données avec les autres membres de la fédération.

5.4.2 « Object Model Template »

L'« Object Model Template » (OMT) décrit le format et la syntaxe devant être utilisés pour représenter l'information dans les modèles HLA. Il comprend les inclusions d'objets, les attributs, les interactions et les paramètres. Mais, il ne décrit en aucune manière les informations spécifiques aux domaines des objets de simulation (véhicules dans le cadre d'une simulation de flux routier, produits pour la simulation de systèmes industriels, ...).
L'OMT est un canevas structurel standardisé pour spécifier les modèles orienté-objet de HLA.
L'OMT peut être utilisé pour décrire un membre d'une fédération (ou objet fédéré), créer un modèle de simulation (SOM), ou décrire un ensemble nommé d'objets fédérés interagissants (ou fédération), créer un modèle de fédération (FOM). En d'autre termes, l'objectif principal de l'OMT est de faciliter la mise en œuvre de l'interopérabilité entre les simulations, et la réutilisation des composants de simulation.

5.4.2.1 « Federation Object Model »

Durant le développement d'une fédération HLA, il est primordial que tous les membres d'une fédération puissent comprendre la nature et les caractéristiques des communications nécessaires entre les objets fédérés. L'objectif premier d'un modèle de fédération (« Federation Object Model » ou FOM) est de fournir une spécification des échanges de données entre les objets fédérés dans un format commun et standard. Le contenu de ces données inclue une énumération de tous les objets et classes

d'interaction pertinents dans la fédération, ainsi que les attributs ou les paramètres qui caractérisent ces classes. En d'autres termes, les éléments composant un FOM correspondent à un « contrat de modélisation informationnelle » qui est nécessaire, mais pas suffisant, pour répondre à l'interopérabilité entre les objets fédérés.

5.4.2.2 « Simulation Object Model »

Une étape critique dans la formation d'une fédération est le processus qui permet de déterminer la composition des systèmes individuels de simulation afin de respecter les besoins et les objectifs des commanditaires. Un modèle de simulation (« Simulation Object Model » ou SOM) est une spécification des capacités intrinsèques qu'une simulation individuelle peut proposer à l'ensemble d'une fédération. Le format standard dans lequel sont exprimés les SOM facilite la détermination des possibilités d'utilisation d'un système de simulation dans une fédération.

5.5 Conclusion

Face aux besoins sans cesse croissants des industriels (apparition de nouveaux types d'organisation, ...), la communauté scientifique a proposé de nombreux concepts et outils. L'un d'eux est la modélisation en entreprise présentée dans le chapitre 4. Mais cette dernière ne répond pas entièrement aux besoins des industriels. Il est nécessaire d'utiliser une technologie plus ancienne : la simulation informatique. Cette dernière a toujours accompagné le développement de l'informatique. Utilisée par les industriels depuis les années 70 [Rasmussen et George, 1978 ; Thomas et DaCosta, 1979], la simulation est considérée comme l'un des meilleurs outils de modélisation et d'analyse des systèmes industriels au même titre que les méthodes analytiques comme l'analyse statistique ou la programmation linéaire. Ainsi, elle réunit un certain nombre d'avantages au nombre desquels nous pouvons mentionner la relative facilité de mise en œuvre en comparaison des autres méthodes d'analyse, ou d'interprétation des résultats obtenus. Un autre avantage est sa capacité à tenir compte des phénomènes dynamiques des systèmes industriels (pannes, génération d'entités, ...). De plus les mécanismes stochastiques, mis en œuvre par les outils de simulation, permettent une modélisation efficace des comportements des systèmes industriels.

Mais la simulation n'est pas une réponse universelle pour les besoins des industriels. En effet, les méthodologies pour la simulation ne prennent que rarement en compte l'ensemble des facteurs intervenant dans le comportement de tels systèmes. Ainsi, la modélisation en entreprise permet de créer des modèles tenant compte des aspects économiques, productiques, ... Ces modèles peuvent alors être utilisés pour la simulation.

D'autre part, les outils de simulation utilisés par les industriels offrent en général des visions différentes quant à la modélisation et la simulation des systèmes industriels.

Ces problèmes sont partiellement résolus par l'apparition de méthodes « généralistes » telles que la « Conical Methodology » ou ASCI. Toutefois, elles ne prennent pas en compte les évolutions récentes des organisations des entreprises. Ainsi, les systèmes industriels distribués ne sont pas directement modélisables via ces méthodes.

La distribution informatique des modèles de simulation est supportée par les technologies issues de la simulation distribuée [LIN et FISHWICK, 1996 ; DAMANI et GARG, 1998]. Ces dernières permettent de synchroniser et de faire communiquer des modèles se trouvant dans des lieux différents [FUJII et al., 1999]. Malheureusement, nombre des outils proposés supportent mal l'interopérabilité des modèles. La simulation distribuée supporte en général des modèles de simulation construits selon les mêmes canevas (protocole de communication, synchronisation, domaine de simulation, ...). Toutefois, des travaux comme HLA permettent de mettre en avant l'interopérabilité des modèles, c'est-à-dire que ces derniers peuvent communiquer sans se soucier du type réel des simulations distantes (discret, continu ou mixte) ou de leur domaine (vie artificielle, système industriel, ...).

Ce manque de support, de la part des outils de simulation vis-à-vis des systèmes industriels distribués, a fait l'objet de travaux de la part de la communauté scientifique. [BURLAT et al., 1998] et [KABACHI, 1999] proposent des architectures multi-agents pour la simulation du sous-système décisionnel des systèmes de production. La simulation distribuée à base de composants est une autre possibilité pour permettre la simulation de systèmes distribués [CHEN et SZYMANSKI, 2001]. [BERCHET, 2000] propose une architecture permettant d'aider au pilotage d'un système industriel. ARÉVI est un outil de simulation qui permet de simuler le comportement de systèmes industriels selon le point de vue de la réalité virtuelle [DUVAL et al., 1997]. Il est basé sur une architecture à agents qui permet de supporter la distribution lors de la modélisation, mais aussi lors du processus de simulation.

Ainsi, la communauté est très active dans le domaine de la simulation distribuée. Nous avons constaté que les concepts d'agent et de système multi-agents sont de plus en plus utilisés pour mettre en œuvre la simulation.

Toutefois, les architectures proposées ne supportent pas l'ensemble des contraintes inhérentes aux entreprises virtuelles. Prenons l'exemple de la confidentialité de modèles de simulation. En effet, il est impossible de centraliser la conception d'un modèle de simulation d'une telle entreprise à la vue du besoin de certains de ses membres de garder une partie de leurs modèles confidentiels. D'autre part, ce type de système distribué est souvent instable dans le temps. Ainsi, les architectures existantes ne permettent pas encore de prendre en compte la constitution dynamique

des composants d'un système distribué.
Face à l'ensemble de ces problèmes et des solutions qui y ont été déjà apportées, nous pensons qu'il manque une architecture spécialisée pour la simulation de systèmes industriels distribués. Il faut que cette architecture supporte à la fois les concepts hérités d'une méthodologie de modélisation (entreprise virtuelle, ateliers, machines, ...), mais aussi la distribution informatique des modèles tout en tenant compte des spécificités des systèmes industriels distribués. Nous pensons que les systèmes multi-agents sont une réponse adaptée à la simulation de tels systèmes. Nous présentons dans le chapitre suivant les concepts attachés aux systèmes multi-agents, ainsi que les raisons de notre choix en faveur de cette technologie.

Chapitre 6

Systèmes multi-agents

Dans ce chapitre, nous présentons les bases des systèmes multi-agents, ainsi que quelques plateformes et environnements de modélisation. Enfin, nous concluons par les raisons qui nous ont fait choisir les systèmes multi-agents pour répondre à notre problématique de modélisation et de simulation de systèmes industriels distribués.

6.1 Genèse

Dans cette section, nous présentons les problèmes et les considérations qui ont amené la communauté à développer les concepts des agents et des systèmes multi-agents.

6.1.1 Des systèmes experts...

Longtemps, l'informatique en général et l'intelligence artificielle (IA) en particulier ont considéré les programmes comme des entités individualisées et capables de rivaliser avec l'être humain dans des domaines précis. Il est possible de s'en rendre compte dès le début de l'IA. En effet, depuis le début des développements des programmes expérimentaux de Newell, Shaw et Simon [Newell et al., 1963] sur la démonstration automatique de théorèmes, le terme « intelligence artificielle » a été utilisé pour désigner un projet de recherche consistant à concevoir une machine intelligente capable de réussir aussi bien qu'un être humain des tâches jugées complexes.
Cette façon de concevoir les programmes informatiques comme des sortes de « penseurs » repliés sur eux-mêmes se retrouve directement dans la notion de *système expert*. Ces systèmes sont des programmes informatiques capables de remplacer un ou plusieurs êtres humains dans leurs tâches réputées les plus complexes et qui réclament de l'expérience, du savoir-faire et une certaine forme de raisonnement.
Cependant cette conception centralisatrice et séquentielle se heurte à plusieurs obstacles d'ordre théorique et pratique. Sur le plan théorique, l'intelligence artificielle

s'était engagée sur une voie à la fois trop optimiste et en même temps trop réductrice. L'intelligence, comme la science [LATOUR, 1989 ; LESTEL, 1986], n'est pas une caractéristique individuelle que l'on pourrait séparer du contexte social dans lequel elle s'exprime. En d'autres termes, les autres sont indispensables à notre développement cognitif et ce que nous appelons « intelligence » est autant dû aux bases génétiques qui définissent notre structure neuronale générale qu'aux interactions que nous pouvons avoir avec le monde qui nous entoure, et en particulier, avec la société humaine. Les systèmes experts manquent de capacités d'interaction avec leur environnement et leurs semblables [DREYFUS, 1979 ; SEARLE, 1991].

D'autre part, les systèmes informatiques deviennent de plus en plus complexes. Il est nécessaire de les décomposer en modules « faiblement couplés », et en unités indépendantes dont les interactions peuvent être limitées et parfaitement contrôlées. Pour des raisons différentes, les concepteurs de systèmes experts industriels sont arrivés à des conclusions semblables à partir des difficultés soulevées par le développement de bases de connaissances. En effet, le savoir-faire, les compétences et les connaissances sont détenus par des individus différents qui, au sein d'un groupe, communiquent, échangent leurs connaissances et collaborent à la réalisation d'une tâche commune. Dans ce cas, la connaissance du groupe n'est pas égale à la somme des connaissances des individus : chaque savoir-faire est lié à un point de vue particulier et ces différentes visions, parfois contradictoires, ne sont pas nécessairement cohérentes entre elles. La réalisation d'une tâche commune nécessitera alors des discussions, des mises au point, voire même des négociations pour résoudre les conflits éventuels.

Les systèmes experts ont rendu de nombreux services dans le cadre de la résolution de problèmes complexes (joueurs d'échecs, optimisation de procédés industriels complexes, ...). Toutefois, la complexité de tels systèmes pose un réel problème quant à leur conception et à la maîtrise de leur fonctionnement. De plus, la distribution des connaissances nécessaires à la résolution de certains problèmes, ou tout simplement la prédisposition naturelle d'un problème à être distribué, ont mis en avant certaines limites des systèmes experts.

6.1.2 ...aux systèmes distribués

À partir de toutes ces remarques, la communauté scientifique s'est penchée sur la possibilité de diminuer la complexité de la résolution des problèmes en distribuant celle-ci. Ainsi, l'Intelligence Artificielle Distribuée (IAD) s'est intéressée aux problématiques liées à la distribution de systèmes experts [GASSER, 1992]. Par conséquent, elle s'est penchée sur la conception de systèmes experts plus petits qui communiquent et interagissent pour résoudre un problème plus important. Toutefois, l'autonomie de ces différents systèmes reste limitée. En effet, ils restent par

nature des systèmes experts. D'autre part, l'adaptation de ces systèmes experts à des problèmes différents reste un problème toujours aussi complexe à résoudre. Ainsi, l'IAD a donné naissance à des architectures permettant de résoudre plus efficacement les problèmes d'intelligence artificielle. Ils n'ont toutefois pas proposé de solution à tous les problèmes hérités de l'IA. D'une part, la définition de l'intelligence reste quelque chose de non consensuel (elle reste difficile à définir, comprendre et reproduire). D'autre part, certains problèmes d'intelligence artificielle ne peuvent être résolus efficacement par un système expert. Prenons l'exemple de la simulation du comportement d'une population dans un lieu clos. La mise en œuvre d'un système expert est très délicate car, il faut d'une part modéliser le comportement des individus, et d'autre part, simuler les interactions et les influences entre ces différents individus. Nous voyons clairement que l'IAD permet de résoudre le problème du comportement des individus, mais qu'en est-il des interactions entre ceux-ci ?
À la lumière de ces nouveaux problèmes, la communauté scientifique s'est intéressée à la modélisation des activités simples ou complexes sous la forme d'entités relativement autonomes et indépendantes qui interagissent entre elles. Ces entités, appellées *agents*, travaillent au sein de communautés selon des modes parfois complexes de coopération, de conflit et de concurrence, et qui peuvent parfois survivre et se perpétuer. Ainsi, de ces interactions peuvent émerger des structures organisées qui en retour contraignent et modifient les comportements des agents la composant. Nous définissons les agents et les systèmes multi-agents dans la section suivante.

6.2 Définitions

Les agents et les systèmes multi-agents sont la résultante d'une évolution naturelle de l'intelligence artificielle, qu'elle soit distribuée (IAD) ou non (IA). Dans cette section, nous définissons ces deux concepts.

6.2.1 Agent

Les agents peuvent être soit des *entités physiques* qui agissent sur le monde réel, soit des programmes informatiques que l'on désigne par *entités virtuelles*. Nous nous intéresserons exclusivement à cette deuxième catégorie. Ils sont *immergés* dans un environnement sur lequel ils sont *capables d'agir* et non pas seulement de *raisonner* comme dans les systèmes d'IA classique. Ainsi, les actions réalisées par les agents modifient leur environnement. Ces changements influent sur leurs processus futurs de prise de décision. Les agents peuvent aussi directement *communiquer* entre eux, ou via l'environnement. Ils sont doués d'*autonomie*, c'est-à-dire qu'ils ne sont pas dirigés par des commandes venant de l'extérieur mais par un ensemble de tendances qui peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de

satisfaction ou de survie que l'agent cherche à optimiser. Une autre caractéristique des agents est qu'ils n'ont qu'une vision locale du système, c'est-à-dire qu'ils peuvent avoir un modèle partiel de leur environnement. L'ensemble de ce qui est perçu par l'agent appartient à son *environnement*.
De la définition 6.1, nous retenons essentiellement la vision proposée par Nick JENNINGS qui nous semble plus appropriée et plus d'actualité.

— DÉFINITION 6.1 : AGENT —

« On appelle agent une entité physique ou virtuelle

a) qui est capable d'agir dans un environnement,

b) qui peut communiquer directement avec d'autres agents,

c) 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),

d) qui possède des ressources propres,

e) qui est capable de percevoir (mais de manière limitée) son environnement,

f) qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),

g) qui possède des compétences et offre des services,

h) qui peut éventuellement se reproduire,

i) 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.

» [FERBER, 1995]
« 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. » [JENNINGS et al., 1998a]

6.2.2 Système multi-agents

Certains systèmes dans lesquels évoluent les agents sont des *systèmes multi-agents* (SMA). Dans la définition 6.2, nous retenons surtout celle proposée par Brahim CHAIB-DRAA. Ainsi, la différence entre une collection d'agents (par exemple les agents d'interface ou les agents sur Internet) et un système multi-agents réside essentiellement dans la capacité qu'ont les agents à interagir pour résoudre collectivement

un problème. De plus, nous considérons que la définition proposée par Jacques Ferber est trop réductrice de par la nécessité qu'ont les agents d'être situés. Toutefois, elle reste intéressante de part sa définition ensembliste d'un sma.

— Définition 6.2 : Système multi-agents —

« *On appelle système multi-agents (ou* sma*), un système composé des éléments suivants :*

1. *un environnement E i.e., un espace disposant généralement d'une métrique ;*
2. *un ensemble d'objets O. Ces objets sont situés i.e., pour tout objet il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs i.e., qu'ils peuvent être perçus, créés, détruits et modifiés par les agents ;*
3. *un ensemble A d'agents, qui sont des objets particuliers ($A \subseteq O$), lesquels représentent les entités actives du système ;*
4. *un ensemble de relations R qui unissent des objets et des agents entre eux ;*
5. *un ensemble d'opérations Op permettant aux agents de A de 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.*

» *[* Ferber*, 1995]*

« *Un système multi-agents (ou* sma*) est un système distribué composé d'un ensemble d'agents (...) interagissant selon des modes de coopération, de concurrence et de coexistence [* Chaib-draa*, 1994 ;* Chaib-draa*, 1996 ;* Moulin *et* Chaib-draa*, 1996]. Un* sma *se distingue d'une collection d'agents indépendants par le fait que les agents interagissent en vue de réaliser conjointement une tâche ou d'atteindre conjointement un but particilier. » [* Chaib-draa *et al., 2001]*

—

Les sma sont à l'intersection de plusieurs domaines scientifiques :
- les *systèmes distribués* sont à la base des sma [Fagin et al., 1995],
- les *interfaces homme-machine* permettent aux agents d'interragir avec les êtres humains (agents sur Internet, agent d'interface, ...) [Kobsa, 1989 ; Negroponte, 1995],
- les *bases de données ou de connaissances distribuées coopératives* regroupent l'en-

semble des mécanismes et des concepts nécessaires aux agents pour mettre en œuvre les processus de raisonnements[BABIN et al., 1997],
- les *systèmes pour la compréhension du langage naturel* sont utilisés pour formaliser les messages échangés par les agents [ALLEN et PERRAULT, 1980 ; COHEN et PERRAULT, 1979 ; GALLIERS, 1988],
- les *protocoles de communication et les réseaux de télécommunication* forment la base nécessaire à la communication entre des entités informatiques autonomes [NWANA et NDUMU, 1999],
- la *programmation orientée agents et le génie logiciel* aident à la conception et au déploiement des SMA [JENNINGS, 2000 ; SHOHAM, 1993a ; THOMAS, 1993],
- la *robotique cognitive et la coopération entre robots* est une branche scientifique spécifique qui permet d'appliquer, vérifier et étudier certains comportements des SMA [LESPÉRANCE, 1991],
- les *applications distribuées* sont des applications idéales pour les SMA (web, internet, ...) [CHAIB-DRAA, 1995 ; JENNINGS et WOOLRIDGE, 1998b],
- mais aussi des disciplines connexes comme la sociologie, la psychologie sociale ou les sciences cognitives qui permettent de poser les bases qui permettront aux agents de raisonner et de communiquer.

6.2.3 Agent cognitif

Par extension de la définition générale d'un agent, on appelle agent cognitif une entité informatique définie comme suit par Mickael WOOLRIDGE [WOOLRIDGE et JENNINGS, 1995] :

— DÉFINITION 6.3 : AGENT COGNITIF —

Un agent cognitif est un agent capable d'autonomie et de flexibilité dans le choix de ses actions pour atteindre un objectif. Cette capacité inclue (...), la pro-activité (...), et la sociabilité.

Un agent cognitif se distingue donc de la notion générale d'agent par le fait qu'il possède un mécanisme lui permettant de *raisonner pour choisir* les actions à mettre en œuvre pour atteindre ses objectifs. Un agent cognitif peut être défini par sa capacité à posséder un comportement dirigé par des buts (architecture BDI proposée par [RAO et GEORGEFF, 1995]). Un agent est dit *pro-actif* lorsqu'il peut prendre l'initiative de modifier ses objectifs pour atteindre ceux qui lui ont été donnés durant sa conception. Ainsi, un agent est capable de s'attribuer de nouveaux objectifs s'il « estime » qu'ils sont nécessaires pour atteindre ses objectifs initiaux (ceux définis par le concepteur de l'agent). Un agent cognitif peut aussi mettre en œuvre des

processus sociaux lui permettant d'interagir avec d'autres agents afin de satisfaire ses objectifs.

Les agents cognitifs peuvent mettre en place des mécanismes de planification [Allen et al., 1990 ; Wilkins, 1988]. Cette dernière vise à élaborer et à ordonner une liste d'actions à réaliser pour atteindre un objectif. Durant les années 70, la communauté scientifique s'est surtout intéressée aux modèles symboliques de l'environnement, aux spécifications symboliques des actions pouvant être réalisées par les agents, aux algorithmes de planification [Fikes et Nilsson, 1971]. Le principal objectif de ces travaux était de démontrer l'efficacité des algorithmes de planification. Mais, face aux problèmes de performances sur des systèmes de taille réelle, certains chercheurs ont exploré une autre voie en remettant en question les bases de l'intelligence artificielle symbolique [Chaib-draa et al., 2001].

6.2.4 Agent réactif

Parmi les critiques les plus virulents, Rodney Brooks exposa son opposition aux modèles symboliques et proposa une approche alternative : l'intelligence artificielle réactive [Brooks, 1986 ; Brooks, 1991a ; Brooks, 1991b]. En effet, il estimait que les comportements intelligents devaient émerger des interactions entre des comportements plus simples. Ainsi, cette nouvelle approche est à la base des systèmes multi-agents réactifs. Dans ces systèmes, l'environnement E prend une place prépondérante. Dans un grand nombre d'entre eux, les agents communiquent uniquement au travers de E. D'autre part, les agents sont bâtis sans utiliser de représentation symbolique ni de mécanisme de raisonnement. Dans cette *architecture subsumption*, les agents ont des comportements correspondants à des machines à états finis, et réagissent aux stimuli provenant de leur environnement.

— Définition 6.4 : Agent réactif —

Les agents réactifs basent leurs décisions entièrement sur le moment présent, sans aucune référence à des actions ou des événements passés. Ils répondent simplement à des stimuli provenant de l'environnement.

Nous utilisons la définition 6.4 proposée par Gerhard Weiss [Weiss, 1999].

Dans la suite de ce chapitre, nous présentons un peu plus en détail les approches de conception de sma. Ensuite, nous décrivons les raisons qui nous ont poussé à choisir les technologies et les concepts des systèmes multi-agents pour répondre à notre problème de modélisation et de simulation de systèmes industriels distribués.

6.3 Approche « Voyelles »

La modélisation et la conception d'un SMA sont des problèmes majeurs et très délicats. En effet, la communauté SMA n'a pas encore trouvé de consensus pour définir ce qu'est un SMA. Ainsi, les différences entre les visions d'un SMA résident essentiellement dans l'approche de modélisation et de conception de ce dernier.
Par exemple, la plateforme MADKIT [GUTKNECHT et al., 2000] considère qu'un SMA peut être défini en partant de la définition de l'organisation des agents. Cette approche est nommée AALAADIN [FERBER et GUTKNECHT, 1998]. Elle est basée sur les concepts de *rôle*, de *groupe* et d'*agent*. Ainsi, toute la conception du SMA est guidée par la définition de ces trois composantes. Toutes les autres composantes d'un système multi-agents (environnement, comportement des agents, ...) sont laissées au choix du concepteur du SMA.
L'approche de conception qui nous semble la plus intéressante est l'approche « Voyelles » (ou AEIO) [DEMAZEAU, 1995 ; DEMAZEAU, 1997] qui a été adoptée et reprise par l'équipe *Systèmes Multi-Agents* de l'ENSM/SE. En effet, elle distingue quatre dimensions : Agent, Environnement, Interaction, Organisation, pour analyser, concevoir et décrire un SMA. Ne s'agissant pas d'une méthodologie, il est possible de l'utiliser en mettant l'accent sur n'importe laquelle des dimensions. Ainsi, là où l'approche de MADKIT force à aborder la conception d'un SMA par l'aspect organisationnel, l'approche AEIO laisse le choix de la ou des dimensions à mettre en avant.
Notons aussi que l'approche AEIO trouve plutôt son intérêt et sa puissance dans la conception de SMA cognitifs sans toutefois s'y restreindre. Notre problématique est de modéliser des systèmes industriels selon l'approche systémique proposée par Jean-Louis LEMOIGNE. Cette approche considère notamment la dimension organisationnelle de tels systèmes. Les centres de décision la composant doivent mettre en œuvre des processus de prise de décision qui peuvent être complexes. Ainsi, l'approche « Voyelles » permet d'intégrer des agents cognitifs qui mettront en œuvre les processus organisationnels du système industriel modélisé.
Un autre attrait de l'approche AEIO est sa vision générale des SMA. En effet, elle permet de considérer la conception d'un SMA selon la dimension qui semble la plus intéressante sans toutefois en privilégier une. Nous pensons que cette vision facilitera la conception d'un SMA qui répondra à notre problématique initiale. La réalisation d'un environnement de modélisation et de simulation de systèmes industriels distribués est un objectif complexe et long à mettre en œuvre. Ainsi, nous ne savons pas encore si une dimension doit être privilégiée par rapport à une autre. Par conséquent, l'approche « Voyelles » nous permet d'utiliser toute la puissance des systèmes multi-agents sans restreindre les possibilités méthodologiques de conception.
Malheureusement, les plateformes SMA qui supportent l'approche « Voyelles » sont

encore rares. Citons MAST (« Multi-Agent System Toolkit ») qui est en cours de développement au sein de notre laboratoire *Systèmes Multi-Agents* [Boissier et al., 1998]. Mais nous ne considérons pas ce manque d'infrastructures logicielles comme un inconvénient rédhibitoire. En effet nous pensons que le processus de simulation peut s'appuyer sur des outils existants (Arena®, Simple++®, ...). Il est donc possible de ne pas utiliser de plateforme SMA existante en réalisant une implantation simple d'une architecture respectant les modèles qui permettront de mettre en œuvre le processus de simulation.
Nous présentons maintenant les quatre dimensions de l'approche « Voyelles ».

6.3.1 Dimension Environnement

L'environnement est le monde dans lequel les agents sont plongés. Cet environnement contient l'ensemble des objets passifs pouvant être perçus ou actionnés par les agents. Cette dimension reste encore peu étudiée. La principale utilisation actuelle est la modélisation des environnements situés, c'est-à-dire des représentations de territoires sur lesquels les agents évoluent [Magnin, 1996 ; Decker et Lesser, 1993 ; Lardon et al., 2000].

6.3.2 Dimension Interaction

La dimension Interaction représente les possibilités d'interaction offertes aux agents. Ainsi [Ferber, 1995] définit les interactions comme « la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques », et [Foisel, 1998] comme « l'enchaînement d'échanges d'informations ou d'influences ayant lieu entre des agents ». Nous privilégions cette dernière définition.
Selon [Proton, 2002], il existe trois types d'interactions avec communication : par signaux, par échanges de messages, sophistiquées.

6.3.2.1 Communications par signaux

Les signaux sont des informations qui circulent dans le SMA. Un exemple typique de ce type de communication est le comportement d'agents réactifs. En effet, les agents recoivent des stimuli et répondent par des actions qui correspondent chacuns à des signaux reçus et émis [Drogoul, 1993 ; Drogoul et Gerber, 1994 ; Demazeau, 1995]. Ces signaux doivent transiter par l'intermédiaire de l'environnement. Ainsi, dans une communication par signaux, il n'y a pas d'interaction directe en deux agents. Le rôle de chaque signal est défini au sein des agents lors de leur conception. Certains utilisent les signaux pour synchroniser les agents [Georgeff, 1988].

6.3.2.2 Communications par échanges de messages

Ces communications permettent aux agents d'échanger des informations, ou des plans d'actions. Citons par exemples les langages à acteurs qui utilisent ce type d'interaction [HEWITT, 1977]. À la vue de l'évolution des techniques et des travaux actuels, les communications par échanges de messages tendent vers les communications sophistiquées [PROTON, 2002].

6.3.2.3 Communications sophistiquées

Les communications sophistiquées sont initialement issues de travaux sur les langues naturelles, sur les dialogues entre l'homme et la machine, et sur les intentions dans les communications. L'évolution sans cesse croissante des capacités de communication a permis d'introduire ce type de communication au sein des interactions entre entités informatiques. Cela s'est d'autant plus accéléré que l'utilisation des réseaux informatiques s'est faite de plus en plus importante. En effet les programmes fonctionnant au sein de réseaux ont besoin d'échanger des informations et, par évolution naturelle, de mettre en œuvre des processus de communication de plus en plus complexes. Les communications sophistiquées sont basées sur des langages et des protocoles.

Langage :

Le langage est généralement basé sur la théorie des actes de langages [SEARLE, 1969 ; SEARLE, 1972]. Malheureusement, les actes de langages se limitent à une action et ne traitent pas une séquence d'interactions [BRASSAC, 1993]. Le nombre de langages est relativement important et peu homogène : KQML [FININ et al., 1994], ARCOL [SADEK, 1991], ASIC [BOISSIER et DEMAZEAU, 1994], COOL [BARBUCEANU et FOX, 1995], ... La « Foundation for Intelligent Physical Agents » (ou FIPA) tente cependant d'uniformiser la notion de langage en proposant l'« Agent Communication Language » [FIPA, 1998].
C'est en 1975 que John AUSTIN se penche sur l'énonciation et sa définition comme un acte qui sert avant tout à produire des effets sur son destinataire [AUSTIN, 1975]. Ainsi, un acte de langage désigne l'ensemble des actions intentionnelles effectuées au cours d'une communication. Par conséquent, cette dernière est une action dans laquelle nous trouvons un locuteur qui émet un message et un allocuteur que le reçoit.
Un acte de langage est défini par John AUSTIN et ses successeurs [SEARLE, 1969 ; SEARLE, 1972] comme une composition de trois éléments :

– **Composante locutoire :**

  Elle représente le mode de propagation du message (oral, écrit, ...). Dans le cadre des travaux sur les SMA, nous considérons que cette composante est

strictement limitée aux réseaux informatiques de communication.

– **Composante illocutoire :**

Cette composante représente l'intention de l'acte. Elle est généralement exprimée sous la forme $\mathcal{F}(p)$ où $\mathcal{F}$ est la *force illocutoire* et p est le *contenu propositionnel*. $\mathcal{F}$ est souvent représenté par un verbe appelé *performatif*. Par exemple, `Commander(Ferme la fenêtre !)` est un message dont le performatif est `Commander` et le contenu propositionnel `Ferme la fenêtre !`.

John Searle propose de décomposer la force illocutoire en six champs :

1. le *but illocutoire* qui définit la catégorie de l'acte de langage. Nous retrouvons dans ce champ la direction d'ajustement d'un acte (direction que nous présentons dans la suite de cette section) ;
2. le *mode d'accomplissement* décrit la manière selon laquelle le but doit être accompli. Ainsi le performatif `Commander` nécessite une notion d'autorité entre le locuteur et l'allocuteur ;
3. les *conditions du contenu propositionnel* expriment un lien entre $\mathcal{F}$ et p. Par exemple, si $\mathcal{F}$ exprime un objectif directif, alors le contenu propositionnel doit exprimer une action ;
4. les *conditions préparatoires* permettent d'informer l'allocuteur des présuppositions du locuteur dans le contexte de l'acte ;
5. les *conditions de sincérité* représentent les conditions pour lesquelles le locuteur est sincère ;
6. le *degré de puissance* correspond à l'intensité de la force illocutoire. Il dépend du but illocutoire et des conditions de sincérité.

– **Composante perlocutoire :**

La composante perlocutoire exprime les attentes du locuteur quant aux conséquences de l'acte de langage. Elle prend deux aspects selon que l'on se place du côté du locuteur ou de l'allocuteur. Du côté du locuteur, une attente sur l'état du monde et sur l'allocuteur est exprimée. Le locuteur attend la réalisation d'une action future ou d'un nouvel état mental selon le type de l'acte de langage (assertif, directif, ...). Du côté de l'allocuteur, ce dernier doit être capable de reconnaître l'intention exprimée dans la composante perlocutoire afin de produire l'effet attendu. De plus, cette composante permet à l'allocuteur d'élaborer des croyances sur le locuteur. Cette composante est souvent exprimée par des conditions de succès qui permettent une reconnaissante sans ambiguïté de l'intention du locuteur [Chaib-Draa et Vanderveken, 1998].

Les actes de langage peuvent être classés selon cinq catégories selon leur contenu propositionnel. Un but :

- *assertif* permet de décrire l'état du monde (« La fenêtre est fermée. ») ;
- *directif* représente un acte imposant à l'allocuteur de changer l'état du monde (« Ferme la fenêtre ! ») ;
- *promissif* correspond à engager le locuteur à réaliser une action future (« Je vais fermer la fenêtre ») ;
- *déclaratif* permet d'accomplir quelque chose par le seul fait d'énoncer l'acte de langage (« Je ferme la fenêtre ») ;
- *expressif* permet de décrire les états mentaux du locuteur (« J'aime les fenêtres fermées »).

Protocole :

Comme nous l'avons mentionné ci-dessus, les langages ne permettent pas de tenir compte d'un ensemble d'interactions. Les protocoles permettent de répondre à ce problème. Ils sont définis par la FIPA comme des topologies typiques d'interactions [FIPA, 1998]. Ainsi, lorsqu'un protocole est utilisé, l'agent peut s'attendre à rencontrer un certain nombre d'interactions (celles-ci étant présente dans le protocole). Tout comme pour les langages, il existe un grand nombre de protocoles : échange de connaissances [CAMPBELL et D'INVERNO, 1990], protocole d'appel d'offre [SMITH, 1980], ... La FIPA propose une spécification pour les protocoles : le « FIPA-query protocol » [FIPA, 1998].

La figure 6.1 illustre un exemple de protocole d'appel d'offre exprimé selon le formalisme « Agent Unified Modelling Language » (ou AUML) [BAUER et al., 2001]. Chaque « boîte » représente un agent différent. Les flèches correspondent à des messages envoyés d'un agent vers un autre. Dans ce protocole, nous pouvons voir que l'initiateur de l'appel d'offre (`initator`) réalise un appel à participation (`cfp`) auprès de participants. Ces derniers peuvent répondre avec trois messages différents :
- `not-understood` : le participant n'a pas compris l'appel à participation,
- `refuse` : le participant a compris l'appel mais ne désire pas y participer,
- `propose` : le participant a compris l'appel et désire y participer. Pour cela, il réalise une proposition auprès de l'initateur.

Ensuite, l'initateur compare les propositions qui lui sont parvenues et choisit le participant le plus avantageux (envoi du message `accept-proposal`). Les autres participants sont alors informés que leur proposition a été rejetée (message `reject-proposal`). Enfin, le protocole d'appel d'offre se termine par une notification de la part du participant sur le statut de l'appel d'offre : `failure` quand il a rencontré un échec durant la réalisation de sa proposition, et `inform-done` ou `inform-ref` lorsqu'il a terminé sa tâche.

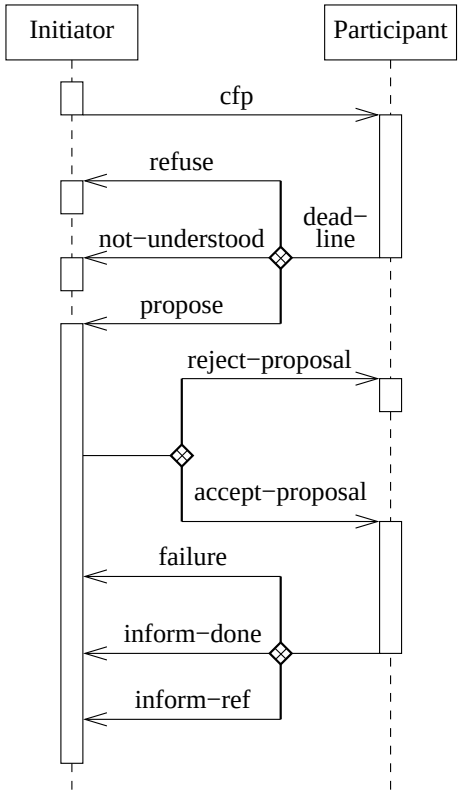


Fig. 6.1 – Protocole d'appel d'offre [FIPA, 2001a]

6.3.3 Dimension Organisation

La dimension Organisation regroupe l'ensemble des moyens et des contraintes permettant de structurer le système multi-agents. Selon [Hannoun, 2002], « l'organisation dans sa spécification la plus simple peut être définie comme une restriction de l'autonomie (*capacité des agents à prendre seul des décisions*) des agents de manière à assurer à des degrés divers leur coopération pour un but donné. Un modèle organisationnel est une spécification formalisée de ces contraintes ».

L'organisation représente l'ensemble des liens entre les agents. Ces relations peuvent être issues d'une *structure organisationnelle* [Hannoun et al., 1998 ; Pattison et al., 1987], ou issues de dépendances calculées à partir des actions, des ressources et des buts des agents [Allouche et al., 1997 ; Allouche, 1998 ; Sichman et Demazeau, 1995]. Une relation est dite relation de dépendance si, pour atteindre ses objectifs, un agent doit nécessairement avoir recours à un autre agent. Ces relations peuvent être formalisées par la théorie des dépendances proposée par [Sichman et al., 1994 ; Sichman et Demazeau, 1995]. La communauté scientifique des sma s'est penchée sur la mise en place de structures organisationnelles. Souvent inspirées par la société humaine, ces structures incluent des termes bien connus. Citons l'exemple d'Aalaadin qui définit une organisation à partir des notions de

rôle et de groupe [FERBER et GUTKNECHT, 1998], ou encore MOISE [HANNOUN et al., 2000 ; HANNOUN, 2002]. Nous nous appuyons sur cette dernière car elle nous semble pouvoir modéliser le plus fidèlement possible les relations au sein du sous-système décisionnel d'un système de production. En effet, MOISE permet de créer des modèles organisationnels à partir de rôles, de tâches associées aux rôles, de groupes, de liens organisationnels (liens d'autorité, d'accointance, et de communication). Selon Mahdi HANNOUN, il existe bien sûr d'autres organisations comme les tableaux noirs [ENGELMORE et MORGAN, 1988], les réseaux contractuels [SMITH, 1980], la théorie du raisonnement social [SICHMAN et DEMAZEAU, 1995], les treillis de production [GASSER et al., 1989], ou les lois sociales [SHOHAM et TENNENHOLTZ, 1993b].

6.3.4 Dimension Agent

La dimension A (pour Agent) désigne l'ensemble des fonctionnalités de l'agent. Ainsi un agent est capable de raisonner, c'est-à-dire de planifier ses actions, et d'introspecter afin de raisonner sur ses propres compétences et connaissances [CORKILL et LESSER, 1983 ; ROSENSCHEIN et GENESERETH, 1985]. D'autre part, un agent est capable d'interagir avec l'environnement ou avec d'autres agents [COHEN et LEVESQUE, 1995 ; CARRON et al., 1999 ; POPULAIRE et al., 1993]. Pour cela, il met en œuvre les protocoles de communication définis dans la dimension Interaction. Un agent peut aussi intégrer les aspects organisationnels (dimension O) dans son processus de raisonnement [SICHMAN et al., 1994]. Ainsi, un agent possède souvent une représentation des autres [KESSER, 1997 ; VERCOUTER, 2000] et de son environnement afin de pouvoir réaliser ses calculs de planification pour atteindre ses objectifs.

Il n'existe pas de concensus sur la méthode ou sur la technique à utiliser pour définir un agent. Toutefois, le modèle BDI (« Belief, Desire, Intention ») est reconnu par une partie de la communauté comme un modèle intéressant. Afin de clarifier ce qui se cache derrière la dimension Agent, nous nous proposons d'expliquer brièvement l'architecture BDI [RAO et GEORGEFF, 1995].

L'architecture BDI propose que l'état d'un agent soit représenté par un triplet représentant son *état mental*. Ce triplet correspond aux croyances (« belief »), aux désirs (« desire ») et aux intentions (« intention ») d'un agent. Les croyances sont les représentations que l'agent se fait de son environnement et de lui-même. Les désirs (aussi appelés objectifs ou « goals ») sont ce vers quoi l'agent veut tendre. Enfin les intentions sont composées des actions que l'agent a décidé de faire pour atteindre ses objectifs [BRATMAN, 1987 ; BRATMAN et al., 1988 ; COHEN et LEVESQUE, 1990].

[NORMAN et LONG, 1996] propose la figure 6.2. Elle correspond à l'architecture

BDI 1.1 et illustre les raffinement successifs pour aboutir à des intentions simples et des actions directement exécutables.

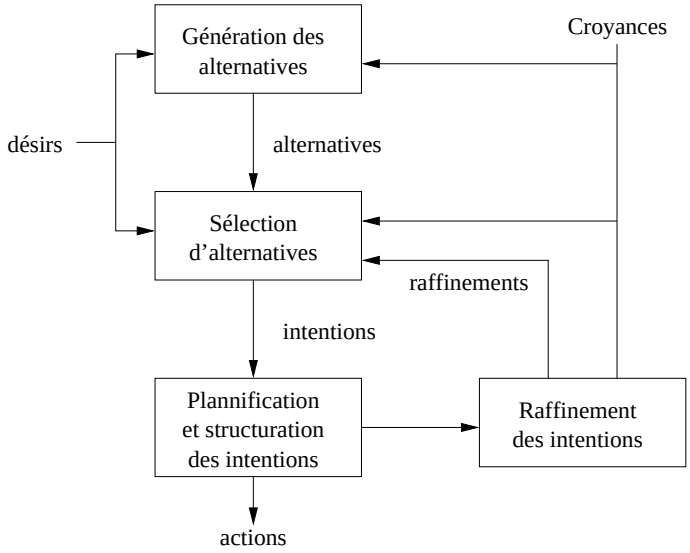


Fig. 6.2 – Architecture BDI [Norman et Long, 1996]

6.4 Conclusion

Les systèmes multi-agents sont une extension de l'intelligence artificielle au sein d'un environnement de plus en plus distribué. Héritiers des concepts orientés objets et de l'intelligence artificielle distribuée, les SMA sont de plus en plus utilisés dans des domaines très variés : la modélisation et la simulation des prises de décision dans les systèmes industriels [Burlat et al., 1998 ; Kabachi, 1999], le pilotage de systèmes industriels [Berchet, 2000], la simulation d'ateliers de production au sein d'un environnement de réalité virtuelle [Duval et al., 1997], ou la simulation de processus naturels [Burkhart, 1994], ... Comme l'illustrent ces divers exemples, les SMA sont fortement utilisés pour modéliser et simuler le comportement des centres de prise de décision des systèmes industriels. Ce fait est notamment dû à l'adéquation des systèmes multi-agents avec la modélisation d'entreprise et leurs simulations. Ils ont les avantages suivants vis-à-vis de la décomposition systémique proposée par Jean-Louis Lemoigne :

– **pour le sous-système décisionnel :**

  Le sous-système décisionnel est certainement celui qui bénéficie le plus des apports des systèmes multi-agents. En effet, comme l'illustrent de nombreux travaux [Burlat, 1996 ; Kabachi, 1999 ; Bournez, 2001], les SMA permettent une modélisation fine et naturelle des structures organisationnelles du système de production. Ainsi les travaux en génie industriel et en économie

visant à définir la nature de cette structure mettent en avant des concepts qui peuvent être rapprochés de ceux proposés par une approche de modélisation organisationnelle multi-agents (AALAADIN, MOISE, ...). En effet, les structures organisationnelles sont composées de centres de prise de décision qui mettent en œuvre des mécanismes, des méthodes et des algorithmes leur permettant de prendre des décisions en fonction de données portées à leur connaissance. Ces centres sont donc fonctionnellement très proches des agents mettant en œuvre une architecture comme BDI. De même, la définition des structures organisationnelles au sein des entreprises a mis en avant les liens pouvant exister entre les centres de décision. Ces liens peuvent être verticaux (autorité), horizontaux (coopération) ou d'accointance. L'analogie avec les structures organisationnelles au sein des SMA est visible dans le cas où nous considérons un modèle MOISE. Il semble naturel que les systèmes multi-agents sont d'un grand intérêt pour la modélisation et la simulation des entités composant le sous-système décisionnel. D'autre part, les riches capacités d'interaction des agents permettent de représenter les différents messages pouvant être échangés au sein d'un système aussi complexe qu'une entreprise. Considérant que nous désirons représenter l'ensemble des interactions entre les entités composant le sous-système décisionnel (chaque centre correspond à au moins un agent), les actes de langage nous semblent une approche efficace pour représenter les ordres, les demandes, ou les diffusions d'information.

– **pour le sous-système informationnel :**

Tout comme pour le sous-système précédent, les agents sont un moyen pour supporter la distribution des modèles de simulation. Chaque composante du système industriel distribué peut alors posséder ses propres données et, s'il le désire, les partager avec d'autres composantes. Les capacités interactionnelles des agents (actes de langages, ...) nous semblent très intéressantes pour supporter l'échange de données complexes telles que les nomenclatures ou les gammes. Même si la solution d'un format d'échange normalisé (comme XML) existe déjà pour répondre à cette problématique, l'utilisation d'un système multi-agents nous permettra de mettre en place des conversations complexes d'échanges de données.

– **pour le sous-système opérationnel :**

Nous rappelons que le sous-système opérationnel représente l'ensemble des infrastructures physiques du système de production. Dans le cas d'un système distribué, telle une entreprise en réseau, cette infrastructure peut être répartie dans des lieux géographiques différents. Cette répartition spatiale peut avoir pour conséquence de donner naissance à plusieurs modèles de simulation. Dans notre exemple d'une entreprise réseau, chaque participant au groupement peut

posséder un modèle de son sous-système opérationnel. La simulation d'un tel groupement se faisait jusqu'à maintenant en centralisant et en regroupant les différents modèles. Mais que faire lorsqu'un membre désire garder confidentielle une partie ou la totalité de son modèle ? Ce problème nous permet d'illustrer l'intérêt d'une approche distribuée de simulation. La discipline de la simulation distribuée permet de répondre à cette problématique, et notamment de répondre aux problèmes techniques de synchronisation des modèles. Malheureusement la simulation distribuée possède ses propres limites. En effet, elle supporte difficilement la dynamique de composition d'un groupe. Il est impossible à de nouveaux arrivants de s'intégrer dans le groupement s'ils ne respectent pas les méthodes de modélisation utilisées par les autres membres (protocoles de communication entre les modèles). À l'instar de HLA [US Department of Defense, 1996], les SMA peuvent apporter un canevas permettant de normaliser les communications entre les différents modèles de simulation. Mais contrairement à HLA, ils supportent plus facilement les retraits et les arrivées de modèles au sein du groupement.

Les agents composant le modèle du sous-système opérationnel reçoivent des ordres de fabrication de leur environnement, et échangent des lots de fabrication. Ils peuvent être pilotés par des agents appartenant au sous-système décisionnel [Berchet, 2000]. Toutefois l'attrait majeur des SMA ne se situe pas réellement au niveau du sous-système opérationnel. En effet, il peut être considéré comme faisant partie de la dimension environnementale du système. Les agents composant le modèle du système de production auraient alors des droits d'accès plus ou moins restreints sur cette ressource. Cette vision nous permet entre autres d'utiliser des outils de simulation ayant déjà fait leurs preuves (Arena®, Simple++®, ...). Ces derniers peuvent représenter le sous-système opérationnel et peuvent être considérés comme des ressources au sein d'un système multi-agents.

À la vue de l'ensemble des avantages apportés par les systèmes multi-agents pour répondre à notre problématique de modélisation et de simulation des systèmes industriels distribués, nous définissons dans la suite de ce mémoire une architecture SMA adaptée à nos objectifs.

Dans le chapitre suivant, nous résumons l'ensemble des concepts, des méthodes et des méthodologies que nous avons présenté durant toute cette partie. Nous décrivons brièvement la démarche qui en partant de la modélisation en entreprise, nous a poussé à considérer les systèmes multi-agents comme une solution à notre problématique.

Chapitre 7

Conclusion

L'évolution des organisations industrielles a poussé la communauté scientifique à étudier de nouvelles approches de modélisation et de simulation. Ces nouvelles organisations correspondent à ce que nous qualifions de systèmes distribués : réseaux d'entreprises [Goranson et al., 1997 ; Ettinghoffer, 1992], entreprises virtuelles [Nunes, 1994]. Nous avons défini ces nouvelles organisations dans la section 2.1.3.
Ainsi, les entreprises ont un nouveau besoin en termes de modélisation et de simulation. Il est donc nécessaire de mettre en œuvre de nouvelles méthodologies et de nouveaux outils permettant d'y répondre. Notre objectif est non seulement de réaliser un environnement prenant en compte ces nouvelles organisations, mais aussi de tenir compte de l'existant. Notamment, nous ne désirons pas créer un nouvel outil de simulation, mais nous préférons adapter ceux qui existent déjà afin qu'ils puissent s'intégrer dans le nouvel environnement. Cet exemple est assez illustratif de nos objectifs du point de vue technique. En effet, nous désirons adapter l'existant à la résolution de notre problématique, avant de créer de nouvelles approches. Ceci permettra de rentabiliser le savoir-faire et les compétences en simulation qui ont été acquises par les entreprises. Nous avons étudié deux disciplines connexes : la modélisation en entreprise et la simulation (et plus particulièrement la simulation distribuée).

7.1 Modélisation en entreprise

La modélisation en entreprise, née des besoins issus de l'accroissement de la complexité des entreprises, est une discipline qui a pour objectif de pouvoir modéliser des entreprises (et, dans notre cas, des systèmes industriels) en tenant compte des nombreux facteurs entrant en ligne de compte dans leurs comportements réels. Ces facteurs peuvent être issus de l'économie, de la gestion de production, ...
Du point de vue de notre problématique, nous voulons modéliser un système indus-

triel distribué. Il nous à donc paru intéressant d'étudier les différentes méthodologies et formalismes proposés dans le cadre de la modélisation en entreprise. En prenant pour canevas les quatre problématiques majeures qui nous intéressent (*cf.* section 2.2), nous décrirons ce qui est intéressant, et ce qui l'est moins dans cette discipline.

Toutes les méthodes de modélisation en entreprise utilisent souvent des concepts communément admis (processus, activité, ...). Malheureusement, chacune d'entre elles utilise une approche différente. Par exemple, IDEF0 [US AIR FORCE, 1993b] est un langage de modélisation par le biais de processus. Un autre exemple est CIMOSA [AMICE, 1993] qui est une approche générale de modélisation. Mais comme le fait remarquer [VERNADAT, 1998], CIMOSA n'a pas pu percer dans le monde industriel par son manque de représentation graphique. Ainsi, la modélisation en entreprise propose un ensemble de méthodes qui permettent de représenter un système industriel. Toutefois, nous pouvons dégager un point commun : une approche systémique de modélisation. L'approche la plus connue est celle de Jean-Louis LE MOIGNE [LE MOIGNE, 1992]. Elle propose une décomposition d'une entreprise en trois sous-systèmes :

- le sous-système opérationnel correspond à l'ensemble des infrastructures physiques composant le système (ateliers, machines, ressources, ...) ;
- le sous-système décisionnel représente l'ensemble des centres de décision, c'est-à-dire des entités pilotant le sous-système opérationnel. En particulier, on distingue trois niveaux dans ce sous-système : le niveau opérationnel (pilotage direct des objets du sous-système opérationnel), le niveau tactique (pilotage à moyen terme des centres opérationnels de décision), et le niveau stratégique (pilotage à long terme des centres tactiques) ;
- le sous-système informationnel qui contient l'ensemble des informations nécessaires aux deux autres sous-systèmes pour communiquer ou pour réaliser leurs tâches. En général, on y trouve les définitions des gammes, des nomenclatures, ...

Nous retenons aussi l'utilisation d'une démarche de modélisation consistant à réaliser une analyse du domaine, à spécifier le modèle de l'entreprise et à concevoir un modèle exécutable ou utilisable par des outils informatiques. L'ensemble des méthodes exposées dans ce mémoire prennent en compte un ou plusieurs sous-systèmes. Par exemple, chaque langage IDEF [US AIR FORCE, 1993a] est plutôt spécialisé dans un sous-système : IDEF0 et IDEF3 pour le sous-système opérationnel, IDEF1 [MAYER, 1992] pour le sous-système informationnel. Quant aux méthodes plus généralistes (CIMOSA, GRAI, PERA, ...), elles supportent les trois sous-systèmes.

Le second problème est le grand nombre d'artefacts de modélisation existants. Cette hétérogénéité des concepts rend difficile toute communication entre les diverses

méthodologies de modélisation en entreprise. Ainsi, pour résoudre ce problème, une partie de la communauté a donné naissance au projet de normalisation « Unified Enterprise Modelling Language » [Vernadat, 2001]. Son objectif est de fournir une ontologie pouvant être utilisée par toutes les méthodologies de modélisation en entreprise. Ce projet est tout à fait en adéquation avec notre objectif de normalisation des éléments de modélisation à une exception près : la modélisation en entreprise se place à un niveau différent de celui de la simulation. En général, cette dernière utilise des concepts plus proches des artefacts physiques (machines, ateliers, ressources humaines, ...) que ceux de la modélisation d'entreprise (processus, activité, ...). Ainsi, si l'effort d'UEML nous semble prometteur et intéressant, il nous faut nous replacer dans notre contexte de travail plus particulier à la simulation. Toutefois, nous gardons en mémoire cette approche pour l'utiliser et nous en inspirer.

Le troisième problème est le support des concepts issus de l'apparition des nouvelles organisations. Ici, nous ne pensons pas qu'il s'agisse de mettre en évidence de nouveaux artefacts de modélisation, mais plutôt de mieux comprendre et modéliser les interactions pouvant exister entre les différentes composantes d'un système distribué. Ainsi, nous pensons que ce point de vue est plus du ressort de la modélisation pour la simulation que de la modélisation en entreprise. En effet, la simulation distribuée propose des techniques pour modéliser et simuler des systèmes distribués. En ce qui concerne les concepts attachés à la modélisation d'entreprises distribuées (types de liens entre les membres, protocoles de discussion, types de contrats, ...) et adaptés aux nouvelles organisations, nous pensons que réaliser une étude dans ce domaine est en dehors du cadre de travail que nous nous sommes fixé. En effet, nous avons décidé de limiter nos recherches aux outils techniques et au développement d'une architecture logicielle capable de supporter les nouvelles organisations d'entreprises. Plus clairement, nous nous attacherons essentiellement aux problématiques de la simulation plutôt qu'à celles de la modélisation en entreprise. Ces dernières pourront faire l'objet de futurs travaux dans le cadre de notre approche méthodologique.

7.2 Méthodologies de simulation

La simulation est l'une des disciplines connexes et complémentaires à la modélisation en entreprise. Elle permet d'évaluer le comportement dynamique d'un système industriel (aléas, allocation des ressources, ...). Très utilisée dans le monde industriel, la simulation est une technique plus aisée à comprendre et à mettre en œuvre que les autres techniques de recherche opérationnelle [Kelton et al., 1998 ; Morgan, 1989 ; Shannon et al., 1980]. Pour mettre en avant de telles techniques, de nombreuses méthodes ont vu le jour. Citons celles qui nous semblent les plus intéressantes : la « Conical Methodology » [Nance, 1981] et « Analyse Spécification Conception

et Implantation » [KELLERT et RUCH, 1998b]. Elles correspondent à ce que nous attendons d'une méthodologie pour la simulation :

- elles offrent des approches de modélisation structurées en plusieurs étapes. En général, nous retrouvons les étapes d'analyse (étude du domaine et de faisabilité), de spécification (construction d'un modèle abstrait), et de conception et d'implantation (construction d'un modèle informatique, puis exécutable) ;
- ASCI propose une vision systémique des systèmes de production, ce qui, à nos yeux, augmente les capacités de réutilisabilité, de compréhension et de maintenance des modèles ;
- Elles proposent des méthodes ou des outils capables de répondre aux besoins de chaque étape de la méthodologie.

Malheureusement, ces méthodes ne tiennent pas encore vraiment compte des caractéristiques des systèmes industriels distribués. Ainsi, si nous voulons réaliser la simulation d'un groupement d'entreprises, il est souvent nécessaire de centraliser les informations et de réaliser un seul modèle de simulation. Hors, nous pensons que ce fait va à l'encontre du droit aux membres du groupement à garder confidentielle une partie des informations nécessaires au processus de simulation.

Les nombreux outils composant la constellation industrielle de la simulation sont tous très efficaces : ARENA® [KELTON et al., 1998], SIMPLE++® [TECNOMATIX®, 2001], ... Malheureusement, il s'agit en général d'outils commerciaux pour lesquels aucune méthodologie n'a été envisagée. Ainsi, les utilisateurs de ces outils réalisent des modèles dont la qualité dépend presque uniquement de leurs compétences et de leurs expériences. Une approche méthodologique utilisant ces outils comme base calculatoire semble intéressante et permettrait aux entreprises de réutiliser leurs compétences et leurs ressources en termes de simulation.

Un autre problème rencontré est la possible hétérogénéité des modèles de simulation des différents membres d'un groupement. En effet, il est difficile avec les outils et les méthodes actuels de construire des modèles capables d'échanger des informations avec n'importe quel autre outil de simulation. En général, ces outils se limitent à supporter les sources d'informations possibles (bases de données, tableurs graphiques, ...). Prenons l'exemple d'« High Level Architecture » [US DEPARTMENT OF DEFENSE, 1996] qui propose une architecture pour l'interopérabilité des modèles de simulation. Elle est très intéressante pour mettre en œuvre un processus de simulation d'un système industriel distribué. En effet, elle supporte la synchronisation des modèles, une partie de leur hétérogénéité, et la distribution au sein d'un réseau informatique. Malheureusement, l'hétérogénéité reste aussi du ressort des modèles de simulation qui doivent tenir compte des différents formats d'information utilisés par les autres modèles. Dans un cadre réel, les membres d'un groupement d'entreprises sont susceptibles d'utiliser des outils distincts, ou incompatibles entre eux. Il

est donc nécessaire de mettre en place une architecture supportant non seulement les communications entre les modèles, mais aussi l'interopérabilité des informations échangées. Cet objectif peut être réalisé en utilisant une approche similaire à celle d'UEML : proposer une normalisation des éléments de modélisation et une ontologie propre à la simulation de systèmes industriels distribués.
D'autre part, l'ensemble des approches proposées a une vision statique des modèles de simulation. Ainsi, dans un groupement d'entreprises, les membres peuvent arriver ou partir à tout instant. Il serait par conséquent très intéressant à un environnement de modélisation et de simulation de réagir à ces événements (par exemple en proposant des alternatives de communication, ou en générant des événements dans le sous-système décisionnel). Malheureusement, la plupart des outils et des méthodologies de simulation ne supporte pas les modifications des modèles durant le processus de simulation. Pour changer un modèle du groupement, il faut arrêter le processus de simulation, changer les paramètres, puis relancer la simulation.
L'ensemble de ces problèmes nous a poussé à rechercher une solution supportant les concepts de modélisation et la distribution des modèles de simulation. Nous pensons que les systèmes multi-agents sont une approche adaptée à notre problématique. Dans la section suivante, nous abordons les raisons de ce choix.

7.3 Systèmes multi-agents

Les systèmes multi-agents [Ferber, 1995] sont les héritiers directs de l'Intelligence Artificielle Distribuée [Corkill et Lesser, 1983 ; Gasser, 1992]. Ils représentent des sociétés d'entités douées d'autonomie et d'intelligence, et capables d'interagir entre elles. Comme nous l'indiquons dans la section 6.4, nous pensons que les systèmes multi-agents sont fortement adaptés à notre problématique. En effet, si nous considérons une approche de décomposition systémique d'un système industriel, les systèmes multi-agents possèdent un certain nombre de qualités permettant :

- **pour le sous-système opérationnel :** de supporter la décentralisation géographique des infrastructures physiques.
- **pour le sous-système informationnel :** de distribuer l'information (gammes, nomenclatures, ...) partiellement ou totalement au sein d'un ensemble de modèles de simulation. Ceci est réalisé grâce aux capacités interactionnelles et cognitives des agents ;
- **pour le sous-système décisionnel :** de modéliser les processus de prise de décision d'un système industriel. En effet, comme l'ont illustré [Burlat, 1996] et [Kabachi, 1999], les agents peuvent être utilisés pour modéliser les différents acteurs au sein d'un processus de prise de décision, et cela quel que soit leur niveau (hiérarchique, stratégique ou opérationnel).

Ainsi, ce tour d'horizon de la modélisation en entreprise, de la simulation et des systèmes multi-agents, nous a permis de proposer un environnement de modélisation et de simulation pour la simulation de systèmes industriels distribués (« Multi-Agent Methodological Approach for Simulation » ou $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$). Nous présentons en détail $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$ dans la partie suivante.

Troisième partie

Approche Méthodologique Multi-Agents pour la Simulation ($\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$)

Chapitre 8

Cycle de vie des modèles

La mise en œuvre d'une approche méthodologique, dont la nécessité est expliquée dans le chapitre précédent, passe par une première étape majeure : la définition d'un cycle de vie des modèles. Nous consacrons ce chapitre à définir le cycle de vie des modèles de simulation des systèmes industriels distribués. À partir des acquis du Génie Logiciel et des travaux déjà réalisés dans le domaine de la simulation, nous proposons d'étendre les approches existantes pour qu'elles tiennent compte des nouvelles organisations d'entreprise.

Ainsi, notre approche méthodologique $\mathcal{MAMA}$-$\mathcal{S}$ est basée sur le cycle de vie composé de cinq phases principales [Galland et Grimaud, 2000a] : l'analyse, la spécification, la conception, l'implantation et l'exploitation. Nous décrivons succinctement ces étapes dans la suite du présent chapitre. Mais avant cela, nous rappelons les bases de Génie Logiciel nécessaires à la constitution d'un cycle de vie.

Dans la suite de cette partie, nous utilisons (sauf mention explicite) le terme « formel » selon la définition suivante :

— Définition 8.1 : Formel —

Un artefact est dit formel lorsque sa définition est basée sur un formalisme prédéfini qu'il soit « graphique », orienté-objet ou mathématique. Ainsi, nous nous distinguons de la définition limitant les artefacts formels à ceux sur lesquels il est possible de réaliser une démonstration mathématique. Dans notre cas, nous incluons ces dernières (spécifications algébriques, logique des prédicats du premier ordre, ...), mais aussi les langages de programmation, et les langages de modélisation (graphiques ou non).

8.1 Du génie logiciel...

Le Génie Logiciel est une discipline née du constat de difficultés inhérentes au développement de grands projets informatiques. Elle regroupe un ensemble de méthodes, de méthodologies et d'outils capables de répondre à ces problèmes cruciaux. [SOMMERVILLE, 1998] explique la nécessité de méthodes et d'outils d'ingénierie de la manière suivante :
« Les problèmes rencontrés dans le développement de gros logiciels ne sont pas directement comparables à ceux que l'on rencontre quand on en écrit de "petits". On peut faire une analogie en comparant la construction d'un pont routier sur un estuaire à celle d'une passerelle sur un torrent. Bien que tous les deux soient des "ponts" et qu'ils possèdent en commun certaines propriétés, un ingénieur ne conçoit pas un pont sur un estuaire par simple agrandissement d'une passerelle. La différence de méthode est due au fait suivant : la complexité des "petits" programmes (ou ponts) peut être appréhendée par une seule personne. Cette personne ayant à l'esprit tous les détails de la conception et de la réalisation. Les spécifications peuvent rester informelles et les effets de toute modification sont en général évidents. En revanche, la complexité des grands systèmes est telle qu'il est impossible à un seul individu d'avoir à l'esprit tous les détails du projet et de les mettre à jour. Un processus de spécification et de réalisation plus formelles est alors nécessaire. »
Par ailleurs, le développement industriel de gros systèmes occupe des dizaines d'hommes pendant plusieurs années. Par exemple, le premier compilateur ADA est le résultat de cinq années de travail d'une vingtaine d'hommes. Dans ce cas, il est évident que les méthodes doivent permettre :
– la communication et le partage du travail,
– la continuité : si les membres de l'équipe quittent le projet ou l'entreprise, leurs travaux doivent être repris et poursuivis avec le moins de perturbation possible.

8.1.1 Principes

Selon [GHEZZI et al., 1991], sept principes constituent les fondements du Génie Logiciel :

– **Rigueur et description formelle :**
La tendance naturelle d'une activité créative est d'être ni précise, ni exacte. Dans le cadre de grands projets, il est nécessaire de mettre en œuvre des processus et des méthodes permettant de rendre rigoureux le développement. Une description formelle autorise la mécanisation de certaines parties du développement d'un logiciel (vérification, transformations d'énoncés, ...). Ces descriptions sont basées sur des langages formels (spécification algébrique, logique du premier ordre, langages de programmation, langages orientés objets

de modélisation, ...).

– **Séparation des sujets d'étude :**

Dans les principes de l'organisation scientifique du travail et de la division de celui-ci en tâches, la difficulté consiste à trouver différentes facettes d'un problème qui soient suffisamment indépendantes pour être traitées séparément. Il existe plusieurs visions permettant de réaliser cette division : le temps (utilisation d'un « planning » en fonction des étapes du cycle de vie), la qualité (séparation de l'efficacité et de la validité), et la modularité (partition du logiciel).

– **Modularité :**

La modularité est la capacité d'un problème à être divisé en sous-problèmes ayant un faible couplage. La qualité de la modularité est généralement définie en fonction des critères suivants :

- le regroupement des procédures ayant des relations sémantiques importantes participe à la *cohésion* des modules ;
- la facilité de tester et de réutiliser un module indépendamment des autres correspond au critère de *couplage* des modules ;
- la *décomposition* d'un système complexe doit être possible, chaque sous-système doit avoir une sémantique cohérente ;
- la *réutilisation* de modules existants doit être possible pour élaborer un nouveau module.

– **Abstraction :**

L'abstraction est la capacité d'un problème à être étudié selon une vision abstraite, c'est-à-dire en s'intéressant à ce qu'est le problème plutôt qu'aux moyens à mettre en œuvre pour le résoudre.

– **Anticipation sur les changements :**

Elle conduit à distinguer la construction de logiciels de celle de systèmes en général. Ainsi, ce principe a pour objectif de faciliter la maintenance corrective, et surtout la maintenance évolutive.

– **Généralité :**

Il s'agit du principe qui consiste à généraliser un problème. Cette vision généralisée peut avoir déjà des solutions qui peuvent être utilisées pour répondre au problème initial.

– **Possibilité de modification incrémentale :**

Le développement progressif d'un problème est nommé développement incrémental. Il s'agit de fournir des versions de plus en plus précises par propositions de raffinements successifs du problème.

Toujours selon [Ghezzi et al., 1991], le Génie Logiciel est l'ensemble des principes, des méthodes, des techniques, des outils et des méthodologies du procédé de

développement relatif aux différentes phases de l'élaboration d'un logiciel. Il correspond à un schéma en couches indiquant que les outils sont construits pour aider la mise en œuvre selon une méthodologie, qui applique des méthodes et des techniques définies selon quelques « bons » principes.

8.1.2 Les étapes du cycle de vie

Le cycle de vie d'un logiciel est l'ensemble des phases et des étapes de son développement et de son évolution. Il est en général constitué des étapes suivantes : l'analyse et la définition des besoins, la spécification, la conception, la réalisation, la vérification, l'installation, l'exploitation et la maintenance.

8.1.2.1 Analyse et définition des besoins

Cette phase doit décrire d'une part les fonctionnalités du système et d'autre part les contraintes non fonctionnelles. Elle est établie par consultation des clients. Elle fait l'objet d'un document écrit appelé le *cahier des charges*.
La définition des besoins est le processus visant à établir quelles fonctionnalités le système doit fournir et quelles contraintes fonctionnelles il doit satisfaire. Une définition compréhensible du problème doit être produite, à partir de laquelle on pourra concevoir et réaliser un logiciel. Il est utile de faire la différence entre but et besoin. Par exemple, « un système facile à utiliser » (propriété non mesurable) est un but alors que « toutes les commandes utilisateur sont sélectionnables par menu déroulant » (propriété vérifiable) est un besoin associé au but.
Le cahier des charges est « l'ensemble des propriétés ou contraintes décrites de façon précise qu'un système logiciel doit satisfaire » [GHEZZI et al., 1991]. Il doit décrire ce que le système doit faire sans préciser les moyens à mettre en œuvre pour atteindre cet objectif. Il est aussi utilisé pour valider la conception, c'est-à-dire pour déterminer si les contraintes et propriétés spécifiées sont satisfaites et si la conception conduit à une solution acceptable.

8.1.2.2 Spécification

Selon le Grand Larousse Illustré, la spécification est la « définition des caractéristiques essentielles (qualités, dimensions, ...) que doivent avoir une marchandise, une construction, un matériel, ... ». Dans le recueil des normes françaises en Génie Logiciel (AFNOR 1989), la spécification du logiciel est « l'ensemble des activités consistant à définir de manière précise, complète et cohérente, ce dont l'utilisateur a besoin ».
D'autre part, certaines approches utilisent la spécification comme une représentation du logiciel :

– minimale (sans redondance),
– suivant une approche « boîte noire » (on ne s'intéresse pas à la structure interne),
– formelle[1] (pour éviter l'ambiguïté et permettre un raisonnement mathématique).

Dans les deux derniers cas, il apparaît clairement que les spécifications expriment formellement l'analyse des besoins. L'expression formelle permet d'atteindre deux objectifs : l'élaboration d'un contrat entre le client et les concepteurs (les droits de l'utilisateur sont les obligations du réalisateur et vice versa), la mise en place d'une démarche théorique. Une spécification formelle et un programme constituent deux énoncés du même problème : la spécification est une définition du problème sous forme de propriétés, le programme est une solution sous forme d'une procédure calculatoire. Une démarche théorique a pour but de montrer que le programme calcule bien des résultats satisfaisant les propriétés énoncées par la spécification.

8.1.2.3 Conception

Cette phase, correspondant à la conception matérielle et logicielle d'un système, comprend :
– la définition de l'interface utilisateur et des services rendus par le système,
– la conception de l'architecture du système, c'est-à-dire la définition des grands modules du système et des interactions entre eux,
– la définition d'un jeu de tests basé sur le cahier des charges et sur l'architecture retenue. Ce jeu servira à valider le système ultérieurement.

La conception fournit des documents et éventuellement une ou plusieurs maquettes du système. Pour chaque module, elle doit fournir une spécification (les objectifs de réalisation) et une réalisation (moyens de réalisation). Elle peut fournir un prototype en fonction de l'une des trois démarches de prototypage suivantes :

– **exploratoire :** on construit une maquette qui a pour objectif de préciser les besoins,
– **expérimentale :** le but de la maquette est de tester la faisabilité de certaines parties qui présentent des difficultés particulières (complexité en temps, en mémoire, ...),
– **évolutive :** la maquette proposée est conçue comme une ébauche du logiciel final.

La conception s'appuie sur des méthodes de modélisation permettant de réaliser une abstraction du logiciel final (ou modèle) [Pierreval, 1990]. Les méthodes et les langages les plus connus sont Merise [Tardieu et al., 1985], Omt [Rumbaugh et al., 1991 ; Gabay, 1998], Fusion [Coleman, 1994] ou Uml [Muller, 1997 ; Lopez et al., 1998]. Rappelons qu'il est couramment admis qu'il est utile de modéliser un système à partir de trois points de vue liés mais distincts. Chacun

[1] *cf.* définition page 97

d'entre eux saisissant des aspects importants du système et tous nécessaires à une description complète :
- le *modèle objet* qui représente les aspects statiques, structurels et informationnels du système,
- le *modèle dynamique* qui représente les aspects temporels, comportementaux et de contrôle,
- le *modèle fonctionnel* qui décrit les aspects fonctions et les transformations proposées par le système.

Ces trois types de modèle décrivent le système selon des vues qui ne sont pas complètement indépendantes les unes des autres. Dans une conception orientée objet, les fonctions sont en général des opérations associées aux objets, et le modèle dynamique exprime les contraintes sur l'utilisation de ces opérations afin de définir un séquencement autorisé.

8.1.2.4 Réalisation

La réalisation implante les modèles en détaillant les diverses abstractions. Il s'agit de fournir une représentation de chaque objet et une réalisation de chaque module dans un langage de programmation, de les tester séparément, et d'effectuer l'intégration des modules en respectant le modèle de comportement pour obtenir le système final.

8.1.2.5 Phase de tests

On réalise des tests globaux pour être certain que les besoins de l'utilisateur exprimés par le cahier des charges sont satisfaits. La procédure de test se déroule généralement en exécutant le programme sur un ensemble de jeux de tests, et en comparant les résultats obtenus avec ceux qui sont attendus.

8.1.2.6 Installation, exploitation et maintenance

L'installation consiste à préparer et à placer le système dans son environnement de fonctionnement réel. L'exploitation est la phase durant laquelle le client utilise le logiciel. À partir de l'expérience ainsi acquise, le client peut découvrir des incohérences ou des erreurs dans le fonctionnement du système. Il peut proposer des modifications permettant au système d'être plus en adéquation avec les besoins des utilisateurs réels. Ces diverses modifications du système sont alors réalisées durant la maintenance.

8.1.3 Les démarches de conception

Les démarches de conception sont pragmatiques dans la mesure où elles sont fondées sur une approche par étapes munies de vérifications de cohérence informelles mais

systématiques de l'une par rapport à l'autre. Deux grandes sortes de démarches sont utilisées : les *démarches en cascade* qui consistent à enchaîner les étapes séquentiellement, et les *démarches incrémentales* qui itèrent plusieurs fois sur les étapes de spécification, de conception, de réalisation et de test. Ces dernières ont pour objectif d'établir des modèles de plus en plus détaillés permettant d'effectuer à chaque niveau d'abstraction des vérifications et de corriger les déficiences au plus tôt.

8.1.3.1 Démarches en cascade

Dans ces démarches, le cycle de vie est représenté sous forme d'une séquence d'étapes. La figure 8.1 illustre le fait qu'il y a autant d'étapes de tests qu'il y a de phases dans le cycle de développement [Sommerville, 1998]. Elle illustre également le fait que chacune des phases de test doit être pensée en même temps que la phase correspondante, indépendamment et sans hypothèse sur les autres phases de construction.

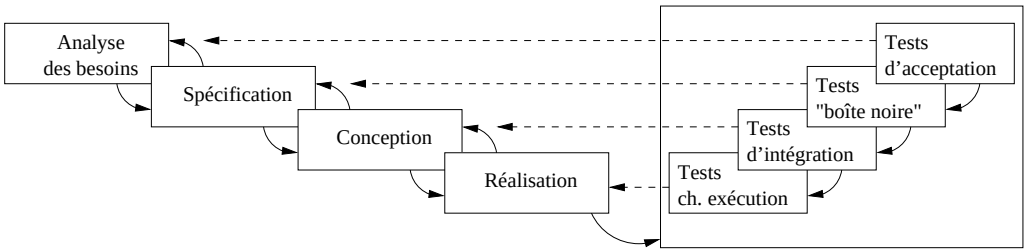


Fig. 8.1 – Démarches en cascade [Sommerville, 1998]

Ce schéma ne tient pas compte de la dimension des structures de l'entreprise. Or c'est peut-être la dimension qui oblige à passer des méthodes « artisanales » de développement aux méthodes industrielles ou d'ingénierie. Ce qui les distingue est que, dans le premier cas, celui qui formalise les besoins, qui analyse le problème, élabore des solutions, les compare, les met en œuvre, les teste, évalue leurs qualités, est un seul et même individu. Par contre, un procédé de construction est dit « d'ingénierie » si les différents acteurs intervenant dans le développement d'un logiciel peuvent être des personnes différentes. En 1981, Barry Boehm indiquait que le poids des différentes phases du cycle de vie d'un logiciel nécessite l'utilisation de ce second type d'approche [Boehm, 1981].

8.1.3.2 Démarches incrémentales

La principale critique que l'on peut faire au modèle en cascade est d'une part sa rigidité due à l'enchaînement linéaire des phases, c'est-à-dire qu'une phase doit être menée à bout pour passer à la suivante. En d'autres termes, le problème doit être

traité entièrement à chaque étape. Une autre critique importante concerne son aspect monolithique dû à un processus de développement se réalisant en une seule passe. Ainsi, les principaux problèmes du modèle en cascade sont [SOMMERVILLE, 1998] :

- La vérification du document issu de l'analyse des besoins (réalisée entre le client et le chef de projet) ne permet pas de s'assurer que l'expression des besoins est complète. Baser des développements sur ce fait sans prévoir de modification peut causer de sérieux troubles.
- Le modèle en cascade ne tient pas compte d'une éventuelle anticipation des changements,
- Ce modèle accorde beaucoup d'importance à la production de documents, c'est-à-dire aux aspects syntaxiques plutôt que sémantiques.

C'est pourquoi d'autres modèles qualifiés d'évolutifs sont proposés. Néanmoins, certains problèmes, tels que l'écriture d'un compilateur, sont bien adaptés au modèle en cascade. En effet, on connaît de manière précise la syntaxe et la sémantique des langages source et cible. C'est déjà plus délicat pour les problèmes de performance à moins qu'ils ne soient exprimés en termes d'optimisation au niveau de la spécification des besoins. Par contre les systèmes très interactifs tels que les traitements de textes sont moins adaptés car l'ensemble des besoins est potentiellement infini. Toutes les applications du domaine de l'intelligence artificielle s'accommodent aussi très mal du modèle en cascade car ce sont des problèmes où la définition des besoins n'est pas complète dès le début de l'analyse. Il est souvent nécessaire de raffiner et de mettre à jour cette définition au court des phases. On ne comprend pas toujours très bien ce que l'on veut mécaniser. [GHEZZI et al., 1991] distingue trois modèles de développement évolutif :

- **Modèle de livraison et développement incrémental :**

  Ce modèle est établi en ajoutant un processus itératif au modèle en cascade. L'idée est de choisir un sous-ensemble des besoins, de lui appliquer la démarche en cascade afin de produire une première version, de l'évaluer, puis de déterminer un cahier des charges pour la seconde version (et ainsi de suite). Ce modèle supprime la phase de maintenance en tant que telle pour l'intégrer dans le cycle de développement.

- **modèle par transformation :**

  Ce modèle est basé sur des travaux de spécification formelle [GERHART, 1991 ; WOOD, 1993 ; ABRIAL, 1996]. Il considère que le développement d'un logiciel est une séquence d'étapes qui transforment graduellement une spécification en une implantation. Ce modèle itère sur les trois étapes suivantes :
  - la définition informelle des besoins,
  - l'analyse, la spécification formelle et la structuration en objets, en fonctions et en comportements,

– la transformation de cette description formelle en une description plus détaillée et de moins en moins abstraite pour finalement aboutir au programme.

Les transformations sont exécutées par les développeurs et vérifiées formellement. À ce niveau, il s'agit d'utiliser des méthodes de transformation systématisées où le développeur porte son attention sur les choix qu'il effectue plutôt que sur le produit obtenu. Mais, pour ne pas avoir à se préoccuper de ce dernier, il est nécessaire de disposer d'outils mécanisant les transformations. Des réponses partielles sont apportées par des outils tels que Raise [The RAISE Language Group, 1992 ; The RAISE Method Group, 1995] ou Larch [Guttag et Horning, 1995 ; Tan, 1994], basés respectivement sur des méthodes de spécification orientée modèle et algébrique. D'autre part, l'approche par raffinement utilisée par la Méthode B [Abrial, 1996] peut être rapprochée du modèle par transformations. Mais, ce dernier reste encore essentiellement du domaine de la recherche [Galland, 1998 ; Bellegarde et al., 2000].

– **Modèle en spirale :**

Le modèle en spirale reprend l'idée d'itération sur les phases du cycle de vie, mais il se concentre sur l'identification et l'élimination des problèmes à haut risque en appliquant un processus soigné de conception qui évite de traiter les problèmes difficiles et triviaux uniformément. Chaque cycle comprend quatre étapes :

1. l'identification des objectifs de la portion de produit que l'on veut prendre en considération en termes de qualité. Durant cette phase, on identifie les alternatives et les contraintes ;
2. l'évaluation des alternatives et des risques potentiels, et le choix d'un modèle de développement. Cette phase peut nécessiter des simulations ou des prototypages ;
3. le développement et la vérification du niveau du produit obtenu ;
4. l'examen et la planification de l'itération suivante dans la spirale.

Ce modèle est un métamodèle dans la mesure où l'étape 2 détermine un processus de développement bien choisi en fonction de l'analyse des risques. Par exemple, si les principaux risques concernent l'interface utilisateur, une approche de prototypage peut s'avérer un modèle adapté. Si l'on se préoccupe essentiellement des risques de sûreté de fonctionnement, un développement basé sur des transformations formelles peut être le plus approprié. Il n'est pas nécessaire d'adopter pour l'ensemble du système un seul modèle à chaque cycle de la spirale. Ce modèle permet de tester la robustesse du logiciel en le soumet-

tant, à chaque itération, à des tests des besoins n'ayant pas été choisis pour cette étape.

8.2 ... à $\mathcal{MAMA}$-$\mathcal{S}$

Au cours de ces dernières décennies, la discipline de la simulation a vu son contexte fortement évoluer. L'augmentation de la complexité des systèmes modélisés et des modèles a obligé les utilisateurs de cette technique à envisager des moyens pour faciliter le développement et l'utilisation des modèles. D'autre part, le succès croissant de la simulation dans le monde industriel augmente d'autant les problèmes de compréhension entre les concepteurs et les utilisateurs, les problèmes de réutilisation des modèles, ... Ainsi, nous retrouvons, transposée au monde de la simulation, toute la problématique qui a donné naissance au Génie Logiciel et que nous avons résumé dans la section précédente. Toutefois, la simulation étant une discipline manipulant d'autres concepts que ceux du Génie Logiciel, nous allons exposer ici les différences d'approche, ainsi que nos apports dans le cadre de $\mathcal{MAMA}$-$\mathcal{S}$.

8.2.1 Phases de modélisation

La première différence entre le Génie Logiciel et la simulation est ce qui est produit. Chez le premier, il s'agit d'un logiciel, alors que la simulation produit des modèles de systèmes. Cependant, on peut facilement voir les points communs entre ces deux concepts, puisqu'un logiciel est considéré comme un système informatique en tant que tel, plutôt que comme son modèle. Ce point de vue est, selon nous, renforcé par un point de vocabulaire du Génie Logiciel : il y a souvent assimilation des termes *système* et *logiciel*. Ainsi, nous considérons que les approches méthodologiques proposées dans le cadre du Génie Logiciel peuvent être utilisées après adaptation à la simulation. Nous retrouvons ce point de vue dans les diverses méthodologies de simulation comme la « Conical Methodology » [NANCE, 1994b] ou ASCI [KELLERT et RUCH, 1998b].
Le Génie Logiciel et la simulation se distinguent aussi par les actions devant être réalisées durant les phases des cycles de vie. Ces différences sont explicitées dans les sections suivantes.

8.2.1.1 Analyse des besoins

Rappelons que, dans le cadre du Génie Logiciel, l'analyse est la phase où doivent être décrites d'une part les fonctionnalités du système et d'autre part les contraintes non fonctionnelles. Pour la simulation, les méthodologies spécialisent ces principes en considérant que le cahier des charges (produit durant l'analyse) doit contenir la

description informelle du système modélisé. Cette description comprend toutes les informations qui semblent nécessaires à la mise en œuvre du processus de simulation :

- La description des infrastructures physiques composant le système : liste des composants des lignes de production (machines, aires d'attente, moyens de transport, ...) ainsi que leurs localisations géographiques et les attributs qui sont spécifiques à ces composants (taux d'occupation, taux de panne, ...) ;
- Les informations nécessaires pour le fonctionnement du système. Par exemple, nous trouvons ici la description des gammes de produits manipulables par les systèmes, ainsi que les nomenclatures qui y sont associées ;
- Les informations nécessaires au pilotage du système : politiques d'allocation des ressources, statistiques sur les ordres de fabrications (OF) ou liste explicite et exhaustive des OF, ... On trouve aussi dans cette catégorie l'ensemble des structures décisionnelles pouvant influencer le comportement du système simulé.
- Une liste de plans d'expérience qui seront utilisés durant la phase d'expérimentation. Chaque plan d'expérience contient l'ensemble des valeurs devant être assignées aux paramètres des modèles de simulation. Notons que les plans d'expériences présents dans le cahier des charges ont deux rôles. Tout d'abord ils servent à vérifier et valider le comportement du modèle. Ensuite ils sont utilisés pour répondre à des interrogations de la part du client.
- La spécification des objectifs de la simulation et des indicateurs de performances permettant d'évaluer l'adéquation des résultats des simulations avec les objectifs.

Le cahier des charges résultant de l'analyse peut utiliser des formalismes issus de la modélisation en entreprise (suite IDEF [US Air Force, 1993a], GRAI [Doumeingts, 1984], ...) ou des systèmes d'information (Merise [Pierreval, 1990], Uml [Lopez et al., 1998 ; Muller, 1997], ...). Ces formalismes et ces langages peuvent aider à la récolte d'informations et à leur validation.

Malheureusement, le problème de la récolte des informations est complexe et ne fait pas encore l'objet d'un consensus au sein de la communauté. Face à cette hétérogénéité, nous avons décidé de ne pas aborder ce problème dans nos travaux. Nous supposerons donc que la phase d'analyse des besoins produit les informations nécessaires à la phase de spécification.

8.2.1.2 Spécification

La spécification dans le cadre de la simulation est la traduction des informations du cahier des charges en un modèle formel[2]. En effet, la phase d'analyse ne permet pas forcément d'obtenir un modèle formel[2]. Il est nécessaire de compulser les informations du cahier des charges et de construire un modèle en utilisant une méthode ou une technique basée sur une approche formelle[2]. Les formalismes et les méthodes uti-

[2] *cf.* définition page 97

lisés dépendent fortement de l'approche méthodologique ; ASCI utilise par exemple les diagrammes Entité-Association-Attribut (EAA) et les diagrammes orientés objets. Si nous prenons une méthodologie issue de la modélisation en entreprise comme CIMOSA [AMICE, 1993], la spécification correspond partiellement au *niveau de modélisation des spécifications de conception*. D'autre part, les concepts utilisés pour modéliser le système industriel à ce niveau sont eux aussi spécifiques à l'approche méthodologique utilisée.

Nous décrivons dans le chapitre 9, la phase de spécification vue selon notre approche $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$. Nous y définissons les artefacts de modélisation nécessaires pour créer un modèle de simulation, les contraintes relationnelles entre ces éléments, ainsi que les contraintes sémantiques associées. Ainsi, notre approche propose d'utiliser un métamodèle écrit selon la norme de métamodélisation proposée par UML [BOOCH et al., 1997]. Ce métamodèle décrira toutes les possibilités liées au formalisme de représentation de systèmes industriels distribués.

8.2.1.3 Conception

La phase de conception consiste à créer un modèle informatique (aussi appelé conceptuel) qui décrit de manière plus précise le modèle issu de la spécification (ou modèle abstrait). L'objectif n'est pas de choisir les infrastructures d'exécution informatique mais bien de réaliser un modèle indépendant de tout outil ou de toute plateforme. Ainsi seuls les choix d'implantation n'influençant pas le choix des infrastructures « physiques » sont réalisés. En général, cette phase est réduite à néant (« Conical Methodology »), ou confondue avec la phase de réalisation (ASCI).

Dans le cadre de $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$, nous considérons que la phase de conception est importante. En effet, nous avons décidé de ne pas introduire les systèmes multi-agents durant la phase de spécification car, comme l'illustrent les méthodologies présentées dans l'état de l'art, nous pensons que les choix de réalisation doivent être cachés le plus possible à l'utilisateur de la méthodologie, et a fortiori durant les deux premières phases du cycle de vie. Ainsi nous considérons que la conception est la phase nous permettant de construire un modèle de simulation qui est basé sur la modélisation de systèmes multi-agents. L'utilisation des SMA n'est pas en contradiction avec la définition de la conception issue du Génie Logiciel. En effet, l'approche de modélisation de SMA choisie (approche Voyelles [DEMAZEAU, 1997]) est indépendante de toute implantation informatique. Par conséquent, nous pouvons créer un modèle multi-agent respectant cette approche sans pour autant choisir une plateforme d'exécution (MAST, ...). Dans le chapitre 10, nous décrivons en détail cette phase de conception.

8.2.1.4 Réalisation

La réalisation est l'implantation du modèle issu de la conception sur une plate-forme informatique particulière. En général, les méthodologies proposent un outil spécifique pour réaliser l'implantation. Le plus souvent, des outils existants sont utilisés : ARENA® [KELTON et al., 1998], SIMPLE++® [TECNOMATIX®, 2001], QNAP [VERAN et POTIER, 1984], ... Le résultat de cette phase est un modèle qui peut être exécuté pour générer les résultats à partir des plans d'expériences définis durant l'analyse des besoins.
Nous présentons dans le chapitre 11, la phase de réalisation vue selon notre approche méthodologique. Nous y décrivons les différentes contraintes devant être respectées par les outils de simulation qui veulent être compatibles avec notre approche.

8.2.1.5 Phase de tests

La phase de tests permet de mettre en œuvre des tests globaux à partir de jeux de tests. L'objectif est de trouver un ensemble raisonnablement petit de tests qui permette d'approximer l'information que donnerait un test exhaustif. En effet, ce dernier est impossible pour la plupart des programmes. Par exemple, si un programme possède trois paramètres d'entrée dont les valeurs sont comprises dans l'intervalle [1..1000], un test exhaustif dans $\mathbb{N}^3$ exigerait d'exécuter le programme 10^9 fois. Si chaque exécution prenait une seconde, le test durerait plus de 31 années ! Cette problématique héritée du Génie Logiciel se retrouve dans les méthodologies pour la simulation. En effet, les plans d'expérience (équivalents des jeux de tests du Génie Logiciel) ne permettent pas de vérifier exhaustivement l'ensemble des comportements possibles d'un système industriel simulé.
Nous tenons à différencier la *vérification* de la *validation* d'un modèle de simulation. La vérification est le processus mis en place durant la phase de tests qui permet de vérifier que le modèle de simulation produit bien les résultats attendus. La validation est réalisée par l'utilisateur final du modèle. Elle correspond à une validation des résultats proposés par le modèle sur des données réelles. Par conséquent, la validation est du ressort de la phase d'exploitation.

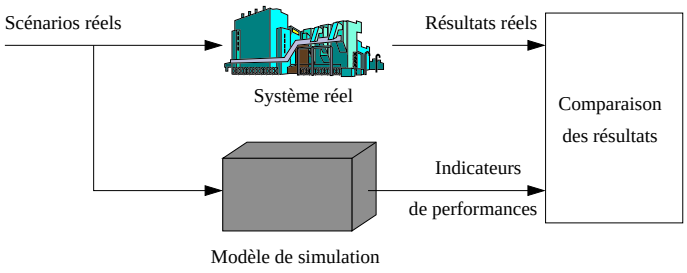


FIG. 8.2 – Tests d'acceptation d'un modèle de simulation

La vérification du modèle doit être réalisée selon deux niveaux :

– **Tests de construction :**

L'objectif de ces tests est de vérifier que le modèle de simulation respecte les règles de construction syntaxiques et sémantiques données par le langage de modélisation utilisé. Dans le cadre de $\mathcal{MAMA}$-$\mathcal{S}$, ces tests en « boîte blanche » seront réalisés par l'application des règles de bonne construction imposées par le métamodèle UML (phase de spécification), par une méthodologie de modélisation de systèmes multi-agents (phase de conception) et par les contraintes syntaxiques et sémantiques des outils informatiques utilisés (phase de réalisation). Nous verrons dans les chapitres suivants comment ces tests sont mis en œuvre au sein de notre approche méthodologique.

– **Tests d'acceptation :**

Les tests d'acceptation se réalisent en « boîte noire ». En effet, leur objectif est de vérifier que les résultats proposés par le modèle sont compatibles et cohérents avec ceux proposés par le système réel (*cf.* figure 8.2). De ce fait, ce qui nous intéresse n'est pas le modèle en tant que tel, mais plutôt son comportement face à un ensemble de données d'entrées. Ces dernières sont représentées par les plans d'expérience dégagés durant la phase d'analyse des besoins. Il existe deux cas de figure :

1. Le système réel n'existe pas :
   Dans ce cas, il faut calculer analytiquement (statistiques, théorie des files d'attente, ...) un résultat de référence. Pour obtenir un modèle analysable, ce dernier est simplifié. Cette simplification peut prendre plusieurs formes : retrait des aléas du modèle, gestion d'une seule entité par modèle, étude du régime permanent dans le cas où il y a un grand nombre d'entités, ... Une fois le résultat analytique mis en évidence, il peut être utilisé comme s'il s'agissait d'un résultat produit par un système existant.
2. Le système réel existe :
   Les scénarios (ou plans d'expérience) utilisés sont issus des données réelles traitées par le système industriel. Par exemple, nous pouvons y trouver l'ensemble des lots de fabrication traités par un atelier durant une période donnée. Les indicateurs de performances correspondant aux résultats de la simulation sont comparés aux résultats réels. Malheureusement, cette comparaison aboutit rarement à la conclusion que le résultat réel r est strictement identique au résultat simulé s. La raison majeure de cette différence est la propriété stochastique des modèles de simulation qui ne permettent pas d'avoir des résultats invariables entre deux exécutions ayant les mêmes données d'entrée. Pour permettre la comparaison entre r et s, il faut mettre en place des intervalles de confiance. Ainsi, s'il existe

un intervalle de confiance de 5 % pour r, alors il faut trouver s_{min} et s_{max} tel que la probabilité que r se trouve entre s_{min} et s_{max} soit de 95 % :

$$\exists s_{min}, s_{max} / s_{min} \leq s \leq s_{max} \wedge \mathrm{P}(s_{min} \leq r \leq s_{max}) \geq 0.95$$

Le calcul de cet intervalle permet d'affirmer que le résultat simulé s est identique au résultat réel r avec une marge d'erreur de 5 %. Nous n'aborderons pas ici la construction d'un intervalle de confiance. Cette problématique a été largement abordée par la communauté scientifique [Futschik et Pflug, 1995 ; Goldsman et Schruben, 1990 ; Kilmer et al., 1999 ; Barchard et Hakstian, 1997 ; Borkowf, 2000].

8.2.1.6 Installation, exploitation et maintenance

Les phases d'installation, d'exploitation et de maintenance ont strictement les mêmes sémantiques que les étapes existantes dans les approches « classiques » du Génie Logiciel.
Dans le cadre de $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$, nous ne proposons pas de modifier ces phases. De nombreux travaux se sont déjà intéressés aux problématiques liées à l'utilisation d'un modèle de simulation (analyse des résultats, ...).

8.2.2 Approche itérative

Du point de vue de la démarche de modélisation, la plupart des méthodologies reprennent une approche itérative. Nous rappelons que cette approche consiste à décomposer le problème (ici, la modélisation d'un système industriel) en sous-problèmes, et d'appliquer une démarche en cascade ou une démarche itérative sur ces sous-problèmes.
Ainsi, la figure 8.3 illustre le cycle de vie des modèles durant une seule itération (formalisme proche de SADT [Lissandre, 1990]). Nous retrouvons les phases décrites ci-dessus. La figure 8.4 illustre l'aspect itératif du cycle de vie d'un modèle de simulation. Notre approche méthodologique ne redéfinit pas un nouveau cycle de vie mais se contente d'adapter certaines étapes pour permettre le support des systèmes industriels distribués (*cf.* descriptions des étapes ci-dessus). Nous faisons apparaître nos apports au niveau de la spécification, de la conception et de la réalisation.
Dans la spécification, nous introduisons la définition d'un métamodèle pour les systèmes de production distribués. Ce métamodèle est défini selon la norme de métamodélisation proposée par Uml [Booch et al., 1997]. D'autre part, nous proposons l'intégration d'un système multi-agents dont le rôle sera de vérifier la cohérence syntaxique et sémantique des modèles distribués (brique « Vérification

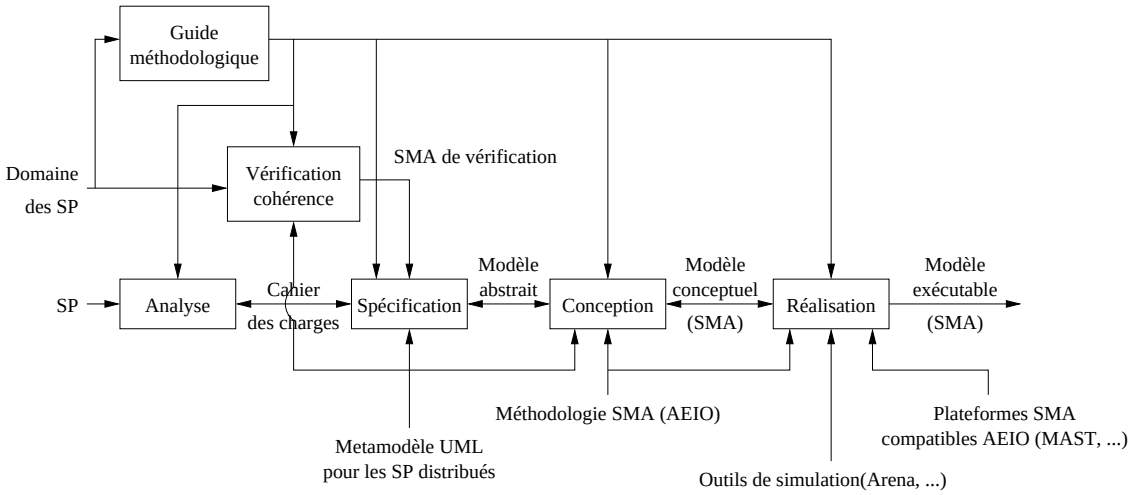


FIG. 8.3 – Cycle de vie – Démarche en cascade durant une itération

cohérence »). Notons dès maintenant que ce système multi-agents est totalement différent du système qui servira de support au processus de simulation.

Selon notre point de vue, la conception nous permet de créer un modèle multi-agents représentant le modèle abstrait issu de la spécification. C'est ainsi que nous introduisons l'approche « Voyelles » [DEMAZEAU, 1997] présentée dans l'état de l'art de ce mémoire. D'autre part, nous proposons un ensemble de règles permettant de traduire le modèle abstrait en une société d'agents.

Durant la réalisation, nous introduisons les outils de simulation existants (ARENA®, SIMPLE++®, ...). Nous rappelons que nous ne désirons pas créer de nouveaux outils de simulation. Notre objectif est de proposer une architecture logicielle adaptant les outils existants à la simulation de systèmes industriels distribués. Pour ce faire, nous proposons un ensemble de contraintes que doivent respecter les outils de simulation pour pouvoir s'intégrer dans notre environnement de simulation. Il s'agit essentiellement de pouvoir utiliser une interface de communication commune à l'ensemble de l'environnement. Cette dernière fait aussi partie intégrante de la société d'agents qui sera le support informatique pour la distribution. Nous détaillons notre approche de réalisation dans le chapitre 11.

En plus des phases classiques d'une approche méthodologique (de l'analyse à l'exploitation), nous faisons apparaître sur la figure 8.3 deux autres « briques » : le guide méthodologique et la vérification de cohérence.

Le guide méthodologique est une étape durant laquelle sont spécifiés les principes de la méthodologie (cycle de vie, artefacts de modélisation, méthodes, ...) utilisée pour construire un modèle de simulation. Il s'agit d'un document décrivant à la fois les spécifications de $\mathcal{MAMA}$-$\mathcal{S}$, mais aussi un guide d'utilisation de celles-ci. La vérification de cohérence est une phase durant laquelle est construite une architecture multi-

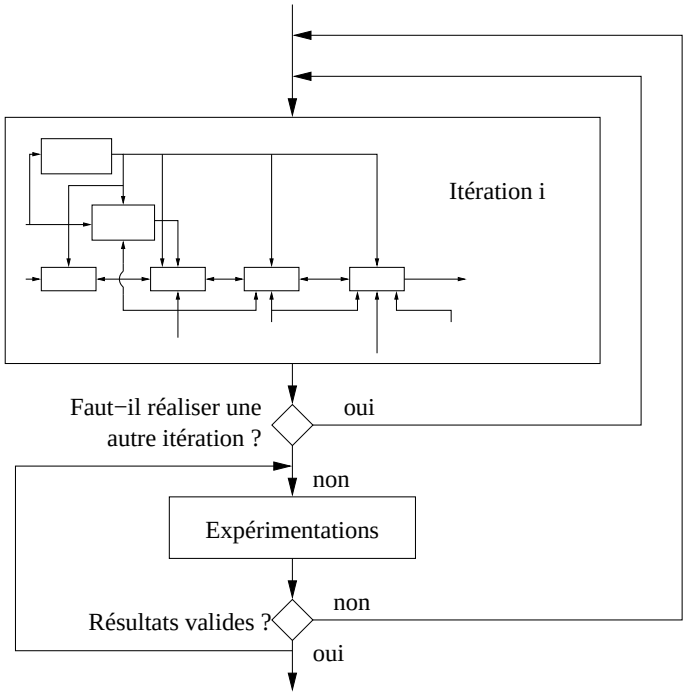


Fig. 8.4 – Cycle de vie – Démarche itérative

agents dont le rôle est de vérifier la cohérence des modèles abstraits de simulation. Cette phase est entièrement décrire dans la section 9.4. Ces deux phases sont réalisées par les concepteurs de l'approche méthodologique $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$.

8.3 Conclusion

L'élaboration d'une approche méthodologique passe obligatoirement par l'étude du cycle de vies des modèles. À partir de l'expérience acquise par le Génie Logiciel et par les méthodologies de simulation existantes, nous avons extrait un cycle de vie adapté à notre problématique. Notre proposition s'appuie sur une adaptation des approches existantes et non pas sur le développement d'un cycle de vie totalement nouveau et différent.

Ainsi, nous avons retenu l'utilisation d'approches de modélisation de type itératif ou en spirale. Chaque itération étant composée de quatre phases majeures : l'analyse des besoins, la spécification, la conception et la réalisation. Le support de la modélisation de systèmes industriels distribués nécessite la modification de trois de ces phases : la spécification, la conception et la réalisation.

L'analyse des besoins reste identique par ses objectifs à ce qui existe déjà dans d'autres approches méthodologiques comme la « Conical Methodology » ou ASCI. Elle doit permettre de rédiger un cahier des charges qui décrit informellement le système à modéliser. Nous y retrouvons les concepts de récolte de données comme les

gammes ou les nomenclatures, la définition des structures organisationnelles correspondant au système de pilotage de l'entreprise ou l'expression de plans d'expérience permettant de vérifier la validité des modèles produits. Notons au passage que l'utilisation au jour le jour du modèle de simulation, c'est-à-dire en lui fournissant des plans d'expérience adaptés aux questions des utilisateurs, est du ressort de la seule phase d'expérimentation.

La spécification est la première phase à laquelle nous apportons des modifications. En effet, nous estimons qu'un modèle abstrait du système à simuler doit être réalisé durant cette étape. Pour permettre la simulation de systèmes industriels distribués, nous introduisons un métamodèle UML définissant les artefacts de modélisation accessibles à l'utilisateur de $\mathcal{M}_{\mathcal{A}}\mathcal{M}\mathcal{A}$-$\mathcal{S}$. Le principe de base de cette phase est d'aider à la spécification d'un modèle de simulation. Ainsi, les utilisateurs doivent posséder des compétences minimales dans le domaine de la simulation. Toutefois, le formalisme que nous proposons nous semble facilement accessible à des personnes ayant des compétences très faibles en simulation.

Selon notre approche méthodologique, la conception nous permet de définir un modèle multi-agents correspondant au modèle du système issu de la spécification. Nous posons toutefois comme hypothèse que les choix des outils de simulation et des plateformes multi-agents seront réalisés durant la phase suivante du cycle de vie. Ici, nous proposons un ensemble de règles permettant de traduire un modèle abstrait d'un système de production en une société d'agents capable de réaliser le processus de simulation.

La réalisation est la dernière phase dont nous avons modifié l'approche. Durant cette étape, les choix de réalisation sont faits. Parmi ces choix, nous incluons ceux des outils de simulation (ARENA®, SIMPLE++®, ...) et des plateformes multi-agents (MAST, ...). Dans les spécifications de cette phase, nous décrivons les contraintes devant être respectées par l'un et l'autre pour pouvoir s'intégrer dans notre environnement de simulation.

Enfin, la phase d'expérimentation permet de déployer le modèle de simulation chez le client, de l'utiliser et éventuellement de mettre en œuvre un processus de maintenance. Ces phases n'ont pas fait l'objet de notre étude.

Certains peuvent considérer que la mise en œuvre de notre approche méthodologique s'avère trop lourde. En effet, nous proposons un canevas formel et complexe de modélisation. Il est plus difficile et plus long à mettre en place que des approches « intuitives » de modélisation (par exemple avec uniquement ARENA®). Toutefois, nous estimons que ce que nous perdons en modélisation lourde est gagné en matière de compréhension des structures composant le système industriel, et de correction des modèles. De plus, nous pensons que la mise en œuvre de $\mathcal{M}_{\mathcal{A}}\mathcal{M}\mathcal{A}$-$\mathcal{S}$ peut être de plus en plus efficace si les principes de modularité et de réutilisation sont respectés.

D'autre part, $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$ est une méthodologie destinée à la simulation de systèmes industriels distribués. De ce fait, elle tend à répondre aux problèmes de modélisation coopérative (problèmes que nous n'abordons pas dans le cadre de ce mémoire).
Dans les chapitres suivants, nous décrivons en détail les deux première phases que nous nous proposons d'adapter à la simulation de systèmes industriels distribués, ainsi que les contraintes de réalisation et nos propositions d'implantation d'un environnement de modélisation et de simulation.

Chapitre 9

Spécification

La phase de spécification est cruciale dans notre approche méthodologique. En effet, il s'agit du moment où le premier modèle exprimé formellement[1] doit être produit. Dans ce chapitre, nous présentons les bases méthodologiques et les principes sous-jacents à la spécification d'un modèle abstrait. Dans un premier temps, nous rappelons les concepts de base de la simulation. Ceux-ci nous permettront de développer nos explications sur les artefacts de modélisation proposés dans $\mathcal{M}_{\mathcal{A}}\mathcal{M}\mathcal{A}$-$\mathcal{S}$. Nous décrivons aussi le métamodèle Uml qui nous permet de mettre en œuvre les éléments de modélisation. Enfin, avant de conclure ce chapitre, nous présentons les principes de la vérification d'un modèle abstrait de simulation.

9.1 Principes de la Simulation à Événements Discrets (SED)

La simulation possède un certain nombre de concepts sous-jacents. Ils incluent les systèmes, les modèles, les événements, ainsi que la définition de modèles de simulation à événements discrets. De nombreux travaux ont contribué à mettre en avant ces différents concepts [Banks et al., 1996 ; Law et Kelton, 1991]. Nous utilisons plus spécifiquement les travaux de John Carson en ne retenant que les définitions propres à la simulation à événements discrets [Carson, 1993] :

– **Système, modèle et événement :**

Un *modèle* est une représentation d'un *système* réel. Un *événement* est une action qui change l'état du système (l'arrivée d'un ordre de fabrication, la fin d'une production, ...). Il existe deux classes d'événements : les événements *internes* (ou endogènes) et *externes* (ou exogènes). Un événement endogène correspond à un événement généré au sein de la simulation (départ d'une production, ...), alors qu'un événement exogène a pour origine l'environnement

[1] *cf.* définition 8.1 page 97

extérieur à la simulation (arrivée d'une commande provenant d'un client, ...).

– **Variables d'état du système :**

Les *variables d'état* d'un système sont les informations nécessaires à la définition du comportement du système à un niveau de détail suffisant pour répondre aux objectifs de simulation. Ces informations représentent l'état du système à un instant précis. Ainsi, les valeurs des variables d'état restent inchangées durant l'intervalle de temps se trouvant entre deux événements.

– **Entités et attributs :**

Une *entité* représente un objet qui doit être explicitement défini. Selon [BANKS et al., 1996], elle peut être qualifiée de *dynamique* si elle se déplace au sein du système, et de *statique* si elle est exclusivement utilisée par d'autres entités. Dans la suite de ce mémoire, nous abuserons du terme « entité » pour représenter une entité dynamique. Un exemple d'entité est un lot de fabrication transitant par les différentes machines d'un atelier (entité appartenant au « flux physique »). Un second exemple est un ordre de fabrication passant d'un atelier vers un autre (entité appartenant au « flux décisionnel »). Une entité peut posséder des *attributs* qui correspondent à des valeurs locales à celle-ci. Les attributs permettent de décrire ce qui est transporté par une entité (identificateur du lot de fabrication, couleur des produits, ...).

– **Ressources :**

Une *ressource* est un objet qui fournit des services pour les entités. Une ressource peut fournir un service à plusieurs entités (cas d'un serveur parallèle). Une entité peut demander un service à une ou plusieurs instances d'une ressource. Dans la plupart des cas, si ces dernières ne sont pas disponibles, l'entité est placée dans une file d'attente. Si, par contre, elle est autorisée à utiliser la ressource, l'entité se l'alloue et, lorsqu'elle n'en a plus besoin, elle la libère. Une ressource peut posséder divers états qui incluent au minimum l'état de veille (« iddle ») et l'état d'activité (« busy »). Ces deux états peuvent être raffinés, ou d'autres états peuvent être insérés (« failed », « blocked », « starved », ...). Généralement, les seules ressources représentées dans les modèles de simulation sont les *ressources critiques* c'est-à-dire les ressources pouvant interrompre le flux d'entités.

– **Files d'attente :**

Les entités sont gérées en leur allouant une ressource, en les attachant à des notifications d'événements et en les plaçant dans des listes ordonnées. Ces dernières représentent les *files d'attente*. Elles peuvent utiliser une politique de gestion particulière (« First-In-First-Out » ou FIFO, ...). Cette politique peut être fonction des valeurs des attributs transportés par l'entité.

– **Activités et délais :**

Une *activité* est une durée correspondant au temps mis par une activité pour réaliser sa tâche. Ainsi, lorsqu'une activité commence, il est possible de planifier le moment où elle se terminera. La durée peut être une constante, une valeur aléatoire obtenue à partir d'une distribution statistique, le résultat de l'évaluation d'une équation, une donnée provenant d'un fichier externe ou du résultat d'un calcul à partir des variables d'état. Un *délai* est une durée obtenue à partir de conditions propres au système. Par exemple, lorsqu'une entité se trouve dans une file d'attente, on ne peut pas savoir a priori quand elle en sortira. Il est donc impossible de planifier le moment où un délai arrivera à terme. Les simulations à événements discrets contiennent des activités qui permettent de faire avancer leurs horloges. Le début et la fin d'une activité ou d'un délai sont considérés comme des événements.

– **Modèle de simulation à événements discrets :**
Un modèle de simulation à événements discrets est défini comme un modèle dont les états changent seulement à des moments précis dans le temps. Ces moments sont caractérisés par les événements. Les événements sont générés en fonction des activités et des délais. Les entités sont en concurrence pour l'utilisation des ressources composant le modèle. Ainsi, les modèles de simulation à événements discrets sont utilisés au sein de mécanismes permettant de faire avancer le temps simulé, de modifier les variables d'état à chaque événement, et de capturer et libérer des ressources pour les entités dynamiques.

En résumé, un modèle est composé d'entités qui parcourent des processus composés d'activités, en utilisant momentanément des ressources. Les files d'attente servent à stocker temporairement les entités en attendant la libération de ressources.
Ce rappel, concernant les concepts et le vocabulaire attachés à la simulation à événements discrets, nous offre à la fois un cadre strict, mais aussi une transition parfaite pour introduire l'ensemble des artefacts de modélisation nécessaire pour construire un modèle de simulation. Dans la section suivante, nous décrivons l'ensemble de ces éléments dans le cadre de la spécification d'un modèle abstrait de simulation.

9.2 Artefacts de modélisation

Comme nous l'avons indiqué dans le chapitre 8, la spécification est une phase du cycle de vie de $\mathcal{MAMA}$-$\mathcal{S}$ qui consiste à produire un modèle abstrait du système industriel distribué à simuler. Cette construction doit être réalisée à partir des informations récoltées et exposées dans le cahier des charges.
Nous rappelons que notre approche de modélisation considère que les systèmes de production distribués peuvent être décomposés selon l'approche systémique de Jean-

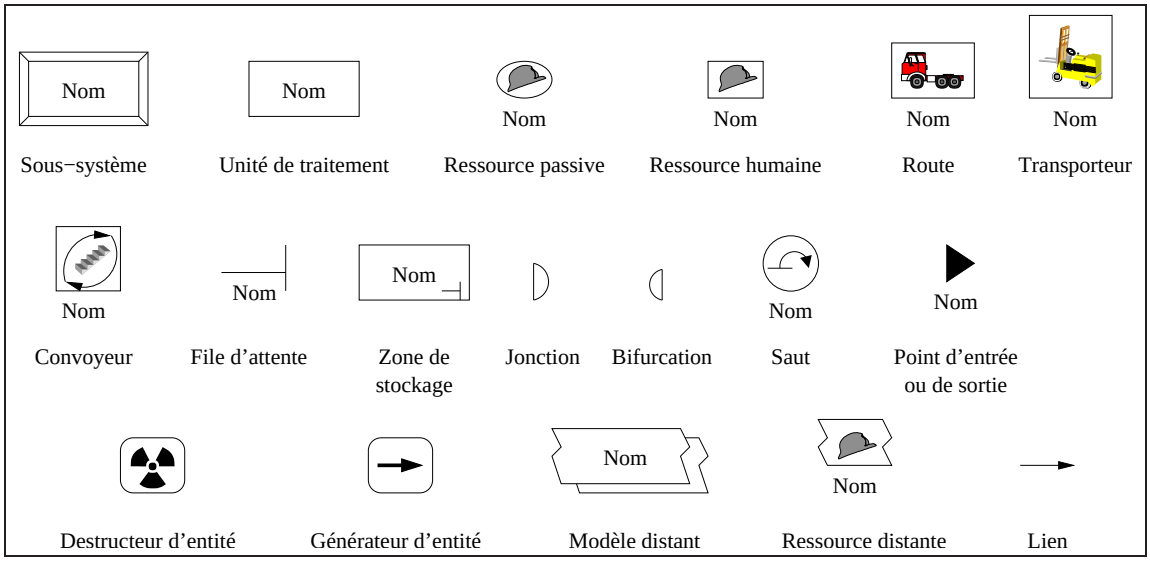


FIG. 9.1 – Un formalisme pour le sous-système physique

Louis LE MOIGNE [LE MOIGNE, 1992] : un sous-système opérationnel, un sous-système informationnel et un sous-système décisionnel. Ainsi, nous proposons un ensemble de concepts de modélisation pour chacun de ces sous-systèmes, ainsi que des concepts communs. Nous considérons donc que le modèle abstrait est composé de trois sous-modèles fortement inter-dépendants.

9.2.1 Sous-système opérationnel

Après une étude des différents éléments de modélisation proposés par les méthodologies de modélisation en entreprise, de simulation, et par les différents outils de simulation existants, nous avons dégagé un ensemble d'artefacts de modélisation propres au sous-système opérationnel [GALLAND et al., 2000b ; GALLAND et al., 2001a ; ERKOLLAR et FELFERNIG, 1999 ; CARSON, 1993]. D'autre part, ces concepts ont été comparés et rapprochés de ceux qui sont couramment utilisés dans le cadre de notre expérience d'enseignement [JULLIEN, 1991 ; GALLAND et al., 2000c]. Nous avons regroupé l'ensemble de ces éléments de modélisation dans différentes catégories. Chacune d'entre elle fait l'objet d'une explication dans les sous-sections suivantes. D'autre part, la figure 9.1 illustre le formalisme retenu pour chacun des éléments de modélisation.

9.2.1.1 Décomposition

Le *modèle* est l'ensemble des éléments composant le sous-système physique. Il peut être décomposé en *sous-modèles*. Ces derniers sont des modèles dont les entrées et les sorties peuvent être utilisées par un modèle englobant.

La décomposition est rendue nécessaire par l'accroissement de la complexité des systèmes à modéliser. Selon les théories de l'organisation scientifique du travail qui ont été reprise en Génie Logiciel, elle permet d'améliorer la qualité de la modélisation en termes de complexité, de maintenabilité, et de compréhension.
Dans la suite de ce document, nous définissons le modèle (ou sous-modèle) comme un ensemble d'artefact de modélisation qui possède les propriétés suivantes :
– son nom unique dans son contexte de nommage (le modèle contenant ce modèle),
– un ensemble d'entrées et de sorties identifiées par des noms uniques,
– un ensemble d'attributs décrivant des propriétés physiques du modèle (description de la localisation géographique, ...),
– un ensemble d'artefacts de modélisation contenus par le modèle.
Nous parlons de *modèle de niveau n* pour désigner un modèle correspondant à la $n^{\text{ème}}$ décomposition du modèle du sous-système opérationnel. Enfin, nous nommons *sous-système* un artefact de modélisation appartenant à un modèle et représentant un des sous-modèle le composant. Ces derniers ne font apparaître que le nom, les attributs et les entrées/sorties d'un sous-modèle.

9.2.1.2 Ressources critiques

Nous rappelons que dans le cadre de la simulation, nous nous intéressons uniquement aux ressources critiques, c'est-à-dire aux ressources pouvant interrompre le flux des entités physiques. Nous les regroupons dans deux catégories :

– **Les ressources critiques actives :** elles sont nécessaires pour la réalisation d'une activité. Nous considérons qu'une ressource active possède les propriétés suivantes :
  – un nom unique,
  – un ensemble de liens (*cf.* définition des liens ci-dessous) correspondant aux relations dynamiques existantes entre la ressource et les artefacts qui l'entourent.
  – un ensemble contenant les états possibles d'une ressource active (occupée, libre, en panne, ...),
  – un ensemble d'attributs décrivant les propriétés physiques de l'unité de traitement (modèle d'une machine, ...).

  De plus, nous raffinons la notion de ressource active en quatre catégories :

  1. Les *unités de traitement* sont des ressources actives représentant une activité de transformation, d'agrégation ou de production sur les entités transitant par elles. Ainsi, une unité de traitement est aussi définie par son activité (période temporelle).
  2. Les *ressources humaines* ont pour objectif de modéliser les opérateurs humains ayant une influence directe sur le flux d'entités. Ces ressources per-

mettent d'apporter de la valeur ajoutée aux entités transitant par elles. Il s'agit d'une typologie particulière de ressources pour lesquelles l'approche SMA permet de lever un certain nombre de contraintes, notamment dans le sous-système décisionnel.

3. Les *zones de stockage* sont des ressources permettant de modéliser une zone de stockage. Il s'agit d'un artefact héritant son comportement des unités de traitement et des files d'attente.

4. Les *ressources de transport* sont des ressources dont l'objectif n'est pas de transformer, ou d'agréger des entités. Elles permettent de modéliser les déplacements physiques de celles-ci. Nous considérons trois types de ressources de transport :

   (a) les *routes* correspondent à des ressources où seules les contraintes temporelles sont prises en compte (délais de transport) : transports routiers ou ferroviaire, ... Ainsi, ces ressources ne s'intéressent pas aux moyens mis en œuvre pour réaliser le transport, mais seulement aux délais mis pour déplacer une entité d'un point vers un autre ;

   (b) les *transporteurs* utilisent obligatoirement une ressource passive (ou unité de transport) pour réaliser le transport des entités. Les unités sont indépendantes les unes des autres et se déplacent à la demande (taxi, chariot élévateur, ...). Ainsi, grâce à cet élement de modélisation, il est possible de modéliser les moyens mis en œuvre pour réaliser le transport ;

   (c) tout comme les ressources précédentes, les *convoyeurs* utilisent des unités de transport. Mais, elles se distinguent par leurs mouvements continus et par la dépendance qui existe entre elles (tapis roulant, escalier mécanique, ...).

– **Les ressources critiques passives :** elles sont essentielles à la réalisation d'une activité, mais n'apportent pas de valeur ajoutée dans le processus simulé (palette, place d'une aire de stockage, ...). Les ressources passives sont définies par :
  – un nom unique,
  – un ensemble contenant les états possibles d'une ressource active (occupée, libre, en panne, ...),
  – un ensemble d'attributs décrivant les propriétés physiques de la ressource (taille spatiale, ...).

  Les ressources passives sont des ressources généralement utilisées pour la réalisation des activités des ressources actives.

9.2.1.3 File d'attente

Les files d'attente sont des artefacts de modélisation permettant de représenter une liste d'entités qui attendent un événement particulier (libération d'une ressource active, terminaison d'une activité, ...). La politique de gestion de la file peut être du ressort du sous-système décisionnel.

9.2.1.4 Structuration des flux physiques

Afin de pouvoir créer un modèle de représentation de l'infrastructure physique du système industriel, il faut avoir accès à des éléments permettant de structurer le flux. Nous proposons huit artefacts regroupés en six catégories :

- Les *liens* sont des liaisons représentant les trajets possibles d'entités au sein du flux physique. Notons que la détermination d'un trajet durant la simulation est réalisée en comparant la description physique du flux (composée de liens et de ressources) et la description des gammes et des nomenclatures appartenant au sous-système informationnel. Les liens ne sont qu'une représentation sémantique des chemins de transport. Ils ne transportent aucune autre information. En outre, le transport des entités par un lien est instantané. Nous parlerons alors de transfert et non plus de transport.
- Les *jonctions* sont des éléments de modélisation permettant de représenter la fusion d'un ensemble de chemins. Tout comme les liens, ces éléments réalisent leur traitement instantanément. Toutefois, ils peuvent être associés à une loi statistique ou à un centre de décision qui décidera comment fusionner les voies entrantes.
- Les *bifurcations* sont similaires aux jonctions à la différence qu'elles prennent une seule entrée et plusieurs sorties. Tout comme pour les jonctions, les bifurcations peuvent être associées à une loi statistique ou à un centre de décision pour déterminer quelle sortie choisir.
- Les *sauts* sont des moyens syntaxiques pour ne pas représenter l'ensemble des liens. Ainsi lorsqu'une entité arrive dans un saut, il est automatiquement transféré dans un autre saut spécifié statiquement dans ce modèle. Le modèle s'en trouve graphiquement allégé tout en gardant la même sémantique.
- Les *points d'entrée* sont des artefacts représentant les flux physiques venant de sources extérieures au modèle. Cet élément est utilisé pour représenter l'arrivée des entités dans le système (par exemple, les matières premières ou les produits venant d'entreprises sous-traitantes), ainsi que les points où des modèles de simulation distants peuvent envoyer leurs produits. Dans le premier cas, nous proposons un élément spécialisé nommé *générateur d'entités*.
- Les *points de sortie* sont à l'émission ce que sont les points d'entrée pour la réception. Tout comme pour les points d'entrée, il en existe deux catégories. Les *destructeurs d'entités* font partie de la première et ont pour rôle de détruire toute

entité leur parvenant. La seconde catégorie est utilisée pour modéliser les voies possibles du flux vers l'extérieur du modèle (autres modèles, ...).

9.2.1.5 Distribution

Cette dernière catégorie contient l'ensemble des éléments de modélisation spécifiques au support des systèmes industriels distribués. Du point de vue du sous-système opérationnel, nous considérons deux artefacts : le *modèle distant* et la *ressource distante*. Le modèle distant peut être intégré comme un sous-modèle classique. La différence réside dans le fait que la spécification de ce modèle appartient à un modèle inconnu dans ses détails et défini, par exemple, par un autre membre du groupement. Toutefois, nous proposons une approche de vérification permettant de mettre en valeur toute incohérence d'utilisation d'un modèle par rapport à sa véritable définition. Tout comme pour les modèles, il est possible d'utiliser des ressources dont les définitions réelles et la gestion sont à la charge d'un modèle distant. Un exemple de ce type de ressource est le partage au sein d'un groupement d'un opérateur très spécialisé. Celui-ci peut être utilisé par tous les membres mais n'appartient « administrativement » qu'à un seul d'entre eux.

9.2.1.6 Exemple

Pour illustrer nos propos sur la modélisation du sous-système opérationnel, nous proposons de créer un modèle correspondant à un système distribué simple. Prenons un groupement de trois entreprises E_1, E_2 et E_3. L'objectif de ce groupement est de produire des détecteurs de mouvements. Nous proposons simplement cet exemple afin de faciliter la compréhension des concepts utilisés et pour illustrer l'utilisation de notre approche méthodologique.
La société E_1 produit des capteurs dans son atelier A qui sont ensuite envoyés à la seconde entreprise. Celle-ci assemble dans son atelier B les capteurs avec les boitiers qui sont fabriqués localement. Ensuite, E_2 fait parvenir les détecteurs au troisième membre du groupement qui doit stocker les produits finis.
Le transport entre ces trois entreprises est exclusivement réalisé par les services de transport de E_2. Ces services sont composés d'un parc de 2 fourgonnettes capables de transporter 5000 capteurs ou 3000 détecteurs. D'autre part, l'atelier A utilise une ressource critique qui correspond à un opérateur, et l'atelier B utilise deux ressources : la première est une ressource critique qui permet d'assembler les caméras, et la seconde est une ressource passive représentant un opérateur supervisant les tâches de l'atelier B. Ajoutons que les accords entre les membres du groupement spécifient que E_1 ne sait pas comment sont transportés les capteurs, et que E_2 n'oblige pas E_3 à utiliser ses services de transport. Ces considérations nous permettent de placer les artefacts de modélisation dans les modèles de E_2 et de E_3. La figure 9.2 illustre les

trois modèles graphiques. De plus, chacun des artefacts possède des propriétés qui lui sont propres :

– **Modèle E_1 :** l'artefact « `Chez E2` » définit l'adresse de destination des capteurs. Ainsi, son attribut `remote_address` est initialisé à la valeur « `E2.reception_capteur` » qui correspond au nom de partage de l'entrée définie dans le modèle de E_2.

– **Modèle E_2 :** le nom partagé (`shared_name`) du modèle est « `E2` », et celui du point d'entrée possède la valeur « `reception_capteur` ». D'autre part, les files d'attente sont des structures physiques qui ne peuvent changer de mode de gestion. Étant donné qu'il s'agit d'une contrainte matérielle, nous préférons placer la politique de gestion de la file d'attente dans le modèle opérationnel plutôt que dans le modèle décisionnel. La politique utilisée par les deux files est « First-In-First-Out ». Enfin, un transporteur est défini dans ce modèle. Étant donné les contrats existants entre les membres du groupement, cette ressource est partagée sous le nom de « `service_transport` ». Notons que ce dernier nom peut être différent du nom local de l'artefact (comme cela est illustré dans la figure 9.2).

– **Modèle E_3 :** les administrateurs de l'entreprise E_3 ont décidés de profiter de l'offre d'utilisation des moyens de transport de E_2. Ils utilisent la ressource distante « `E2.service_transport` » (valeur attribuée à l'attribut `remote_address` de l'artefact « `Camions E2` »).

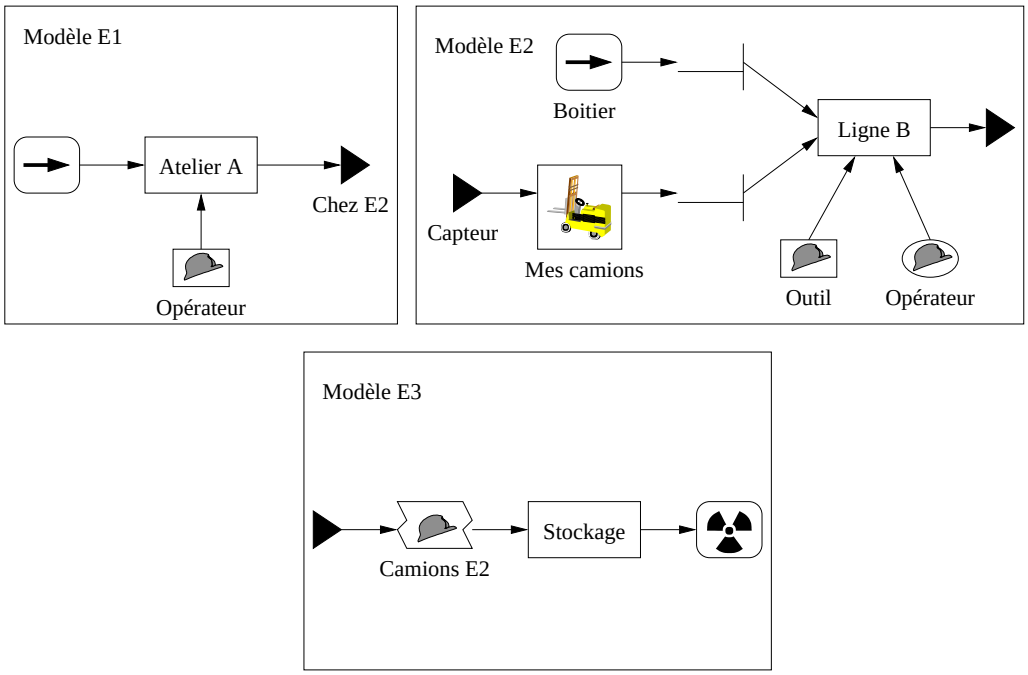


Fig. 9.2 – Exemple de modèles du sous-système opérationnel

9.2.2 Sous-système informationnel

Le sous-système informationnel contient l'ensemble des définitions et des informations nécessaires à la simulation du système, et plus particulièrement, nécessaires aux deux autres sous-systèmes. Les différents concepts composant le modèle informationnel sont issus des travaux réalisés dans les méthodologies de simulation et de modélisation en entreprise. Plus particulièrement, nous nous inspirons d'ASCI [KELLERT et RUCH, 1998b] quant à l'inclusion des nomenclatures et des gammes. Les différentes informations nécessaires à la constitution d'un modèle du sous-système informationnel sont décrites dans les sous-systèmes suivants.

9.2.2.1 Modèle de nomenclature

Les nomenclatures représentent les descriptions hiérarchisées des compositions des produits manipulés par le système. Ainsi, chaque produit peut être décliné en un agrégat de sous-produits, ces derniers pouvant à leur tour être décomposés. Chaque produit est décrit par un ensemble d'attributs en fonction du domaine d'application de la simulation (description textuelle, caractéristiques techniques, ...). Chaque attribut doit posséder une valeur durant la simulation. Pour cela, le modèle informationnel au niveau spécification permet de donner à ces attributs soit une valeur statique, soit un pointeur vers une source d'information extérieure (base de données, ...). Cette dernière possibilité nécessite toutefois d'être précisée durant les phases de conception et de réalisation.
Le modèle de nomenclatures peut être décrit graphiquement par un arbre où les noeuds sont les produits et les arcs sont les relations de composition. La figure 9.3 illustre les trois types d'artefacts graphiques utilisables : un produit, une composition inclusive (« et ») et une composition exclusive (« ou »). De plus, une composition possède un attribut propre qui représente la quantité de composants nécessaire à la fabrication d'un composé.

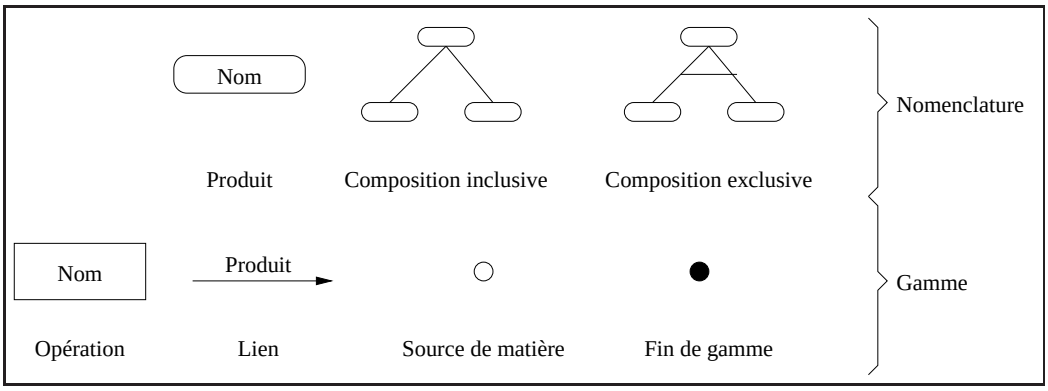


FIG. 9.3 – Un formalisme pour le sous-système physique

9.2.2.2 Modèle de gamme opératoire

Les gammes opératoires représentent la spécification des différentes activités à mettre en œuvre pour obtenir un produit. Ainsi, chaque gamme décrit la séquence d'opérations (c-à-d d'unités de transformation) par laquelle devra passer un lot de fabrication pour obtenir un produit déterminé. Un produit peut être le résultat de plusieurs gammes. Chaque opération est associée à une unité de transformation définie dans le modèle opérationnel. Il est possible par l'intermédiaire des gammes d'obtenir le chemin que devra parcourir un lot dans le système, ainsi que les temps opératoires nécessaires pour chaque opération.

La figure 9.3 illustre l'ensemble des artefacts graphiques disponibles pour construire un modèle des gammes opératoires. Chacune d'entre elles doit débuter par au moins une *source de matière* et se terminer par une et une seule *fin de gamme*. La gamme est construite à partir d'*opérations* qui représentent les transformations réalisées sur un produit, et à partir de *liens* obligatoirement étiquettés par un nom de produit qui a été défini dans une nomenclature. De plus, une opération possède deux attributs obligatoires :

- un *délai* représentant le temps de traitement d'une opération sur un produit. Ce délai est obligatoirement défini dans le sous-système informationnel (*cf.* section 9.2.2.4).
- une référence vers une ou plusieurs *unités de traitement* du sous-système opérationnel. Elles représentent les unités physiques pouvant réaliser la tâche de transformation.

9.2.2.3 Définition d'entité

Les entités représentent les informations physiques et décisionnelles qui circulent dans le modèle. Par l'intermédiaire de la définition d'une entité dans le sous-système informationnel, il est possible de spécifier ses attributs (numéro de lot, produits pouvant être transportés, ...). D'une manière plus pragmatique, les lots de fabrication peuvent être assimilés aux entités parcourant l'infrastructure physique, et ordres de fabrication celles circulant entre les centres de décision.

Nous pouvons faire un parallèle entre la définition d'entités et la conception orientée objets. En effet, la définition d'une entité correspond à celle d'une classe C, et l'entité se déplacant dans le modèle à une instance de cette classe. Nous ne proposons pas de formalisme graphique pour décrire une classe d'entité. Toutefois, le tableau 9.1 peut être utilisé comme canevas de description.

Nom de la classe d'entité (modèle de simulation)	
attribut_1	valeur initiale_1
...	
attribut_n	valeur initiale_n
produit_1	quantité_1
...	
produit_m	quantité_m

TAB. 9.1 – Canevas pour la définition d'une entité

9.2.2.4 Activités et délais

Le temps est une notion primordiale pour la simulation. La spécification de ce dernier peut être réalisée de plusieurs façons :

- par une spécification statique en précisant une date, une durée ou un intervalle (exemple : [34..156], 11224, $[12, +124]$) ;
- par une loi statistique : cette dernière est généralement obtenue après étude des données provenant du système réel. À partir de ces informations, il est possible de déduire une loi statistique qui pourra être utilisée pour calculer aléatoirement un espace de temps. Par exemple, citons les lois normale, exponentielle, ou encore beta [DEVORE, 1995 ; HOGG et CRAIG, 1994 ; DEVROYE, 1985]. Une loi statistique est spécifiée de deux manière différentes :
  - utilisation du mnémonique associé à une loi déjà connue : `NORMALE(m,d)` qui représente une loi normale de moyenne `m` et d'écart type `d` ;
  - utilisation d'une spécification mathématique (si l'environnement informatique de modélisation le permet) : $\forall x, f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}$ où μ est la moyenne et σ^2 la variance.
- par une source de données extérieure : *cf.* section 9.2.2.5.

Les lois temporelles sont utilisées par l'ensemble des objets du système. Elles permettent de décrire les contraintes temporelles nécessaires à la modélisation du comportement dynamique du système. En plus de la description temporelle, une loi doit être identifiée par un identificateur unique.

9.2.2.5 Sources extérieures

La simulation peut avoir besoin d'un grand nombre d'informations déjà stockées dans des supports extérieurs à notre plateforme (bases de données, ...). Nous proposons que le modèle informationnel contienne la description de ces différentes sources. Ainsi, elles pourront être utilisées dans l'ensemble du modèle de simulation. Toutefois, comme pour les activités et les délais, nous ne proposons pas de formalisme de spécification de ces sources de données. Nous supposons que l'utilisateur de $\mathcal{MAMA}$-$\mathcal{S}$

fournira durant les phases de conception et de réalisation l'ensemble des propriétés et des méthodes nécessaires pour accéder à ces sources de données.

9.2.2.6 Distribution

Une partie des objets du sous-système informationnel peuvent être partagés, ou représenter un artefact distant. Selon leur type (distant ou non), ils possède respectivement un des ensemble d'attributs suivant :

- `sharable` indique si l'artefact peut être partagé, et `shared_name` spécifie son nom de partage ;
- `remote_address` représente le nom d'un artefact distant.

Les éléments qui supportent la distribution sont : les modèles de nomenclatures et de gammes opératoires, et les définitions d'entités. Nous limitons les possibilités. En effet, nous considérons que le partage des informations contenues dans un modèle de gamme ou de nomenclature augmentent les possibilités d'incohérence, d'incompréhension et d'erreur au sein des modèles de simulation. D'autre part, nous pensons que les possibilités de distribution mises de côté ne sont pas utilisées dans le milieu industriel.

9.2.2.7 Exemple

Reprenons l'exemple du groupement d'entreprises présenté dans la section 9.2.1.6. Considérons les modèles de nomenclatures (*cf.* figure 9.4). Chaque entreprise possède sa propre vision des produits. Toutefois, le `capteur` utilisé par l'entreprise E_2 est en fait la définition se trouvant dans le modèle de E_1. Ainsi, nous avons un exemple simple d'utilisation de définitions distantes de produits. Pratiquement, le `capteur` de E_1 possède les attributs `sharable` et `shared_name` positionnés respectivement aux valeurs « `yes` » et « `def_capteur` ». Le `capteur` de E_2 possède l'attribut `remote_address` avec la valeur « `E1.def_capteur` ». Il existe un lien du même type entre le modèle de l'entreprise E_3 et celui de E_2.

Considérons maintenant les gammes opératoires. Dans notre exemple, nous considérons que seules les deux premières entreprises doivent définir des modèles de gammes opératoires. En effet, E_3 ne réalise aucune transformation sur les produits. La figure 9.4 illustre la représentation graphique que nous proposons au sein de $\mathcal{MAMA}$-$\mathcal{S}$. Ainsi, dans E_1, la matière première (M.P.) est transformée en capteurs par l'atelier A, et dans E_2, les caméras sont obtenues à partir de l'assemblage de capteurs et de boitiers. Notons qu'à chaque opération d'une gamme est associée une unité de traitement du sous-système opérationnel qui possède le même nom que l'opération. D'autre part, une opération doit définir un ensemble de délais nécessaires pour modéliser leurs traitements. Dans l'exemple, nous considérerons que toutes les

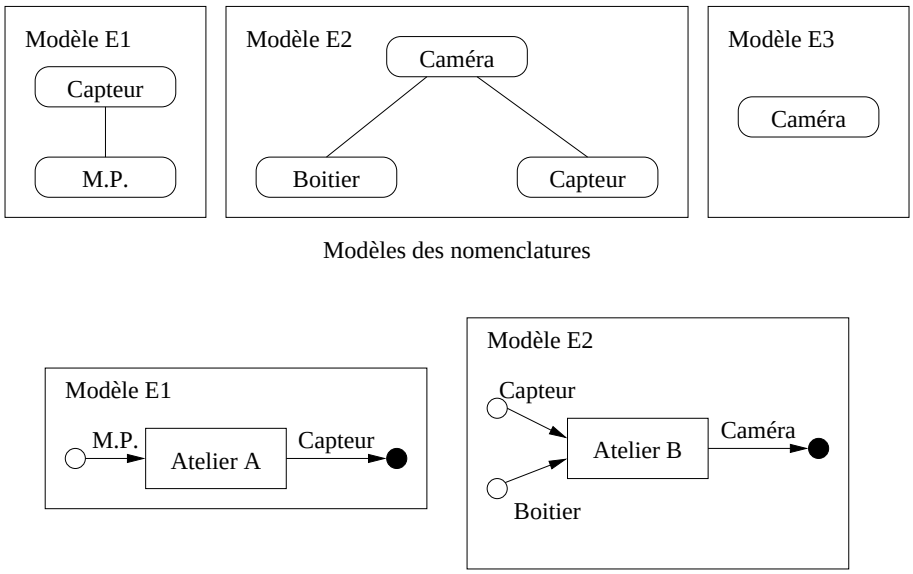


FIG. 9.4 – Exemple de modèles du sous-système informationnel

opérations obéissent à une loi normale de moyenne 10 et de dérivation standard 0.5 (`NORMALE(10,0.5)`).

Maintenant, nous pouvons nous intéresser à la définition des classes d'entités. Pour cela, nous définissons une classe dans chaque modèle :

- Dans E_1, les entités du flux physique possèdent un attribut correspondant aux quantités de produits qu'elles peuvent transporter. Cet attribut prend les valeurs 500 pour les matières premières et 100 pour les capteurs. Nous définissons une seconde classe d'entités correspondant à un ordre de fabrication transitant dans le sous-système décisionnel. Cette entité transporte un attribut correspondant à la quantité de produit à fabriquer.
- Dans E_2, les entités « physiques » sont définies sur le même modèle que celles de E_1. Elles sont capables de transporter 200 capteurs et 10 boitiers ou caméras (ces derniers ayant la même taille physique). Le modèle de E_2 définit lui aussi une classe d'entités correspondant aux ordres de fabrication.
- Enfin, le modèle E_3 utilise les mêmes classes d'entités que celle définies dans E_2.

9.2.3 Sous-système décisionnel

Le sous-système décisionnel permet de décrire l'ensemble des moyens de pilotage du système de production. Cette description est appellée structure organisationnelle.

9.2.3.1 Structure organisationnelle

Nous basons notre approche sur les travaux de [Burlat, 1996 ; Kabachi, 1999] qui proposent une approche à agents pour modéliser le système décisionnel. Mais plus encore, nous nous rapprochons des travaux de [Berchet, 2000]. Elle propose une approche de modélisation du sous-système décisionnel qui est intéressante par son utilisation des concepts de coordination et de coopération entre les différents acteurs de ce système. Ainsi nous pensons que le sous-système décisionnel est composé d'un ensemble de *centres de décision* [Berchet, 2000]. Ces centres ont pour rôle de prendre des décisions en fonction d'informations qu'ils récoltent dans leur environnement, et d'envoyer des « ordres » à d'autres centres de décision ou à des objets appartenant aux autres sous-systèmes. La figure 9.5 illustre l'ensemble des représentations graphiques associées à chaque artefact de modélisation. Chaque centre se positionne parmi trois horizons [Kabachi, 1999] :

– **Niveau opérationnel :**

  Les centres de décision du niveau opérationnel (ou centres opérationnels) sont les seuls à pouvoir piloter directement les objets appartenant au sous-système opérationnel. Un exemple de centre est le gestionnaire d'allocation de ressources.

– **Niveau tactique :**

  Les centres composant ce niveau ont des relations hiérarchiques directes avec les centres des autres niveaux, et des relations de coopération avec les centres de même niveau. Leur rôle est de prendre en charge des décisions à moyen terme (ordonnancement des ordres de fabrication, ...).

– **Niveau statégique :**

  Ces centres de plus haut niveau se spécialisent dans une gestion à long terme du système (gestion des clients, ...).

Les centres sont liés par des relations qui peuvent être de trois types : *autoritaire*, *accointance* et *communication*. Cette vision est issue des travaux de Mahdi Hannoun sur Moise qui est une approche de modélisation des organisations dans les systèmes multi-agents [Hannoun et al., 2000]. En effet, nous considérons que, bien qu'adaptée aux SMA, cette vision des liens reste très proche des sociétés et plus particulièrement des systèmes industriels de production. Décrivons maintenant les caractéristiques de chacune de ces classes de liens :

– **Liens d'autorité :**

  Les liens d'autorité (ou liens hiérarchiques) représentent des relations fortes entre des centres de décision. Ainsi, lorsque le centre de niveau supérieur (source du lien) envoie un message au centre de niveau inférieur (destination du lien), ce dernier doit obligatoirement traiter positivement ce lien, c'est-à-dire qu'il

réalisera la tâche décrite dans le message. Ainsi, le centre source est assuré que son message (que nous nommons « ordre » dans ce cas précis) soit respecté et qu'il y ait au moins une tentative de réalisation. Par exemple, le centre A envoie un ordre de fabrication au centre B. Dans ce cas, B ne pourra pas ignorer l'ordre, et l'inclura dans une nouvelle planification de la production. Par contre, si cet ordre provoque des problèmes (dépassement des délais de production, ...), B doit prévenir A tout en considérant le nouveau plan de production comme valide. Nous voyons bien que B ne peut refuser les ordres provenant de A.

– **Liens de communication :**

Les liens de communication représentent les relations non hiérarchiques. Ainsi, contrairement aux liens hiérarchiques, un centre de décision peut très bien ignorer un message lui parvenant d'un centre en communication avec lui. Ainsi, nous garantissons que les messages arrivent bien au destinataire, mais pas que ce dernier les traite.

– **Liens d'accointance :**

Enfin, les liens d'accointance représentent le fait qu'un centre connait l'existence d'un autre centre. Mais contrairement, aux deux autres types de liens, celui-ci interdit l'envoi de messages. Pour pallier cette contrainte, il faut transformer le lien d'accointance en un lien hiérarchique ou de communication. Notons que ces deux derniers impliquent nécessairement l'existence d'une relation d'accointance.

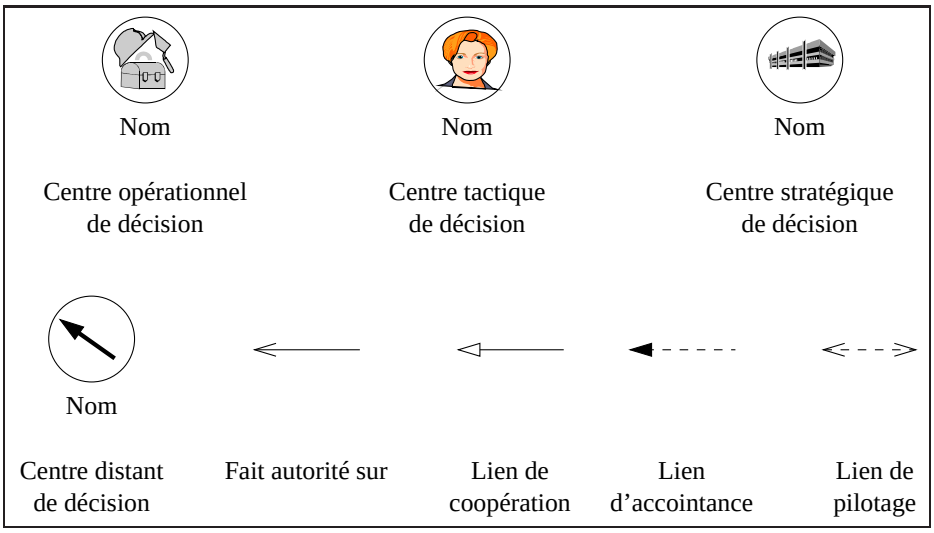


FIG. 9.5 – Un formalisme pour le sous-système décisionnel

Il existe un quatrième type de lien. Ce dernier, appelé *lien de pilotage*, permet de modéliser une relation de pilotage entre un centre opérationnel de décision et un

objet appartenant au sous-système opérationnel. Ainsi, un centre possédant un tel lien peut accéder aux services proposés par l'objet « physique ».
D'autre part, nous introduisons un quatrième type de centre qui correspond aux centres définis dans d'autres modèles de simulation. Ces *centres distants* ne sont que des pointeurs vers des centres dont le modèle en cours de réalisation ne connait que ce que le modèle distant veut bien lui fournir (en général une description informelle des services fournis par ce centre distant). Ainsi, grâce à cet artefact de modélisation, nous pouvons introduire le support des systèmes industriels distribués.

9.2.3.2 Comportements

Chaque centre de décision est défini par son comportement interne. Dans [Galland et al., 2001a], nous distinguons deux types de comportement :

– **comportement réactif :**

Les centres de décision réagissent directement en produisant des actions (ordre, demande, ...) dès qu'un stimulus leur parvient. Ainsi, aucun algorithme particulier n'est mis en œuvre par ces centres. Il ne s'agit que d'un comportement simple d'action-réponse. L'exemple typique de ce type de centre est celui des gestionnaires d'allocation des ressources. Le comportement peut être décrit par du pseudocode respectant la grammaire suivante et décrite plus en détail dans l'annexe B :

```
comportement ::= "CONTEXT" ident "WHEN" événement
                 "THEN" code "END"

événement ::= "EACH_TIME" '(' date ')'
            |
            "RECEIVE" ident [ "FROM" partenaire  ]
            [ "WITH PARAMS" paramètres  ]
            |
            condition

code ::= instruction ';' code |vide

paramètres ::= ident [ ',' paramètres  ]

partenaire ::= ident
             |
             "OPERATIONAL" ident

date ::= délai
       |
       heure
       |
       "INFORMATION" '(' ident expressions ')'
```

```
instruction ::= "SEND TO" ident "TYPE" type
                "NAMED" ident "DATA" expression
                |
                "FOREACH" '(' expression [ "AS" ident  ]  ')'
                "DO" code "DONE"
                |
                "WHILE" expression "DO" code "DONE"
                |
                "IF" expression "THEN" code [ "ELSE" code  ]
                "END IF"
                |
                "CALL" ident '(' ident expressions ')'
                |
                "OPERATION" '(' ident expressions ')'
                |
                "INFORMATION" '(' ident expressions ')'
                |
                ident '=' expression
                |
                vide
expressions ::= ',' expression expressions
                |
                vide
expression ::= ident
               |
               "OPERATION" '(' ident expressions ')'
               |
               "INFORMATION" '(' ident expressions ')'
               |
               code OCL
type ::= "ORDER"
         |
         "COMMUNICATION"
condition ::= code OCL
ident ::= chaîne
```

– **comportement cognitif :**

Les centres mettent en œuvre un algorithme leur permettant de répondre à leurs objectifs (planification, pilotage, ...). Ces comportements doivent être

eux aussi décrits dans le modèle décisionnel. Nous distinguons plusieurs types de spécification de comportements :

- algorithmique : langage de spécification procédural ou langage de programmation (langage Pascal, langage algorithmique, ...),
- événementiel : langage à base d'événements (Visual Basic®, ...),
- formelle : spécifications formelles comme le langage associé à la méthode B [Abrial, 1996],
- par machine à états comme ceux proposés par le langage Uml [Muller, 1997],
- à l'aide de langages à acteurs [Hewitt, 1977],
- par des méthodes issues des systèmes multi-agents : par exemple, description formelle des objectifs, des croyances, des intensions des centres avec l'approche bdi (*cf.* section 6.3.4).

9.2.3.3 Exemple

Reprenons notre exemple de groupement de trois entreprises. Ici, nous nous intéressons exclusivement à la modélisation du sous-système décisionnel. Dans un premier temps, nous nous intéressons au mode de gestion de ce groupement. Ici, nous considérons qu'il s'agit d'une gestion à la fois en flux tiré et en flux poussé. En effet, l'entreprise E_3 est en relation directe avec le marché. Chaque fois que son stock de caméra ne permet pas de répondre à la demande, cette société envoi un ordre de fabrication à E_2. Cette dernière est gérée en flux tiré. À chaque ordre de fabrication provenant de E_3, elle fait correspondre un ordre de fabrication pour E_1. Enfin cette dernière entreprise lance la production du nombre de capteurs réclamés par E_2. Le processus de production utilise quant à lui une politique de gestion en flux poussé. La figure 9.6 illustre les modèles décisionnels pour chacun des trois membres du groupement.

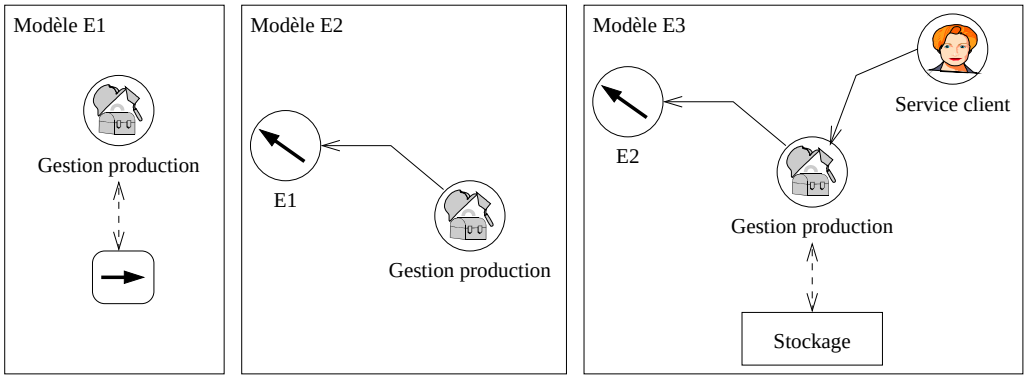


Fig. 9.6 – Exemple de modèles du sous-système décisionnel

Chaque centre de décision possède son propre comportement :

– **Service client :**

La service client génère des ordres de fabrication selon une loi statistique déterminée durant la phase de conception (récolte et analyse des données préliminaires). On considère qu'il s'agit d'un comportement typiquement réactif. En effet, ce centre réagit uniquement à un stimulus temporel et envoie irrévocablement des ordres de fabrication au centre de gestion de la production. Le comportement peut être décrit ainsi :

```
CONTEXT "Service client"
WHEN EACH_TIME( INFORMATION( "loi arrivée commandes" ) )
THEN
  SEND TO "Gestion production"
    TYPE ORDER
    NAMED "ordre de fabrication"
    DATA INFORMATION( "taille d'un OF" ) ;
END
```

– **Gestion de production de E_3 :**

La gestion de production permet de faire le lien entre les ordres de fabrication provenant du service clientèle et l'état réel du stock. Si ce dernier est trop faible pour répondre à la demande du marché, le centre de gestion de la production envoi un ordre de fabrication à l'entreprise E_2. Le comportement de ce centre est le suivant :

```
CONTEXT "Gestion production"
WHEN RECEIVE "ordre de fabrication"
     FROM "Service client"
     WITH PARAMS ( "quantité à produire" )
THEN
  état_stock = OPERATION( get_quantity ) ;
  IF état_stock < "quantité à produire"
  THEN
    SEND TO E2
      TYPE ORDER
      NAMED "ordre de fabrication"
      DATA  "quantité à produire" ;
  END IF
END
```

– **Gestion de production de E_2 :**
Ce centre génère un ordre de fabrication de capteurs vers E_1 pour chaque ordre provenant de E_3 :

```
CONTEXT "Gestion production"
WHEN RECEIVE "ordre de fabrication"
     WITH PARAMS ( "taille OF" )
THEN
  SEND TO E1
    TYPE ORDER
    NAMED "ordre de fabrication"
    DATA "taille OF" *
         INFORMATION( nomenclature_camera, "quantité capteur" ) ;
END
```

– **Gestion de production de E_1 :**
Ce centre génère une entité physique pour chaque capteur devant être produit. Ainsi, son comportement peut être défini comme suit :

```
CONTEXT "Gestion production"
WHEN RECEIVE "ordre de fabrication"
     WITH PARAMS ( "taille OF" )
THEN
  i = 1
  WHILE( i <= "taille OF" )
  DO
    OPERATION( generate_entity, INFORMATION( entité ) ) ;
    i = i + 1 ;
  DONE
END
```

9.3 Métamodèle UML

Dans cette section, nous présentons le métamodèle Uml qui est à la base du langage de modélisation que nous proposons pour de la phase de spécification. Nous n'expliquerons pas les principes de la métamodélisation dans ce mémoire. Plus d'informations sont disponibles dans les documents relatifs à Uml [Muller, 1997 ; Booch et al., 1997].
La métamodélisation Uml nous semble très adaptée pour la définition de nouveaux langages graphiques de modélisation. D'autre part, des travaux comme ceux de [Er-

KOLLAR et FELFERNIG, 1999] illustrent parfaitement l'adéquation entre UML et les modèles de simulation.

La spécification du métamodèle complet est présentée dans l'annexe A. Nous y exposons la syntaxe abstraite, les règles de construction et les points particuliers de la sémantique attachée. La syntaxe abstraite est un diagramme des classes UML qui représente les contraintes structurelles et fonctionnelles des divers artefacts de modélisation. Les règles de construction sont des contraintes sur le comportement et les fonctionnalités des objets. Nous les exprimons en logique des prédicats du premier ordre. Toutefois, elles peuvent être directement traduites en « Object Constraint Language » (OCL) qui est le langage de spécification des contraintes proposé par UML [BOOCH et al., 1997].

Dans cette section, nous allons nous limiter à l'explication de quelques extraits du métamodèle.

9.3.1 Sous-système opérationnel

La figure 9.7 illustre un extrait du métamodèle UML décrivant la notion d'unité de traitement (PROCESSINGUNIT). Cette syntaxe abstraite représente un ensemble de

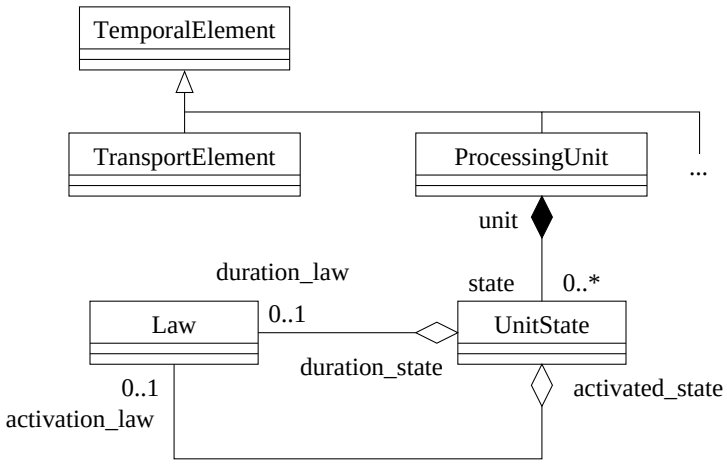


FIG. 9.7 – Extrait du métamodèle du sous-système opérationnel

classes définissant les artefacts de modélisation. Elle définit le fait qu'une unité de traitement possède un nombre fini d'états (*state*). Ces derniers possèdent comme propriété un délai représentant le durée d'activation de l'état (*duraction_law*) ou une activité représentant la date à laquelle l'état devient actif (*activation_law*). Plus précisement, les objets composant la syntaxe abstraite peuvent être définis comme suit :

Law

Law représente une expression temporelle utilisée par les modèles de simulation.

Associations

activated_state: Représente l'état d'une unité de production dont l'activation est déterminée par une loi.

duration_state: Correspond à un état d'une unité de production dont la durée de validité est déterminée par une loi.

Méthodes

+ sameAs(x : Law) Indique si l'expression de la loi courante est équivalente à l'expression de la loi passée en paramètre. Cette implantation renvoie toujours la valeur booléenne `vrai` (*cf.* surcharges dans les classes filles).

ProcessingUnit

ProcessingUnit représente les unités de production (machines, ...).

Associations

manager: Représente l'entité décisionnelle supervisant cette unité de production.

routingunit: Désigne un éventuel lien entre l'unité de production du flux physique et une unité de traitement des produits du modèle des gammes.

state: Une unité de production peut être dans différents états durant sa vie (panne, maintenance, travail, repos, ...). Cette association permet de représenter l'ensemble des états possible d'une unité de production.

TemporalElement

TemporalElement est une abstraction d'une ressource utilisant une activité ou un délai pour calculer le temps de traitement d'une tâche sur une entité.

TransportElement

Les entités ont parfois le besoin d'être transportées d'une partie du modèle vers un autre. Nous introduisons les éléments de transport qui permettent ce transfert.

UNITSTATE

Chaque unité de production peut posséder des états différents (panne, maintenance, repos, ...). Cette classe d'objets permet de décrire chacun d'entre eux. Un état est caractérisé par une loi d'activation permettant de déterminer les instants où la machine doit changer d'état, ou par une loi statistique représentant la durée d'activation.

Associations

activation_law: Représente la loi d'activation de l'état, c'est-à-dire la loi statistique déterminant la périodicité d'activation de l'état.

duration_law: Désigne le lien existant entre un état et la définition d'une loi déterminant la durée d'activation d'un état. Par exemple, l'état de « panne » a une durée déterminée par une loi NORMALE.

unit: Désigne l'unité de production à laquelle l'état appartient.

Attributs

+ entityprocessing : Boolean = false — L'état d'une unité de traitement peut correspondre au traitement « normal » des entités (utilisation des lois de traitement héritées de TEMPORALELEMENT) ou à des états ne réalisant aucun traitement sur ces dernières (panne, ...). Cet attribut permet de spécifier le type d'état dont il s'agit. Par défaut, un état n'est pas considéré comme un état de traitement « normal ». Le modélisateur devra donc indiquer explicitement que l'état qu'il déclare est un état permettant à la ressource associée d'utiliser les lois statistiques héritées de TEMPORALELEMENT.

Règles de bonne construction:

[1] Si une loi de durée est associée à un état, il y a toujours une loi d'activation associée à ce même état:
$\forall x / x \in$ UNITSTATE,
$\text{card}(x.durationlaw) > 0 \Rightarrow \text{card}(x.activationlaw) > 0$

9.3.2 Sous-système informationnel

La figure 9.8 illustre un extrait du métamodèle Uml décrivant les artefacts de modélisation d'un modèle de nomenclature. Cet extrait de métamodèle représente la notion de lien de composition entre un produit composé (ComposedProduct) et un composant (Product). Ainsi, la classe CompositionLink représente l'artefact de modélisation correspondant à un lien de composition. Les classes de cet extrait du métamodèle Uml sont définies comme suit :

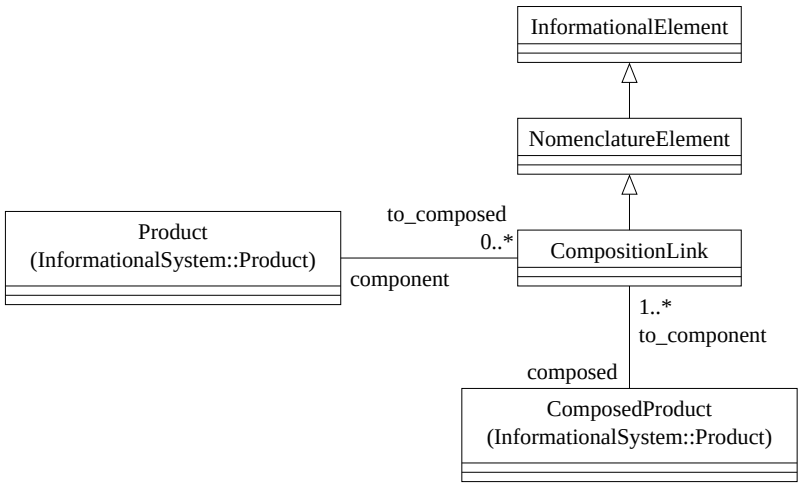


Fig. 9.8 – Extrait du métamodèle du sous-système informationnel

CompositionLink

Les produits et leurs composants sont reliés par des transitions afin de schématiser les différentes constructions de produits possibles. Une transition part d'un produit décomposable pour aboutir à un produit quelconque.

Associations

component: Une transition aboutit à un produit quelconque composant le produit de départ.

composed: Une transition part d'un produit décomposable.

Attributs

+ quantity : Integer = 1 La quantité de composés pour former le composé.

Règles de bonne construction:

[1] Les deux produits d'un lien de composition doivent appartenir au même modèle de nomenclatures que l'instance de CompositionLink:
$\forall w, x, y, z /$
$(w, x, y, z) \in$ NomenclatureModel $\times$ CompositionLink $\times$ Product^2,
$(x \in w.\text{allContents} \wedge y \in x.component \wedge z \in x.composed)$
$\Rightarrow (y \in w.\text{allContents} \wedge z \in w.\text{allContents})$

[2] Les deux produits d'un lien de composition doivent être différents:
$\forall w, x, y, z /$
$(w, x, y, z) \in$ NomenclatureModel $\times$ CompositionLink $\times$ Product^2,
$(x \in w.\text{allContents} \wedge y \in x.component \wedge z \in x.composed)$
$\Rightarrow y \neq z$

ComposedProduct

Cette classe d'objets représente les produits composés à partir d'autres produits. Il s'agit de la seule définition d'un produit pouvant être partagée avec d'autres modèles informationnels.

Attributs

+	::sharable : Boolean = true	Un produit composé peut être partagé avec d'autres modèles d'informations.

Associations

to_component: Un produit composé fait l'objet d'une définition de composition à travers un CompositionLink.

Règles de bonne construction:

[1] Les produits composants doivent appartenir au même modèle de nomenclatures:
$\forall x, y, z / (x, y, z) \in$ ComposedProduct $\times$ Product $\times$ NomenclatureModel,
$(y \in x.component \wedge z \in x.inmodel) \Rightarrow z \in y.inmodel$

InformationalElement

Cette classe représente les éléments de modélisation composant le modèle informationnel.

NomenclatureElement

Cette classe définit l'élément de base du modèle de nomenclatures. Chaque structure composant ce dernier doit obligatoirement être une spécialisation de NomenclatureElement (sauf pour les produits).

Règles de bonne construction:

[1] Un élément de modélisation de type NomenclatureElement appartient uniquement à un modèle de nomenclatures:
$\forall x, y/(x, y) \in$ NomenclatureElement $\times$ SimulationModel,
$x \in y$.allContents $\Rightarrow y \in$ NomenclatureModel

Product

Un produit est la source ou le résultat d'un processus de production. Il peut être de la matière première, des produits semi-finis ou finis. La classe Product est une abstraction qui est spécialisée selon le type du produit.

Associations

composed: Un produit peut entrer dans la composition d'autre produit.

features: Correspond à l'ensemble des attributs attachés à un produit.

Opérations additionnelles:

[1] L'opération attributNames permet de construire l'ensemble des noms des attributs associés au produit:
attributNames $= \{n/\forall x \exists n, x \in self.features \wedge n = x.\ name\}$

9.4 Vérification de cohérence

L'une des principale qualités attendues pour une méthodologie est la proposition de mécanismes permettant de vérifier ou de valider les modèles à partir des spécification des langages ou des formalismes décrits dans le guide méthodologique. Ainsi, $\mathcal{M}_A\mathcal{MA}$-$\mathcal{S}$ met en place deux mécanismes de vérification. Le premier (dit « vérification locale ») consiste à valider la cohérence d'utilisation des artefacts de modélisation. Le second (dit « vérification distribuée ») permet de pallier aux déficiences de vérification du premier mécanisme vis-à-vis d'un modèle de système distribué. En effet, les modèles de simulation n'étant pas omniscients, il est nécessaire de vérifier que les paramètres utilisés par l'un correspondent aux paramètres définis par l'autre.

9.4.1 Vérification locale

La création d'un modèle de simulation impose la vérification de la cohérence de celui-ci. Ainsi, la vérification locale est réalisée en comparant le modèle d'un utilisateur avec le métamodèle Uml intégré à $\mathcal{M}_A\mathcal{MA}$-$\mathcal{S}$. Ce métamodèle décrit toutes les contraintes structurelles et fonctionnelles autorisées pour modéliser un système industriel distribué.

L'utilisation d'un langage graphique orienté objet (diagrammes de classes) et d'un langage formel à base de logique (logique des prédicats du premier ordre ou son adaptation en langage informatique : « Object Constraint Language » [BOOCH et al., 1997]) permet de mettre en œuvre des mécanismes simples de vérification des modèles.
L'implantation effective de ces mécanismes de vérification peut différer d'un logiciel à un autre. Prenons l'exemple d'ARGO/UML [2] ou d'ARAKHNÊ [3] . À chaque modification apportée au modèle, ces logiciels calculent la validité de certaines propriétés susceptibles d'influencer l'action réalisée. Si l'évaluation révèle une incohérence d'utilisation ou de modélisation, alors ARGO/UML et ARAKHNÊ génèrent une critique du modèle qui est immédiatement donnée à l'utilisateur. De plus, ces deux logiciels considèrent que les contraintes structurelles imposées par le langage orienté-objet ne peuvent être transgressées. Si cela se produit, les critiques se transforment en erreurs et l'opération ayant engendré le problème est annulée.
Prenons l'exemple illustré par la figure 9.9. Il représente un extrait de métamodèle composé de trois classes. La classe A est composée d'au maximum 5 instances de la classe B. D'autre part, la classe C (héritière de B) possède une relation simple avec la classe B. La contrainte entre les deux associations indique que l'instance liée à C doit appartenir à la relation de composition héritée.

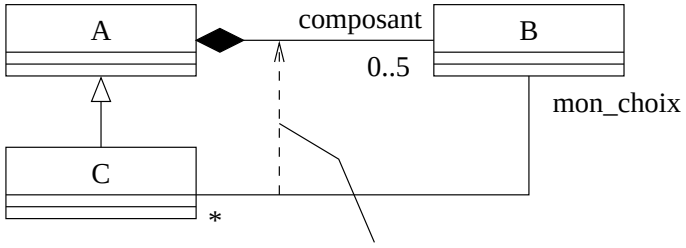


FIG. 9.9 – Extrait d'un métamodèle pour la vérification locale

Ce métamodèle représente donc les contraintes structurelles et sémantiques devant être respectées par un modèle. Soit I_C une instance de la classe C existant dans le modèle à vérifier. Du métamodèle présenté dans la figure 9.9, nous pouvons déduire l'ensemble de contraintes suivant :

1. $0 \leq \text{card}(I_C.\text{composant}) \leq 5$ (contrainte héritée de A)
2. $\forall x \in I_C.\text{composant}, x$ `instanceof` B (contrainte héritée de A)
3. $\text{card}(I_C.\text{mon\_choix}) = 1$
4. $\forall x \in I_C.\text{mon\_choix}, x$ `instanceof` B

[2] `http://argouml.tigris.org/`
[3] `http://www.arakhne.org/`

5. $\forall x \in I_C$.mon_choix, $x \in I_c$.composant

Ces contraintes sont écrites en utilisant la syntaxe de l'« Object Constraint Language » [Booch et al., 1997], et où, par souci de lisibilité, nous remplaçons les mots clés du langage par les symboles mathématiques correspondants.
Ainsi, chaque fois que l'instance I_C est modifiée, le mécanisme de vérification évalue l'ensemble de ces contraintes et génère une erreur si l'une d'entre elles n'est pas vérifiée.

9.4.2 Vérification distribuée

Si la vérification statique des modèles est nécessaire, elle n'est pas suffisante. En effet, nous considérons que chaque composante d'un système distribué peut posséder son propre modèle. Leurs constructions peuvent être réalisées parallèlement. La vérification statique permet de vérifier la cohérence d'un modèle, mais pas celle de l'utilisation d'objets définis dans des modèles distants. Prenons l'exemple de deux modèles modélisant un système distribué. Ce système est composé de deux machines, A et B. Elles utilisent la même ressource R. Ayant choisi arbitrairement de créer deux modèles, il est donc nécessaire que ces derniers partagent l'utilisation de la ressource. Nous avons choisi de définir les caractéristiques de R dans B. Ainsi, dans A apparaît une « ressource distante », c'est-à-dire une ressource dont la définition des caractéristiques réelles appartient à un autre modèle (en l'occurrence B). Les mécanismes de vérification statique nous permettent, de valider la concordance entre chacun des modèles et les possibilités du langage graphique de modélisation. Par contre, le métamodèle Uml ne permet pas de vérifier que les attributs réels de R (définis dans B) sont compatibles avec la manière dont est utilisée R dans A (est-ce le bon type de ressource ? ...).

9.4.2.1 Architecture générale

Nous proposons de mettre en place un second mécanisme de vérification qui permettra de valider la cohérence d'un modèle distribué. Pour cela, nous avons décidé d'utiliser un système multi-agents. Ce système est totalement différent de celui qui est produit par la phase de conception. Son unique rôle est d'aider les concepteurs à créer des modèles abstraits cohérents. La figure 9.10 illustre l'architecture du système multi-agents pour la vérification. Nous considérons que le modèle de simulation est une composante de l'environnement (ou ressource dans le sma). Pour chacune de ces ressources, un agent est défini. Il a pour rôle de mettre en œuvre le processus de vérification. Nous voyons aussi apparaître des agents facilitateurs. Le rôle de ces derniers est de faciliter la mise en communication des agents entre eux. Nous abordons plus précisément la définition de ces agents dans le chapitre de la conception.

En effet, il s'agit là d'une composante essentielle de l'architecture multi-agents pour la simulation que nous proposons.

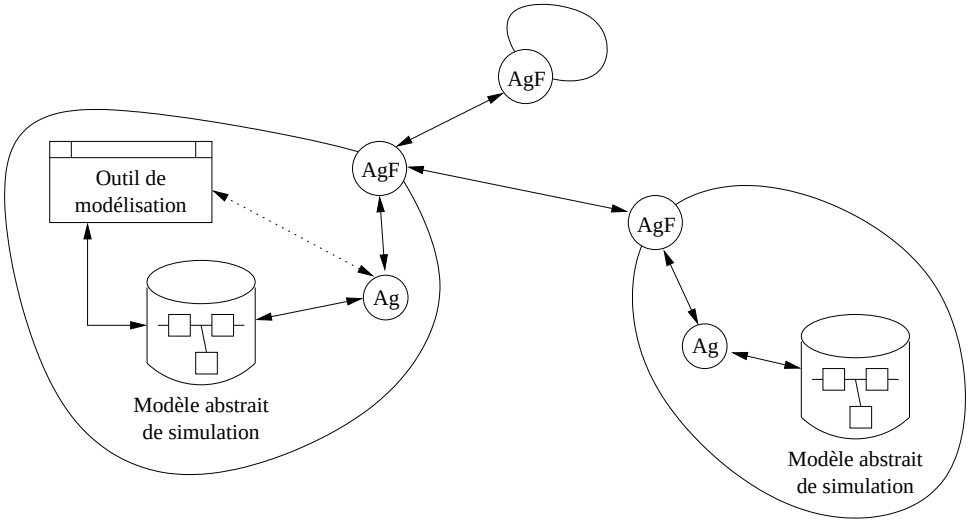


FIG. 9.10 – Architecture du SMA de vérification

Plusieurs architectures permettaient de répondre à cette problématique de vérification dynamique. Nous avons décidé de privilégier les systèmes multi-agents car ils nous semblent avoir les avantages suivants sur un système composé d'objets distribués :

- La mise en œuvre de protocoles de communications est facilitée par l'intégration d'une facette Interaction dans la définition des SMA.
- Trouver la définition distante d'un artefact de modélisation est facilitée par l'utilisation d'une partie de l'architecture proposée pour le modèle de simulation orienté multi-agents. Ainsi, le service de « pages jaunes » des agents facilitateurs nous permet de savoir à qui s'adresser pour avoir la définition d'un artefact.
- Nous considérons que le développement des modèles est parallèle et asynchrone. D'autre part, nous voulons différencier le modèle de l'environnement permettant de le construire. Ainsi, si nous considérons que ces derniers ne sont pas toujours actifs, il est nécessaire pour les modèles de l'être. Ce fait est rendu nécessaire par le besoin des autres modèles d'accéder aux éventuelles définitions partagées. Les SMA nous semblent adaptés à cette persistance du mécanisme de vérification vis-à-vis des outils de modélisation.
- La dernière raison, pour laquelle nous avons choisi un SMA, est le besoin d'autonomie que doit avoir un tel système de vérification. Les agents de vérification doivent avoir un comportement autonome et indépendant de l'outil de modélisation. Nous considérons que ce dernier n'est pas toujours présent. Par conséquent, il faut que les agents possèdent une certaine autonomie de réaction pour réagir aux changements se produisant dans les modèles distants.

Nous allons maintenant nous attacher à définir un peu plus formellement les caractéristiques du système multi-agents que nous proposons. Pour cela, nous utilisons l'approche « Voyelles » présentée dans le chapitre 6.3.

9.4.2.2 Définitions préalables

Nous définissons un certain nombre d'opérateurs qui nous permettrons de clarifier les définitions :

- La notation $\mathcal{E}_o$ est le membre $\mathcal{E}$ du n-uplet définissant o ;
- id_{obj} est l'identification de l'objet obj ;
- $exp[v/id]$ est l'opérateur qui permet de remplacer toutes les occurences de id par v dans l'expression exp ;
- $\mathrm{val}_{obj}(id)$ est la valeur de l'attribut identifié par id et appartenant à l'objet obj ;
- owner_{obj} est l'objet qui contient l'objet obj ;
- $\mathrm{eval}_{obj}(exp)$ est une fonction permettant d'évaluer la validité de l'expression exp dans le contexte d'obj. exp doit être une formule évaluable selon la théorie de la logique des prédicats du premier ordre. Le contexte de obj correspond à l'espace de nom dans lequel se trouve obj. $\mathrm{eval}_{obj}(id)$ est équivalent à :
$$\begin{cases} \mathrm{val}_{obj}(id) & \text{si } \forall a, \langle id, a\rangle \in \mathcal{A}_{obj}, \\ \mathrm{eval}_{\mathrm{owner}_{obj}}(id) & \text{sinon.} \end{cases}$$
- Nous noterons $\vec{obj}$ la définition distante d'un objet obj. Ceci implique que les formules suivantes soient vérifiées :
  - $\models \mathrm{val}_{obj}(remote)$,
  - $\models \mathrm{val}_{\vec{obj}}(sharable)$, et
  - $\models \mathrm{id}_{obj} = \mathrm{id}_{\vec{obj}}$.

9.4.2.3 Définition de l'environnement

L'environnement du système multi-agents de vérification est composé de deux types d'éléments :

- **Outils de modélisation :**

  Ces outils mettent en œuvre la méthodologie $\mathcal{MAMA}$-$\mathcal{S}$. Leur rôle est de permettre l'édition des modèles de simulation. Nous considérons que ces objets ne sont pas des ressources utilisables par les agents. En effet, ils ne font que modifier l'état des modèles de simulation. Toutefois, nous considérons qu'ils peuvent mettre en place ou à jour les systèmes multi-agents lorsque ces derniers n'existent pas ou ne contiennent pas les agents adéquats. Il s'agit en quelque sorte d'un outil d'administration du système multi-agents que nous proposons.

– **Modèles abstraits de simulation :**

Les modèles abstraits de simulation créés dans les outils de modélisation sont des ressources utilisables par les agents de vérification. Un modèle m est décrit par $\langle id, \mathcal{O}, \mathcal{S}, \mathcal{L}, \mathcal{A} \rangle$ où :

- id est l'identificateur unique du modèle ;
- $\mathcal{O}$ est l'ensemble des définitions des artefacts de modélisation composant le modèle (à l'exception des sous-modèles) ;
- $\mathcal{S}$ est un ensemble de sous-modèles composant le modèle. Leurs définitions respectent la même forme que celle de m ;
- $\mathcal{L}$ est l'ensemble des liens composant le modèle. Un lien est défini par $\langle a, b \rangle$ tel que $(a \in \mathcal{O} \vee a \in \mathcal{S})$ et $(b \in \mathcal{O} \vee b \in \mathcal{S})$, et correspond à un lien partant de a pour aboutir à b ;
- $\mathcal{A}$ est l'ensemble des attributs du modèle. Un attribut est défini par $\langle id, valeur \rangle$ où id est l'identificateur unique (dans le modèle m) de l'attribut et $valeur$ est la valeur de l'attribut.

Un artefact de modélisation est défini par $\langle id, \mathcal{C}, \mathcal{D}, \mathcal{A} \rangle$ où :

- id est l'identificateur de l'objet ;
- $\mathcal{C}$ est l'ensemble des contraintes statiques liées à l'élément de modélisation. Une contrainte statique est une expression en logique des prédicats du premier ordre qui est obtenue à partir du métamodèle UML ;
- $\mathcal{D}$ est l'ensemble des contraintes à respecter lors de l'utilisation distante de l'artefact. Ces expressions sont spécifiées dans le métamodèle UML ;
- $\mathcal{A}$ est l'ensemble des attributs associés à l'objet. Nous considérons que les attributs suivant sont toujours présents :
  - *sharable* indique si cette définition de l'objet peut être utilisée dans un modèle distant ;
  - *remote* indique si l'objet est une représentation locale d'une définition distante ;
  - *types* est un ensemble contenant les identificateurs des classes d'objets issues du métamodèle UML et dont l'instance *obj* hérite.

  Les attributs *sharable* et *remote* sont mutuellement exclusifs : $\forall o, \mathrm{val}_o(sharable) \Leftrightarrow \neg\, \mathrm{val}_o(remote)$.

9.4.2.4 Définition d'un agent de vérification

Les agents de vérification sont définis par $\langle id, m, \mathcal{E}, \mathcal{R} \rangle$ où :

- id est l'identificateur de l'agent ;
- m est le modèle abstrait de simulation sur lequel l'agent va réaliser les vérifications ;
- $\mathcal{E}$ est l'ensemble des conversations de l'agent en cours de déroulement ;
- $\mathcal{R}$ est l'ensemble des objets de m sur lequel travaille l'agent ($\mathcal{R} \subseteq \mathcal{O}_m$).

Chaque fois que l'état d'un objet o appartenant à $\mathcal{R}$ change, l'agent met en œuvre le processus de vérification. Ce processus est composé des étapes suivantes :

1. **Identification de l'agent de vérification connaissant la définition de $\vec{o}$:**
   L'agent met en œuvre le processus d'identification d'un agent en engageant une discussion avec son agent facilitateur. Nous présentons ce protocole dans le chapitre consacré à la conception. Toutefois, nous pouvons dire que la formule suivante est vérifiée si un agent connaissant $\vec{o}$ a été trouvé :

   $$\exists a, id_a = id_o \wedge o \neq d \wedge \text{owner}_o \neq \text{owner}_a \wedge \text{val}_a(sharable) \wedge \text{val}_o(remote) \wedge a = \vec{o}$$

   où a et o sont des agents.

2. **Demande des contraintes et des attributs de $\vec{o}$:**
   Cette étape consiste à engager des conversations utilisant les protocoles présentés dans la section 9.4.2.5.

3. **Réalisation de la vérification :**
   L'agent vérifie la propriété suivante sur le modèle m :

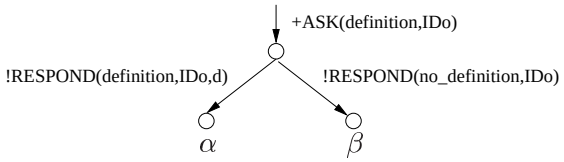
   $$\forall o \in \mathcal{O}_m, \forall c \in \mathcal{C}_o, \forall a \in \mathcal{A}_{\vec{o}}, \text{eval}_o(c[\text{val}_{\vec{o}}(a)/\,\text{id}_a])$$

4. **génération d'un avertissement si la vérification a échouée :**
   Cette étape consiste pour l'agent à générer des erreurs lorsque la formule précédente n'est pas une tautologie.

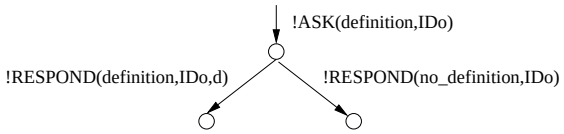
9.4.2.5 Définition des interactions

Nous considérons que les interactions d'un agent de simulation sont limitées à l'utilisation de deux protocoles. Le premier est utilisé par un agent pour récupérer les informations nécessaires à son objectif de vérification. Le second est le protocole servant à répondre à une demande d'informations de la part d'un autre agent.

Un protocole défini les étapes du déroulement d'une conversation entre agents. Nous utilisons la notation des diagrammes états-transitions pour représenter les protocoles. La figure 9.11(a) illustre le protocole de demande d'informations (c-à-d d'une définition d'un objet) par un agent. Ce protocole est mis en œuvre durant l'étape 2 du déroulement de l'exécution de l'agent. Les étiquettes des transitions correspondent aux messages envoyés (préfixés par +) ou reçus (préfixés par !). L'état α correspond à un succès de la récupération de la définition distante, alors que dans l'état β, l'agent génère une erreur. La figure 9.11(b) illustre le protocole utilisé par un agent pour répondre à une demande d'envoi de définition.



(a) Demande d'une définition distante



(b) Fourniture d'une définition distante

FIG. 9.11 – Protocoles pour la vérification

L'utilisation d'un protocole par agent est réalisée par l'intermédiaire des conversations (*cf.* $\mathcal{E}$ d'un agent). Une conversation est définie par $\langle id, a, p, state, \mathcal{A} \rangle$ où :
- id est l'identificateur de la conversation ;
- a est l'agent avec lequel la conversation est en cours ;
- p est le protocole utilisé pour gérer cette conversation ;
- $state$ est l'état dans lequel se trouve le protocole p ;
- $\mathcal{A}$ est l'ensemble des attributs créés durant la conversation. Ces attributs sont utilisés pour stocker temporairement des informations nécessaires au déroulement du protocole.

Ce système multi-agents de vérification à été très partiellement implanté dans notre environnement de simulation. Son développement complet reste une perspective.

9.5 Conclusion

La spécification est la première phase du cycle de vie des modèles dans laquelle nous introduisons de nouveaux concepts. L'objectif de cette étape est de produire un modèle abstrait du système industriel modélisé. Ce modèle abstrait a la particularité d'avoir une représentation suffisamment générique pour qu'elle puisse être adaptée à tout outil ou à toute méthodologie de simulation. Dans la suite de ce mémoire, nous verrons comment nous traduisons ce modèle abstrait en un modèle multi-agents.
La spécification propose un ensemble d'artefacts de modélisation qui a été décrit par l'intermédiaire d'un métamodèle UML. Nous avons décidé de redéfinir par ce biais les éléments de modélisation afin de nous permettre une traduction plus aisée par la suite. En effet, la définition des artefacts est réalisée selon un langage orienté objet. Grâce à cette modélisation explicite de l'ensemble des liens et des interactions de ces

éléments de modélisation, il nous sera possible d'écrire des règles de mécanisation de la traduction du modèle abstrait vers un modèle de simulation plus concret.
D'autre part, nous proposons une architecture permettant de vérifier la cohérence des modèles distribués. En effet, le métamodèle Uml n'est pas suffisant pour réaliser cette vérification. Il est nécessaire de mettre en place une société d'agents capables d'échanger les informations partagées par plusieurs modèles et de valider chaque contrainte de modélisation via un mécanisme d'évaluation d'expressions exprimées en logique des prédicats du premier ordre.
Dans le chapitre suivant, nous présentons la conception. Il s'agit de la phase durant laquelle le modèle abstrait est transformé en un modèle multi-agents.

Chapitre 10

Conception

La conception est la seconde phase pour laquelle nous apportons des modifications par rapport à une approche « classique » du cycle de vie des modèles de simulation. Elle permet de construire un modèle multi-agents à partir du modèle abstrait issu de la phase de spécification. Dans ce chapitre, nous présentons l'architecture du système multi-agents capable de supporter la simulation de systèmes industriels distribués. Enfin, avant de conclure ce chapitre, nous décrivons quelques-unes des règles de transformation utilisées pour construire le modèle conceptuel à partie du modèle abstrait.

10.1 Architecture du système multi-agents

Dans cette section, nous définissons l'architecture du SMA qui servira de base au modèle conceptuel. Nous commençons par présenter l'architecture générale, puis nous décrivons les deux types d'agents composant le SMA : les agents facilitateurs et les agents pour la simulation.

10.1.1 Structure générale

Afin de permettre la mise en place d'un processus de simulation par l'intermédiaire de sociétés d'agents, nous proposons une infrastructure illustrée par la figure 10.1. Elle est composée principalement par des sociétés d'agents interconnectés. Nous avons décidé de diviser le SMA en un ensemble de systèmes plus petits. Ainsi, chaque modèle composant le modèle abstrait (c-à-d chaque modèle distribué) est représenté par un seul de ces sous-systèmes multi-agents. Ce choix a été guidé par une raison de modularité et de dynamisme. Chaque modèle de simulation distribuée étant partiellement autonome vis-à-vis des autres modèles, il nous semble que ces sous-systèmes multi-agents sont les SMA les plus grands dont on puisse être certain de la définition. D'autre part cette décomposition engendre un support intéressant de la modularité.

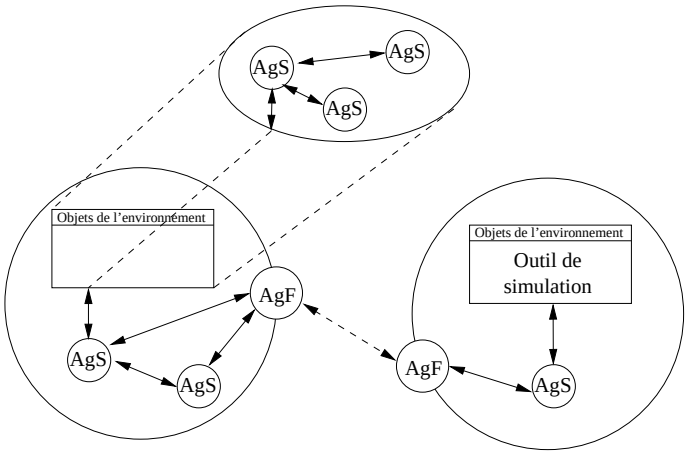


FIG. 10.1 – Infrastructure SMA pour la simulation

En effet, lorsqu'un membre du réseau d'entreprise veut quitter le groupement, du point de vue de la conception, il n'y a qu'a retirer le sous-système multi-agents correspondant. Cette décomposition est d'autant plus utile qu'elle facilite la gestion dynamique des modèles de simulation. En effet, retrouver le bon modèle distant, ou le bon objet distant est plus aisé avec cette décomposition.

La figure 10.1 illustre aussi la présence de deux grandes classes d'agents :

- Les agents facilitateurs (AGF) ont pour rôle de faciliter la transmission des messages entre les agents devant réaliser le processus de simulation (AGS). Ainsi les agents facilitateurs sont des intermédiaires obligatoires pour réaliser les interactions entre les différentes sociétés d'agents. D'autre part, l'ensemble des agents facilitateurs maintient une base de connaissances contenant les descriptions partielles des ressources et des services du SMA (ressources dans le modèle de simulation, noms de modèles distants, ...). Les AGF sont chargés de router les messages afin qu'ils atteignent au moins l'un des agents gérant le service correspondant (demandé dans le message).
- Les agents pour la simulation (AGS) composent les sociétés d'agents mettant en œuvre le processus de simulation. Nous proposons une architecture permettant à l'agent de communiquer avec les agents facilitateurs, ainsi qu'avec les autres éléments de son environnement (agents et objets de l'Environnement).

Les agents facilitateurs sont les seuls agents définis statiquement au sein de $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$. Leur architecture et leurs interactions sont définies dans notre approche méthodologique et ne peuvent faire l'objet d'une éventuelle extension de la part d'objets appartenant à un modèle de simulation. Ces derniers peuvent influencer la conception des agents pour la simulation.

Enfin, la figure 10.1 illustre aussi un autre aspect important de notre approche : la récursivité de l'architecture. En effet, chaque agent ou objet de l'environnement

peut être à son tour un système multi-agents de niveau inférieur. Dans ce cas, il peut soit s'agit d'un ensemble d'agents respectant les spécifications d'interaction de $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$, soit d'une société d'agents interagissant par l'intermédiaire d'un AgF.
À partir de cette description, nous définissons le modèle conceptuel comme un système multi-agents décrit par le n-uplet $\langle id, \mathcal{S}, \mathcal{A} \rangle$ où :
– id est l'identificateur du modèle conceptuel ;
– $\mathcal{S}$ est l'ensemble des sous-systèmes multi-agents composant le modèle conceptuel ;
– $\mathcal{A}$ est l'ensemble des attributs associés au modèle (nom, description, ...).
Chaque sous-système multi-agents est décrit par $\langle id, facilitateur, \mathcal{O}, \mathcal{A}, \mathcal{T} \rangle$ où :
– id est l'identificateur de la société d'agents ;
– $facilitateur$ est l'agent facilitateur décrit dans la section suivante ;
– $\mathcal{O}$ est l'ensemble des ressources composant l'environnement du sma ;
– $\mathcal{A}$ est l'ensemble des agents pour la simulation ;
– $\mathcal{T}$ est l'ensemble des attributs associés à la société.
Nous présentons dans les sections suivantes les deux classes d'agents dégagées.

10.1.2 Agent facilitateur

Un agent facilitateur (ou AgF) est un vecteur aidant à la communication entre les agents pour la simulation. Il permet une modularité plus aisée des différentes composantes de la société d'agents.
Nous définissons un agents facilitateur selon trois des facettes de l'approche « Voyelles » [Demazeau, 1995].

10.1.2.1 Facette « Agent » de l'AGF

Un AgF est composé d'un module comportemental lui permettant de jouer le rôle de passerelle ou de facilitateur. Nous considérons que les agents facilitateurs sont accueillants au sens de [Vercouter, 2000] vis-à-vis d'autres AgF. La figure 10.2 illustre cette architecture :

– **Connaissances :**
  Il s'agit de la base de connaissances nécessaire aux modules de raisonnement de l'agent : services proposés par les agents pour la simulation (AgS), sémantiques d'une action ou d'un plan [Vercouter, 2000], ...

– **Connaissances d'accueil :**
  C'est l'ensemble des connaissances nécessaires au comportement accueillant de l'agent. Ces connaissances sont divisées en deux parties : la représentation des autres AgF, et les connaissances communes à tous les agents [Vercouter, 2000].

– **Comportement accueillant :**

Ce module permet à l'agent de mettre en œuvre un comportement accueillant vis-à-vis des autres facilitateurs. Lorsqu'un AGF veut entrer dans le système, il se présente aux autres facilitateurs en utilisant la méthode proposée par Laurent VERCOUTER (pas par « broadcast », mais par présentations successives). Une fois introduit dans la société, le nouvel agent facilitateur maintient la cohérence de ses connaissances sur les services proposés par les agents de simulation qui lui sont associés, et sur ses connaissances sur les autres AGF.

– **Comportement de passerelle :**

Ce module permet à l'agent de servir de passerelle entre différentes sociétés d'agents. Il utilise la partie *représentation des autres* des connaissances d'accueil. Lorsque l'agent reçoit une demande de transmission de message de la part d'un agent de simulation, il recherche l'agent facilitateur satisfaisant les besoins (c-à-d proposant les services demandés par l'AGS), et transmet le message.

– **Actions & interactions :**

Il s'agit du module qui permet de mettre en œuvre les actions et les interactions décidées par le module de raisonnement.

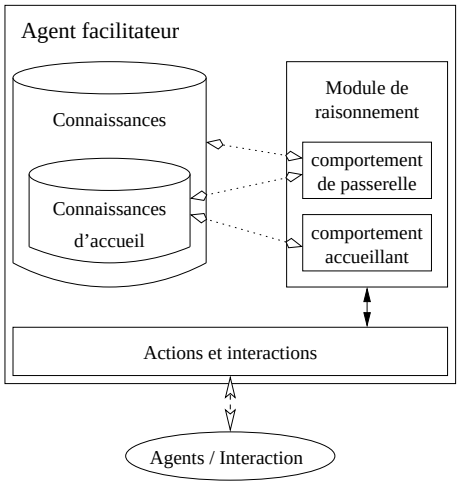


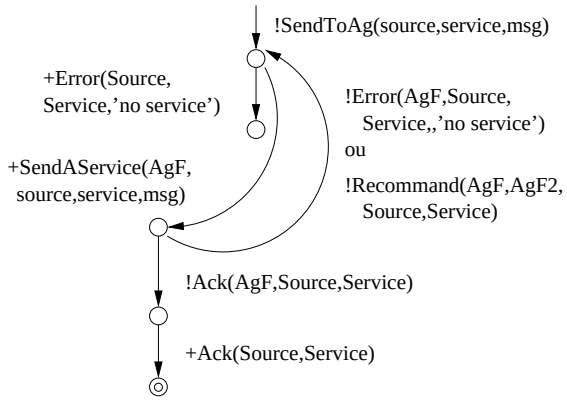
FIG. 10.2 – Architecture d'un agent facilitateur

Nous définissons un agent facilitateur comme le n-uplet $\langle id, \mathcal{G}, \mathcal{S}, \mathcal{F}, \mathcal{A} \rangle$ où :
– id est l'identificateur de l'agent ;
– $\mathcal{G}$ est l'ensemble des agents pour la simulation enregistrés dans la société gérée par l'AGF ;
– $\mathcal{S}$ l'ensemble des services proposés par les agents de simulation gérés par l'agent facilitateur. Chaque service est défini par $\langle id, agent \rangle$ où id est l'identificateur du service et $agent$ est l'identificateur de l'agent fournissant le service ;
– $\mathcal{F}$ est l'ensemble des services proposés par d'autres agents facilitateurs ;

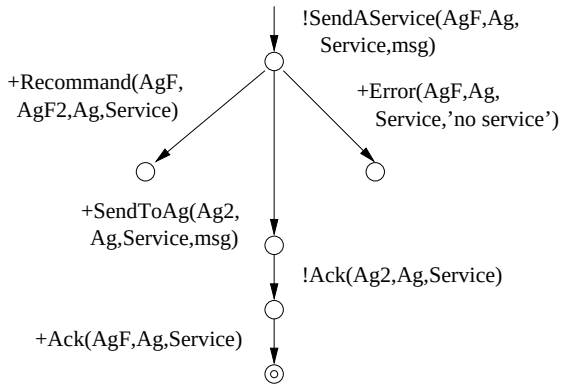
– $\mathcal{A}$ est l'ensemble des attributs associés à l'agent facilitateur.

10.1.2.2 Facette « Interaction » de l'AGF

L'agent facilitateur possède deux grandes classes d'interactions : les interactions avec les autres agents facilitateurs et les interactions avec les agents pour la simulation. Chacune de ces classes possède un ensemble de protocoles répondant à des objectifs particuliers.



(a) Demande de service par un AgS



(b) Demande de service par un AgF

Fig. 10.3 – Protocoles pour un AgF

– **Entre agents facilitateurs :**

Les AgF ont des relations très fortes avec les autres facilitateurs. Rappelons que les AgF sont des agents accueillants. Cette caractéristique d'ouverture nous permet d'avoir une société d'agents facilitateurs cohérente. En effet, chaque AgF possède des relations d'accointance avec les autres AgF pouvant répondre à ses besoins. Nous utilisons la définition proposée par Laurent Ver-

COUTER [VERCOUTER, 2000] qui décrit toutes les caractéristiques essentielles que doivent posséder les agents accueillants et un SMA ouvert et distribué. Nous considérons que l'arrivée et le départ d'AGF, et le maintient des bases de connaissances à propos des autres AGF sont réalisés correctement.

Ainsi, chaque fois qu'un agent facilitateur a besoin d'une compétence particulière, et qu'il ne la trouve pas dans sa base courante, il réalise une recherche d'un AGF proposant le service recherché. Cette recherche est réalisée par l'intermédiaire de la méthode d'accueil et de présentation proposée par [VERCOUTER, 2000]. De manière schématique, elle se déroule suivant les étapes suivantes. Un agent A_0 recherchant un service :

1. présente ses besoins à un autre agent A_1 ;
2. si A_1 possède les services demandés, l'algorithme s'arrête sur un succès ;
3. si A_1 connait un autre agent A_2 pouvant répondre à la demande de A_0, il recommande ce dernier à A_2, puis reprise de l'algorithme avec A_2 à partir de l'étape 1 ;
4. si A_1 n'a aucune connaissance sur le service demandé, A_0 reprend l'algorithme à la première étape en choisissant un autre agent pour se présenter. Ce choix est réalisé à partir de ses pages blanches et jaunes construites grâce au mécanisme d'accueil et de présentation utilisé lors de l'arrivée d'un nouvel agent.

– **Avec des agents pour la simulation :**

Il existe trois protocoles utilisés par un AGF pour interagir avec les agents pour la simulation : le protocole de recherche de service, le protocole de demande de service par un AGS, et le protocole de demande de service envoyé par un AGF.

1. Le *protocole de recherche de service* est initié par un AGS via l'envoi d'un message `WhichForService`. Dans ce cas, et si le service n'est pas connu de l'agent facilitateur, ce dernier met en œuvre la méthode de présentation utilisée lors de l'arrivée de l'AGF dans la société [VERCOUTER, 2000].
2. Le *protocole de demande de service par un* AGS est lui aussi initié par un agent pour la simulation.

   Il correspond à l'envoi d'un message par un AGS vers un autre AGS. La figure 10.3(a) représente ce protocole. Les étiquettes des transitions sont les messages envoyés (préfixés par `+`) ou les messages reçus (préfixés par `!`). La définition de la sémantique exacte de ces messages est présentée ci-dessous. De manière schématique, ce protocole consiste à trouver l'AGF associé à un AGS pouvant répondre au service demandé, puis de transmettre ce message.

3. Le *protocole de demande de service par un* AgF est initié par un AgF qui désire envoyer un message à un agent pour la simulation associé au facilitateur. La figure 10.3(b) illustre ce protocole.

Ces protocoles utilisent des messages qui peuvent être formalisés ainsi :
- `SendToAg(from,serv,paramètres)` est une demande du service `serv` par l'agent `from`. Les `paramètres` sont des données dépendantes du service demandé.
- `SendToAg(to,from,serv,paramètres)` est une demande du service `s` par l'agent `from` à l'agent `to`.
- `Error(from/to,[ag,]serv,message)` est une notification d'erreur envoyée par `from` ou reçue par `to` à propos du service `serv` initialement demandé par `ag`.
- `Ack(from/to,[ag,]serv)` est une notification de réussite envoyée par `from` ou reçue par `to` à propos du service `serv` initialement demandé par `ag`.
- `SendAservice(from/to,ag,serv,paramètres)` est la demande d'un service envoyée par un agent facilitateur. `from/to` est l'émetteur ou le récepteur du message. `serv` représente le service demandé par l'agent de simulation `ag`.
- `Recommand(from/to,facilitator,ag,serv)` est la recommandation envoyée ou reçue par `from/to` auprès de l'agent facilitateur `facilitator` pour le service `serv` initialement demandé par l'agent `ag`.

10.1.2.3 Facette « Organisation » de l'AGF

Les agents facilitateurs jouent le rôle de passerelle entre les différentes sociétés d'agents. Ils possèdent des liens d'accointance avec les autres agents facilitateurs et des liens d'autorité avec les agents composant la société à laquelle est associé l'AgF. L'expression d'une telle organisation avec Moise ne pose aucun problème [Hannoun et al., 2000 ; Hannoun, 2002].

10.1.3 Agent pour la simulation

Cette catégorie d'agents n'est pas entièrement spécifiée par $\mathcal{MAM}$A-S. Nous ne proposons qu'une architecture partielle de la facette interaction qui permettra aux agents de communiquer avec les agents facilitateurs et les objets de l'environnement. La spécification complète d'un agent pour la simulation est obtenue à partir de cette architecture de base et d'une traduction des comportements inscrits dans le modèle abstrait. Par exemple, les centres de décision du modèle abstrait peuvent être traduits en un agent dont le comportement sera obtenu à partir du modèle comportemental abstrait.
Nous considérons que l'agent pour simulation est construit autour de deux niveaux d'interactions :
- le premier est le niveau le plus bas, il implante les mécanismes d'interaction avec les agents facilitateurs. Ainsi, chaque fois qu'un agent de simulation reçoit un

message `SendToAg`, il doit renvoyer un message d'acquittement `Ack`.
- Le second niveau est basé sur le premier. Il implante les protocoles d'interaction propres aux objectifs de l'agent. Par exemple, un agent représentant un centre de décision pourra implanter ses propres protocoles de négociation. Ces protocoles sont déduits du modèle abstrait. Ils sont donc connus a priori par tous les agents susceptibles d'utiliser ces protocoles (pas de dynamique d'apprentissage des protocoles). Nous verrons dans la dernière section de ce chapitre comment transformer le modèle abstrait en un ensemble d'agents.

Nous considérons que les agents pour la simulation sont définis par $\langle id, \mathcal{S}, \mathcal{O}, \mathcal{R}, \mathcal{A}, \mathcal{C}, \mathcal{P} \rangle$ où :
- id est l'identificateur de l'agent ;
- $\mathcal{S}$ est l'ensemble des services supportés par l'agent et qui ne sont pas proposés par les objets appartenant à $\mathcal{O}$;
- $\mathcal{O}$ est l'ensemble des objets de l'environnement directement gérés par cet agent ;
- $\mathcal{R}$ est l'ensemble des services proposés par l'agent (internes à ce dernier ou supportés par les objets de l'environnement) ;
- $\mathcal{A}$ est l'ensemble des attributs associés à l'agent ;
- $\mathcal{C}$ est la description du comportement de l'agent. Cette description doit respecter les contraintes d'interface imposées par les autres membres de ce n-uplet ;
- $\mathcal{P}$ est l'ensemble des protocoles de communication utilisés par l'agent.

Ainsi, l'ensemble $\mathcal{R}$ est construit à partir des objets appartenant à $\mathcal{S}$ et à $\mathcal{O}$.

10.1.4 Objets de l'environnement

Les objets de l'environnement sont tous les objets nécessaires au processus de simulation, et qui ne sont pas des agents. Par exemple, une ligne de production, modélisée au sein d'un outil de simulation comme ARENA®, peut être considérée comme un objet de l'environnement.

La seule contrainte sur les objets est qu'ils proposent un ensemble de services et qu'ils peuvent générer un ensemble d'évènements. Nous pouvons faire une analogie avec la programmation orientée évènements où les méthodes servent à accéder aux services d'un objet, et où les messages permettent à ces derniers de générer des évènements concernant leurs changements d'états.

Ainsi, nous définissons un objet comme un n-uplet composé de $\langle id, \mathcal{S}, \mathcal{A} \rangle$ où :
- id est l'identificateur de l'objet ;
- $\mathcal{S}$ est l'ensemble des services proposés par l'objet ;
- $\mathcal{A}$ est l'ensemble des attributs associés à l'objet.

Nous considérons qu'un service peut être une méthode ou un évènement. Ainsi un service est défini par $\langle id, type, \mathcal{P} \rangle$ où :
- id est l'identificateur du service ;

– *type* est le type de l'événement. Il prend pour valeur *method* s'il s'agit d'un service à appeler, et *event* si c'est l'objet lui même qui génère l'appel au service ;
– P est un ensemble définissant les paramètres du service. Un paramètre est un triplet définissant le nom, le type et la valeur d'un paramètre.

D'autre part, les attributs d'un objet (ensemble $\mathcal{A}$) sont considérés comme des constantes. Ils ne peuvent être modifiés que par l'intermédiaire d'appels de services de type `method`. De plus, si l'objet est issu d'une traduction à partir du modèle abstrait, ses attributs correspondront à l'ensemble des attributs associés à l'objet abstrait dans le métamodèle Uml.

10.1.5 Quelques propriétés

Dans cette section, nous allons présenter quelques-unes des propriétés de notre architecture qui n'ont pas pu être présentées plus tôt, faute d'avoir défini toutes les composantes du système. Dans cette section, nous utilisons les notations présentées dans la section 9.4.2.2.

Propriété 10.1 :

Un identificateur est unique dans son contexte. Soit $\mathcal{E}$ un ensemble contenant des n-uplets dont l'un des membres est id :

$$\forall a \in \mathcal{E}, \forall b \in \mathcal{E} / \mathrm{id}_a = \mathrm{id}_b \Rightarrow a = b$$

Propriété 10.2 :

Deux identificateurs de service identiques doivent fournir exactement le même service (aux valeurs des paramètres près). Soit s et s' deux services définis dans le sma *:*

$$\forall s, \forall s' / \begin{pmatrix} type_s = type_{s'} \wedge \\ \mathcal{P}_s \equiv \mathcal{P}_{s'} \end{pmatrix} \Rightarrow \mathrm{id}_s = \mathrm{id}_{s'}$$

où l'équivalence entre deux ensembles de paramètres, noté $E_1 \equiv E_2$, est défini par :

$$\forall a \in E_1, \forall b \in E_2, type_a = type_b \Rightarrow \mathrm{id}_a = \mathrm{id}_b$$

Propriété 10.3 :

Les services internes à un agent pour la simulation ne doivent pas être déjà proposés par un objet de l'environnement qu'il utilise. Soit a un agent pour la simulation :

$$\forall s \in \mathcal{S}_a, \forall o \in \mathcal{O}_a, \forall s' \in \mathcal{S}_o / \mathrm{id}_s = \mathrm{id}_{s'} \Rightarrow s = s'$$

PROPRIÉTÉ 10.4 :

Les services proposés par un agent pour la simulation sont un sous-ensemble des services contenus dans sa définition. Soit a un agent pour la simulation :

$$\mathcal{R}_a \subseteq \mathcal{I}_1 \cup \mathcal{I}_2$$

où $\mathcal{I}_1 \triangleq \{\mathrm{id}_x / x \in \mathcal{S}_a\}$ *et* $\mathcal{I}_2 \triangleq \{\mathrm{id}_x / \forall o \in \mathcal{O}_a, x \in \mathcal{S}_o\}$.

PROPRIÉTÉ 10.5 :

*Les services supportés par un agent facilitateur correspondent à l'ensemble des services exportables des agents pour la simulation dans la société que l'*AGF *gère. Soit f un agent facilitateur :*

$$\forall g \in \mathcal{G}_f, \forall s \in \mathcal{R}_g / \langle \mathrm{id}_s, \mathrm{id}_g \rangle \in \mathcal{S}_f$$

PROPRIÉTÉ 10.6 :

Pour un agent facilitateur a, les agents contenus dans l'ensemble des services des autres AGF *doivent être des agents différents de a :*

$$\forall \langle service, agent \rangle \in \mathcal{F}_a / \mathrm{id}_a \neq agent$$

PROPRIÉTÉ 10.7 :

Un agent facilitateur possède une base de connaissances cohérente sur les autres AGF. *Soit f et f' deux agents facilitateurs :*

$$\forall \langle service, agent \rangle \in \mathcal{F}_f, \exists f' / \mathrm{id}_f \neq \mathrm{id}_{f'} \wedge \mathrm{id}_{f'} = agent \wedge service \in \{x / \forall y, \langle x, y \rangle \in \mathcal{S}_{f'}\}$$

PROPRIÉTÉ 10.8 :

Tous les agents pour la simulation attachés à un agent facilitateur (ensemble $\mathcal{G}$) doivent être présents dans la définition de la sous-société d'agents. Soit s une sous-société d'agents :

$$\forall a \in \mathcal{G}_{facilitator_s} / a \in \mathcal{A}_s$$

PROPRIÉTÉ 10.9 :

Les objets référencés par les agents de simulation doivent être définis dans le modèle de sous-système multi-agents. Soit s une sous-société d'agents :

$$\forall a \in \mathcal{A}_s, \forall o \in \mathcal{O}_a / o \in \mathcal{O}_s$$

PROPRIÉTÉ 10.10 :

Tous les agents facilitateurs doivent avoir un identificateur unique. Soit m le modèle conceptuel :

$$\forall (a, b) \in \mathcal{S}_m^2, \mathrm{id}_{facilitator_a} \neq_i d facilitator_b$$

Propriété 10.11 :

Tous les agents facilitateurs référencés dans la base de connaissance d'un AGF *doivent exister dans le même système multi-agents. Soit m un modèle conceptuel :*

$$\forall a \in \mathcal{AF}, \forall b \in \{x/\forall \alpha, \langle \alpha, x \rangle \in \mathcal{F}_a\}/b \in \mathcal{AF}$$

avec $\mathcal{AF} \triangleq \{facilitator_x/x \in \mathcal{S}_m\}$.

Dans la section suivante, nous présentons quelques règle pour traduire un modèle abstrait en un modèle conceptuel.

10.2 Règles de traduction

Afin de faciliter la création du modèle du système multi-agents utilisant l'architecture décrite ci-dessus, $\mathcal{M}_{\mathcal{A}}\mathcal{MA}\text{-}\mathcal{S}$ propose un ensemble de règles méthodologiques qui, à partir du modèle abstrait de simulation, permettent d'obtenir un modèle multi-agents partiel. Ces règles ne sont pas exhaustives. Dans ce document, nous présentons les plus évidentes.

10.2.1 Syntaxe et sémantique

Les règles de traduction respectent la syntaxe exprimée par la grammaire BNF suivante :

```
rules ::= rule rules |vide

rule ::= "AGENT" aeio_rule_part
         |
         "ENVIRONMENT" rule_part
         |
         "INTERACTION" rule_part
         |
         "ORGANIZATION" rule_part

rule_part ::= [ '[' nom_objet_concret ']' ]  '←'
              nom_object_abstrait '=' description ';'

aeio_rule_part ::= [ '[' nom_agent ']' ]  '←'
                   nom_object_abstrait '=' aeio_description ';'

description ::= '{' pseudocode '}'
```

```
aeio_description ::= 'A' '{' pseudocode '}' '∧'
                     'E' '{' pseudocode '}' '∧'
                     'I' '{' pseudocode '}' '∧'
                     'O' '{' pseudocode '}' '∧'
                     'X' '{' pseudocode '}'
                     |
                     description

nom_objet_concret ::= chaîne
                      |
                      '_'

nom_objet_abstrait ::= chaîne
```

Elle décrit la règle permettant de créer un objet d'une des facettes du SMA conceptuel nommé `nom_objet_concret` à partir d'un objet issu du modèle abstrait de simulation `nom_objet_abstrait`) (si `nom_objet_concret` est égal à "_", alors le nom de l'objet concret est identique au nom de l'objet abstrait). La description est basée sur l'approche AEIO [DEMAZEAU, 1995], qui consiste à décrire au moyen de pseudocode le contenu de chaque facette composant l'élément. Elle peut être aussi directement basée sur du speudocode. Les pseudocodes utilisés pour chacune des facettes peuvent être différents. Par exemple, la facette Organisation peut être décrite à l'aide de MOISE [HANNOUN et al., 2000], et la facette Interaction avec FIPA-ACL [FIPA, 1998]. D'autre part, si plusieurs règles font référence au même `nom_objet_abstrait`, alors elles sont toutes appliquées et cela dans un ordre indéterminé.

$\mathcal{MAMA}$-$\mathcal{S}$ propose un ensemble de règles simples qui permettent de construire rapidement un squelette de modèle multi-agents. Toutefois, elles ne sont pas exhaustives et doivent faire l'objet d'une étude plus précise. De plus, elles doivent être adaptées par l'utilisateur de la méthodologie pour mieux adhérer aux besoins réels de la simulation.

10.2.2 Quelques règles

Dans cette section, nous présentons quelques règles de traduction du modèle abstrait pour obtenir le modèle conceptuel. Pour cela, nous proposons quelques règles pour chacuns des sous-systèmes du système industriel modélisé. L'ensemble des règles que nous avons dégagé est dans l'annexe C.

De manière générale, nous nous déchargeons d'un grand nombre de tâches sur les outils de simulation. Ainsi, nous considérons que seuls les centres de décision, les artefacts distribuables et les artefacts de support de la distribution doivent faire l'objet d'une création d'un agent. Les autres peuvent être déposés dans l'environnement et

feront l'objet d'une modélisation et d'une exécution au sein d'un outil existant de simulation. Par exemple, le modèle abstrait du sous-système opérationnel peut être presque entièrement géré par un outil comme Arena®. Ce dernier propose en effet des artefacts de modélisation compatibles avec nos propres artefacts. Nous décrivons dans le chapitre suivant comment choisir un outil de simulation, mais nous supposons dès maintenant qu'ils peuvent être utilisés dans le processus de simulation (contrainte de réalisation posée dans le chapitre 2.3.4).

10.2.2.1 Sous-système opérationnel

Des exemples de règles (exprimés à l'aide de la grammaire définie ci-dessus) que nous utilisons pour créer un modèle conceptuel du sous-système opérationnel sont :

Règle 10.1 :

Nous considérons que les unités de traitement du sous-système opérationnel peut être entièrement généré dans l'environnement SMA.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{ProcessingUnit} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \\ \left\{ \begin{array}{c} \langle \texttt{getState}, method, \emptyset \rangle, \\ \langle \texttt{beginProcess}, event, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \\ \langle \texttt{endProcess}, event, \{\langle \texttt{entité}, \texttt{entité}, \gamma \rangle\} \rangle, \end{array} \right\}, \\ \texttt{\_.getAttributes()} \cup \\ \left\{ \begin{array}{l} \langle \texttt{delay}, Law, \texttt{"Law\_for\_"+\_.name} \rangle, \\ \langle \texttt{fail}, Law, \texttt{"Fail\_Law\_for\_"+\_.name} \rangle \end{array} \right\} \end{array} \right\rangle \right\}$$

La règle 10.1 permet de créer un objet de l'environnement pour chaque unité de traitement appartenant au modèle abstrait du sous-système opérationnel. Nous avons utilisé un pseudocode basé sur la notation décrite dans le début de ce chapitre (notation utilisée pour définir la notion d'objet de l'environnement). Ainsi, pour chaque objet « `ProcessingUnit` » du modèle conceptuel, un objet conceptuel est créé. Il propose trois services qui sont l'interrogation sur l'état de l'unité (en général occupée, libre, ou en panne), et deux événements générés lorsqu'un lot de fabrication entre ou sort de l'unité. Les attributs de l'objet de l'environnement correspondent aux attributs de l'objet du métamodèle Uml. Notons que les lois statistiques utilisées pour déterminer les différents délais sont décrites dans les attributs hérités du métamodèle.

La base de la transformation du modèle abstrait du sous-système opérationnel est la suivante : nous considérons que ce sous-système peut être entièrement pris en charge

par les outils de simulation existants. Ainsi nous construisons une représentation du modèle abstrait dans l'environnement du système multi-agents.

RÈGLE 10.2 :

Un générateur d'entité fait l'objet d'une traduction sous forme d'un objet de l'environnement. Comme l'artefact de spécification est piloté par un centre opérationnel de décision, aucune loi stochastique n'est attachée à l'objet de l'environnement.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{EntityGenerator} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \\ \left\{ \langle \texttt{generate\_entity}, method, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \right\}, \\ \texttt{\_.getAttributes()} \end{array} \right\rangle \right\}$$

La règle 10.2 représente la transformation de l'artefact de spécification « Générateur d'entité » en un objet de l'environnement du SMA. Notons qu'aucune loi stochastique n'a été attachée à cet objet. Cela est dû au fait qu'un agent pilotera cet objet. Ainsi, la loi statistique est remplacée par un ensemble d'appels à la méthode `generate_entity`.

RÈGLE 10.3 :

Le point de sortie doit est transformé en un objet de l'environnement et en un agent pilotant ce dernier.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{OutgoingFlow} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \left\{ \langle \texttt{forward\_entity}, event, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \right\}, \\ \texttt{\_.getAttributes()} \end{array} \right\rangle \right\}$$

$$\texttt{AGENT} \leftarrow \texttt{OutgoingFlow} = \left\{ \langle \texttt{\_.name}, \emptyset, \{\texttt{\_.name}\}, \emptyset, \texttt{\_.getAttributes()}, \texttt{\_.getBehavior()} \rangle \right\}$$

10.2.2.2 Sous-système décisionnel

Le modèle abstrait du sous-système décisionnel est pour l'essentiel transformé en une société d'agents.

Règle 10.4 :

Chaque centre de prise de décision est traduit en un agent de simulation ayant le même comportement.

```
AGENT [ OperationalCenter ]← OperationalCenter =
  A {
    IF OperationalCenter.behavior INSTANCEOF
       AlgorithmicBehavior THEN
      Behaviour = OperationalCenter.behavior.toPseudoCode
    END IF
  } ∧
  E {
    IF OperationalCenter.behavior INSTANCEOF
       ReactiveBehavior THEN
       Behaviour = OperationalCenter.behavior.toPseudoCode
    END IF
  } ∧
  I {
    IF OperationalCenter.behavior INSTANCEOF ProtocolBehavior
    THEN
      Behaviour = OperationalCenter.behavior.toPseudoCode
    END IF
  } ∧
  O { } ∧
  X {
    S = {x / x ∈ S_OperationalCenter.PhysicalElement}
    ⟨OperationalCenter.name,
      S,
      { OperationalCenter.PhysicalElement },
      {service_x / x ∈ S},
      OperationalCenter.getAttributes()⟩
  }
}
```

Ainsi pour chaque centre de décision, nous créons un agent ayant le même comportement que l'objet du modèle abstrait. D'autre part, lorsqu'il s'agit d'un centre de décision de niveau opérationnel, les services proposés par l'agent sont ceux de l'objet de l'environnement correspondant à l'artefact de modélisation du sous-système

opérationnel.

10.2.2.3 Sous-système informationnel

Le sous-système informationnel est en général traduit sous forme d'objets dans l'environnement. Ces objets sont alors utilisés par les agents pour mettre en œuvre leur processus de raisonnement.

Par exemple, prenons le modèle des gammes. Il est transformé sous forme d'objets de l'environnement. En effet, nous considérons que les outils de simulation comme ARENA® gèrent naturellement la notion de gamme. Ne désirant pas programmer un nouvel outil de simulation, nous proposons d'utiliser ces fonctionnalités dont l'efficacité n'est pas à mettre en doute.

10.3 Exemple

Reprenons l'exemple du groupement d'entreprise présenté dans le chapitre précédent. Cependant, nous limiterons nos propos dans cette section à la modélisation de l'entreprise E_1. En effet, nous considérons que les modèles conceptuels de E_2 et de E_3 n'éclaireraient pas significativement nos propos.

Grâce à la phase de spécification, nous possédons un modèle abstrait de simulation pour chacun des membres du groupement. Par l'application des règles de transformation présentées ci-dessus, nous obtenons un squelette de système multi-agents capable de réaliser la simulation du système industriel distribué.

Le modèle conceptuel de ce groupement d'entreprise est défini par le n-uplet $\langle groupement, \{E_1, E_2, E_3\}, \emptyset\rangle$ (*cf.* définition page 155) où E_1, E_2 et E_3 sont les modèles des sous-systèmes multi-agents pour chacune des entreprises composant le groupement. Dans cette section, nous nous limiterons à la présentation du modèle conceptuel de l'entreprise E_1.

Pour traduire les modèles abstraits des trois sous-systèmes de E_1, nous proposons de procéder sous-système par sous-système. La figure 10.4 illustre la structure générale du modèle résultant du sous-système multi-agents pour E_1. Nous pouvons modéliser l'entreprise E_1 par le n-uplet $\langle E_1, AgF_{E_1}, \mathcal{O}_{E_1}, \mathcal{A}_{E_1}, \emptyset\rangle$ où $\mathcal{O}_{E_1}$ et $\mathcal{A}_{E_1}$ sont définis dans la suite de cette section et AgF_{E_1} est l'agent facilitateur pour E_1.

Le sous-système opérationnel est modélisé par l'intermédiaire de l'environnement. Parmis celles que nous utilisons, se trouvent les règles 10.2 (générateur d'entités), 10.1 (unité de traitement), et 10.3 (point de sortie). L'application de ces différentes règles nous permettent d'obtenir un ensemble d'objets appartenant à l'environnement. Par conséquent, nous obtenons un environnement défini comme suit ($\mathcal{A}_\alpha$ in-

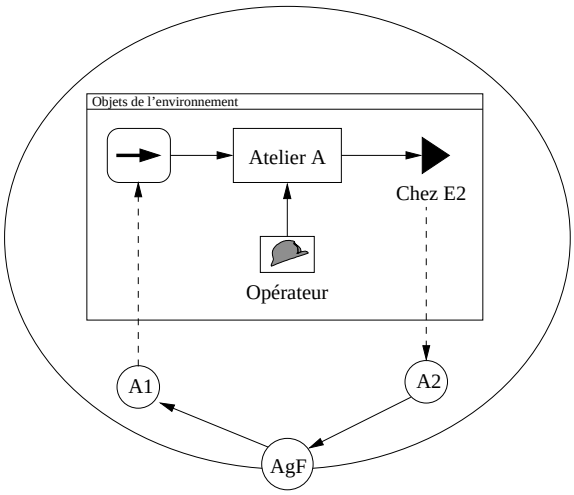


Fig. 10.4 – Exemple - Modèle conceptuel de E_1

dique un ensemble non détaillé de paramètres) :

$$\mathcal{O}_{E_1} = \left\{ \begin{array}{c} \langle \texttt{Atelier A}, \emptyset, \mathcal{A}_1 \rangle, \\ \langle \texttt{Opérateur}, \emptyset, \mathcal{A}_2 \rangle, \\ \langle \texttt{Lien 1}, \emptyset, \mathcal{A}_3 \rangle, \\ \langle \texttt{Lien 2}, \emptyset, \mathcal{A}_4 \rangle, \\ \langle \texttt{Lien 3}, \emptyset, \mathcal{A}_5 \rangle, \\ \left\langle \texttt{Chez } E_2, \left\{ \langle \texttt{forward\_entity}, event, \{ \langle \texttt{entité}, \text{entité}, \alpha \rangle \} \rangle \right\}, \mathcal{A}_6 \right\rangle, \\ \left\langle \texttt{Générateur}, \left\{ \langle \texttt{generate\_entity}, method, \{ \langle \texttt{entité}, \text{entité}, \alpha \rangle \} \rangle \right\}, \mathcal{A}_7 \right\rangle, \end{array} \right\}$$

D'une part, la règle 10.3 nous a non seulement permis de générer un objet dans l'environnement, mais elle introduit également un agent dont le rôle sera de transférer les entités physiques du modèle de E_1 vers celui de E_2. D'autre part, chaque élément du sous-système décisionnel est transformé en un agent. De plus, les modèles des gammes opératoires et des nomenclatures forment une partie de la base de connaissances de l'agent qui représentera la `gestion de production`. Ainsi, nous pouvons définir l'ensemble des agents composant le modèle conceptuel de E_1 :

$$\mathcal{A}_{E_1} = \left\{ \begin{array}{c} \left\langle A_1, \{ \langle ordre de fabrication, method, \{ \langle \texttt{taille OF}, \mathbb{N}^+, 1 \rangle \} \rangle \}, \right. \\ \left. \{\texttt{Générateur}\}, \{ordre de fabrication\}, \emptyset, \mathcal{C}_{A_1} \right\rangle, \\ \left\langle A_2, \emptyset, \{\texttt{Chez } E_2\}, \emptyset, \emptyset, \mathcal{C}_{A_2} \right\rangle \end{array} \right\}$$

Dans la définition précédente, $\mathcal{C}_{A_1}$ et $\mathcal{C}_{A_2}$ représentent respectivement les comportements internes des agents A_1 et A_2. Ces comportements sont directement issus de ceux décrits dans le modèle abstrait. Ne proposant pas encore de règles permettant de transformer automatiquement les comportements du modèle abstrait en com-

portements pour les agents, il nous faut procéder « manuellement ». À partir des comportements définis dans la section 9.2.3.3, nous pouvons décrire, à l'aide d'un langage algorithmique simple, $\mathcal{C}_{A_1}$ et $\mathcal{C}_{A_2}$ comme illustré dans les tables 10.1 et 10.2.

$\mathcal{C}_{A_1} \triangleq$
Attributes:
Générateur : object ← instance de l'objet de l'environnement identifié par `Générateur`
nomenclature_capteur : object ← instance du modèle abstrait des nomenclatures de `capteur`

Event handler ordre_de_fabrication (taille_OF : $\mathbb{N}^+$)
begin
Générateur.generate_entity (taille_OF * nomenclature_capteur.MP_quantity)
end

TAB. 10.1 – Exemple - comportement de l'agent A_1

10.4 Conclusion

La conception est la seconde phase importante du processus de modélisation. À partir du modèle abstrait issu de la spécification, elle permet de construire un modèle de système multi-agents dont le rôle sera de mettre en œuvre le processus de simulation. Dans ce chapitre, nous présentons une architecture de système multi-agents qui nous semble capable de répondre à notre problématique de modélisation et de simulation de systèmes industriels distribués. L'introduction d'agents facilitateurs et d'agents pour la simulation, nous permet de regrouper les agents en fonction de la partie du modèle abstrait dans laquelle ils se trouvent. Ainsi, chaque partie du modèle abstrait correspondant à un membre d'un groupement d'entreprises sera représentée par un ensemble « connexe » d'agents. Nous utilisons alors les agents facilitateurs comme passerelles entre les différents ensembles composant le modèle distribué de simulation.

$\mathcal{C}_{A_2} \triangleq$
Attributes:
Chez_E2 : object ← instance de l'objet de l'environnement identifié par `Chez` E_2

Event handler forward_entity (entité : entité)
begin
use_service (Chez_E2.remoteaddress, entité)
end

TAB. 10.2 – Exemple - comportement de l'agent A_2

Après avoir présenté l'architecture proposée au sein de $\mathcal{MAMA}$-$\mathcal{S}$, nous décrivons un ensemble de règles de traduction qui permettent de passer de manière quasi mécanique du modèle abstrait de simulation au modèle conceptuel.
Dans le chapitre suivant, nous décrivons comment ce modèle conceptuel est utilisé pour créer un modèle de simulation qui soit exécutable.

Chapitre 11

Environnement de simulation

Dans ce chapitre, nous présentons à la fois les contraintes associées à la phase de réalisation du cycle de vie de $\mathcal{M}_{\mathcal{A}}\mathcal{M}\mathcal{A}$-$\mathcal{S}$, mais aussi une description sommaire de l'implantation de l'environnement de simulation associé à notre approche méthodologique.
La réalisation est la dernière phase que nous voulons adapter au support de la modélisation et de la simulation de systèmes industriels distribués. L'objectif est ici de transformer le modèle de système multi-agents obtenu précédement en un système multi-agents directement exécutable. Pour ce faire, c'est lors de cette phase que nous choisirons les outils qui devront réaliser la simulation (Arena® [Kelton et al., 1998], Simple++® [Tecnomatix®, 2001], ...) et les plateformes multi-agents servant de support d'exécution aux agents (MAST [Boissier et al., 1998], ARéVi [Duval et al., 1997], SWARM [Burkhart, 1994], Aalaadin [Ferber et Gutknecht, 1998], ...).
Dans la suite de ce chapitre, nous abordons successivement :
– la problématique de synchronisation des modèles de simulation ;
– les contraintes associées à la phase de réalisation ;
– une proposition d'implantation du sma en utilisant Visual Basic® et Arena® ;
– l'intégration d'autres outils de simulation ;
– les bases d'un environnement de modélisation.
En premier lieu, nous rappelons brièvement nos principaux objectifs et nos propositions.

11.1 Résumé des objectifs et des propositions

Nous avons vu que l'évolution des organisations industrielles a entraîné les communautés scientifique et industrielle à s'intéresser à ces nouvelles structures tant au niveau de la modélisation que celui de la simulation. Malheureusement, les outils et les méthodologies existantes sont rarement adaptés au support de ces systèmes

distribués. Notamment, les problématiques de formalisation de ces systèmes, de mise en évidence des flux et des sous-systèmes, de décentralisation et de réutilisation des modèles nous semblent intéressantes à résoudre.
Pour ce faire, nous proposons, dans les chapitres 9 et 10, une approche méthodologique basée sur UML et sur les systèmes multi-agents. Cette approche étend le cycle de vie « classique » des méthodologies existantes de simulation et propose notamment :

- La phase de spécification permet de créer un modèle abstrait de simulation supportant les nouvelles organisations d'entreprises. Le langage de modélisation est défini au travers d'un métamodèle UML.
- Durant la phase de conception, un modèle de système multi-agents est obtenu à partir du modèle abstrait. Ce nouveau modèle est basé sur une architecture que nous rappelons ci-dessous.
- Enfin, la phase de réalisation avec, entre autres, l'implantation du modèle de système multi-agents. C'est durant cette étape du cycle de vie que les choix des outils de simulation et des plateformes multi-agents sont également réalisés.

Ainsi, nous proposons une architecture logicielle illustrée par la figure 11.1. Selon l'approche « Voyelles » de définition d'un SMA, cette architecture contient un environnement et des agents. L'environnement est un « espace » dans lequel les objets de simulation sont regroupés au sein d'outils de simulation. Ce choix arbitraire de modélisation nous permet à la fois de réutiliser les outils existants de simulation et de considérer une partie du modèle du système industriel comme une boîte noire. Notamment, nous pouvons nous intéresser plus particulièrement au support de la distribution des modèles et à celui du sous-système décisionnel.

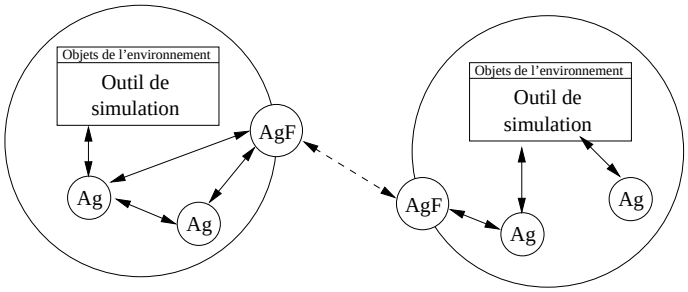


FIG. 11.1 – Architecture du SMA de simulation

L'architecture que nous proposons est composée de sous-systèmes d'agents correspondant à un modèle abstrait local. Ils communiquent entre eux par le biais d'agents facilitateurs qui servent à la fois de passerelles et de pages jaunes et blanches. Dans le chapitre précédent, nous avons vu comment définir l'ensemble des artefacts utilisés dans notre architecture. Dans la suite de ce chapitre, nous allons plus particulièrement nous intéresser à l'implantation d'un SMA de simulation et à l'environ-

nement de simulation.

11.2 Synchronisation des modèles de simulation

L'un des problèmes majeur de la simulation distribuée est la synchronisation des différents modèles [Filloque, 1992]. En effet, il faut s'assurer que les entités transitant d'un modèle vers un autre arrivent ni trop tôt, ni trop tard. Cela se traduit par le fait que le modèle recevant l'entité a une date de simulation inférieure ou égale à celle à laquelle l'unité doit arriver dans ce système. De manière plus formelle, K. Mani Chandy, Jayadev Misra et Randall Bryant ont répondu aux problèmes de synchronisation par la définition de la notion de causalité [Chandy et Misra, 1988 ; Chandy et Misra, 1979 ; Bryant, 1979] :

— Définition 11.1 : Causalité —

La causalité est le principe de non influence du futur sur le passé. Si cette contrainte est respectée localement et que les interactions se font uniquement par messages estampillés, alors la simulation traite les événements dans un ordre consistant avec les ordres partiels imposés par la contrainte de causalité du système réel.

Notons qu'un message est un événement transitant entre deux modèles distribués. De plus, une seconde contrainte doit être respectée pour que la synchronisation des modèles de simulation puisse être effectuée. Cette nouvelle contrainte, dite de vivacité, est définie comme suit :

— Définition 11.2 : Vivacité —

La vivacité est la contrainte imposant au temps de toujours progresser. Son respect permet à une simulation de ne pas se trouver en situation de blocage.

Dans la suite de cette section, nous présentons les trois approches majeures utilisées pour que la causalité soit respectée. Dans ces descriptions, nous utilisons la notation suivante :

- les événements sont définis par le doublet $\langle t_r, type \rangle$ où t_r représente la date d'occurence de l'événement et *type* sa description ;
- la notion d'horloge locale à un modèle, notée TVL, correspond au minimum des estampilles des événements en attente d'un traitement : $\text{TVL} = \min(\{\forall e, tr_e\})$ où tr_e est l'estampille de l'événement entrant e ;

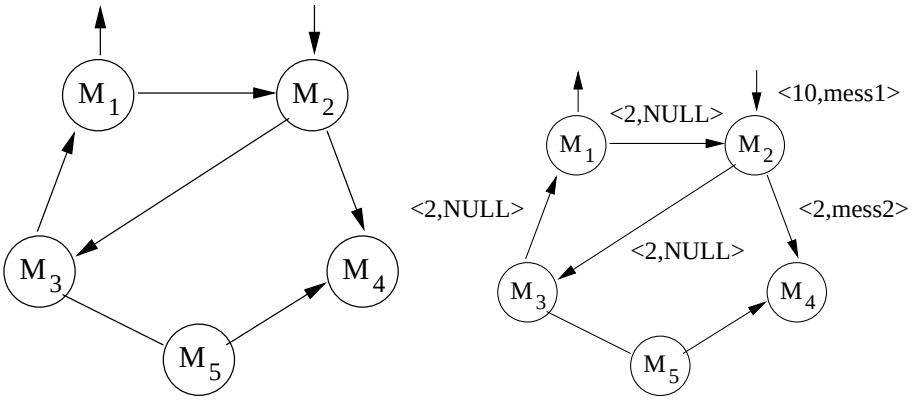
– Pour respecter la contrainte de causalité, l'horloge commune à tous les modèles (notée TVG) est définie comme le minimum des horloges locales : TVG = $\min(\{\forall m, \text{TVL}_m\})$ où $textsctvl_m$ est l'horloge locale du modèle m.

11.2.1 Approches de synchronisation

Les trois approches de synchronisation reconnues par la communauté scientifique sont les approches pessimiste, optimiste et hybride.

11.2.1.1 Approche pessimiste

Dans cette approche, pour vérifier la contrainte de causalité, un modèle m ne doit traiter un événement e que s'il est certain qu'il ne recevra pas des autres modèles un message porteur d'une date inférieure. Ce principe est qualifié de pessimiste par K. Mani CHANDY et Jayadev MISRA [CHANDY et MISRA, 1979].
Ce principe de sûreté peut provoquer des situations de blocage. Ces dernières se produisent lorsqu'un modèle ne reçoit pas de messages sur ses entrées. Par exemple, la figure 11.2(a) illustre un système composé de cinq modèles. Ce système possédant un cycle entre M_1, M_2 et M_3, le modèle M_4 peut rester bloqué dans l'attente d'un message en provenance de M_5 (la contrainte de vivacité n'est alors plus respectée). Ainsi, la différence entre les différents algorithmes pessimistes réside essentiellement dans l'approche de satisfaction de la contrainte de vivacité.



(a) Topologie de modèles distribués

(b) Situation de blocage

FIG. 11.2 – Synchronisation - Exemples

K. Mani CHANDY et Jayadev MISRA proposent un algorithme de prévention des blocages. Pour cela, il emploie des messages supplémentaires appelés NULL. Lors de la simulation, un modèle doit envoyer, dès réception d'un évenement, un message sur

tous ses canaux de sortie. S'il ne peut émettre un message utile, il doit envoyer un message Null contenant sa tvl. Ainsi, un autre modèle pourra décider de traiter un événement grâce aux informations que lui procurent sa propre tvl et la borne inférieure des temps de ses voisins. [Chandy et Misra, 1979] montre que l'utilisation des messages Null avec une estampille correspondant à la tvl ne permet pas de résoudre complètement le problème de satisfaction de la contrainte de vivacité. La figure 11.2(b) illustre un exemple de blocage provoqué par l'utilisation des messages Null. La réception par M_3 d'un message $\langle 2, \text{Null} \rangle$ provenant de M_2 ne lui permet pas d'avancer. Par conséquent, il émet le message $\langle 2, \text{Null} \rangle$ vers M_1. Ce dernier ne pouvant pas non plus avancer, il émet le même message vers M_2. De cette situation émerge un blocage entre M_1, M_2 et M_3. Ce problème est éliminé en introduisant la notion de prédiction (notée σ). Cette valeur locale à chaque modèle est strictement positive et représente la quantité de temps futur pour lequel le modèle peut prévoir son comportement, c'est-à-dire les messages qu'il enverra. Cette nouvelle contrainte de *prédictabilité* impose qu'à chaque envoi d'un message Null, l'estampille soit $\min(\{\forall e, tr_e\}) + \sigma$ pour tout événement e défini par $\langle tr_e, type_e \rangle$.
[Chandy et Misra, 1981] proposent une autre approche pessimiste : la guérison de blocage. Contrairement à la précédente, elle n'utilise pas les messages Null. Chaque modèle consomme l'événement en entrée qui possède la plus petite estampille. Si la liste est vide, le site se bloque. Cette technique permet de ne pas violer la contrainte de causalité. Par contre, celle de vivacité peut être transgressée. Un mécanisme associé au simulateur permet de détecter les blocages et de relancer les sites correspondants. Pour cela, les étapes suivantes sont appliquées :

1. simulation jusqu'à une situation de blocage,
2. détection du blocage,
3. destruction du blocage en relançant le ou les modèles qui ont la plus petite estampille des événéments en entrée.

Cette approche utilise un algorithme dérivé de celui proposé par Edsger Dijkstra et Carel Scholten [Dijkstra et Scholten, 1980]. Malheureusement, cet algorithme admet des situations de blocage sur les sorties notamment à cause de l'hypothèse de files bornées. [Ingels et Raynal, 1990] proposent une solution simplifiée grâce à l'hypothèse des files infinies.

11.2.1.2 Approche optimiste

Un protocole optimiste n'impose aucune contrainte sur le comportement des modèles [Fujimoto, 1990b ; Fujimoto, 1990a]. Il laisse les erreurs de causalité se produire, les détecte et les répare. Un modèle peut avancer dans le temps aussi vite qu'il le peut, consommant tous les événements présents en entrée, et en faisant

le pari qu'il ne se produira aucune erreur. La vivacité est garantie car les modèles n'attendent jamais, leur TVL est incrémentée jusqu'à la fin de la simulation. Une violation de la contrainte de causalité est détectée quand l'estampille d'un événement arrivant est inférieure à la TVL. Un mécanisme est alors mis en œuvre pour annuler toutes les actions réalisées entre la date de l'événement et la TVL courante. Cette approche nécessite la sauvegarde de l'ensemble des états de la simulation. Par rapport aux protocoles pessimistes, on fait l'hypothèse que le temps passé à défaire les traitements erronés sera inférieur à celui nécessaire pour s'assurer que l'on pouvait avancer. Un espace de sauvegarde suffisant est la seconde hypothèse.
David JEFFERSON et Henry SOWIZRAL sont les premiers a avoir proposé un algorithme optimiste de synchronisation sous le nom de « Time Warp » [JEFFERSON, 1985a ; JEFFERSON et SOWIZRAL, 1985b]. Le principe est de diffuser des *anti-messages* pour annuler les effets distants des événements erronés. Dans cette approche les événements sont définis par $\langle te, te, S_e, S_r, s \rangle$ où :
– te est la TVL de l'émetteur, et est utile pour le mécanisme de retour arrière ;
– tr est équivalent à l'estampille de l'événement (date d'arrivée de l'événement) ;
– S_e est le modèle émetteur ;
– S_r est le modèle récepteur ;
– s est le signe du message : $\oplus$ indique un message « normal » et $\ominus$ un anti-message.
Lors de l'arrivée d'un message, le modèle compare la date d'estampille tr à sa propre TVL :

if $tr <$ TVL **then**
le message est simplement placé dans la file d'entrée et sera traité à son tour
else
les traitements effectués entre tr et TVL sont erronés ; il faut faire un retour en arrière qui conduit aux actions suivantes :

1. annuler les traitements locaux en restaurant un des vecteurs d'état sauvegardés avec une date inférieure à tr ;
2. annuler les émissions de messages réalisées après tr. Trois cas sont envisageables :
   (a) le message à annuler n'est pas encore émis : il suffit de le supprimer de la file de sortie ;
   (b) le message émis n'a pas encore été traité par le récepteur : ce dernier le retire de sa file d'entrée ;
   (c) le message a été consommé : le récepteur doit réaliser le traitement de retour en arrière à partir de la première étape ;
3. annuler les demandes d'actions définitives générées après tr.

end if

Cette approche fait l'hypothèse de l'utilisation d'un espace mémoire infini. Passer à une représentation bornée, plus réaliste pour les systèmes réels, nécessite de pouvoir limiter les besoins pour récupérer l'espace inutile. [Samadi, 1985] prouve l'existence d'une borne inférieure en dessous de laquelle le système ne peut jamais revenir. Elle permet d'oublier les états correspondant aux dates inférieures.
De nombreuses variantes ont été étudiées pour limiter l'effet négatif des retours arrière.
L'*annulation paresseuse* permet de retarder l'émission des anti-messages tant que l'on n'est pas sûr que les messages émis soient réellement en faute. Cette approche a pour origine le fait que la ré-exécution peut conduire en partie aux mêmes conséquences que l'exécution en faute [Gafni, 1988 ; Reiher, 1990], et « qu'un bon événement peut être généré par une erreur » [Berry, 1986]. Le principal inconvénient de cette approche est le retard apporté à la correction des erreurs réelles qui permet aux effets dévastateurs de résider plus longtemps dans la simulation.
[Sokol et al., 1988] propose une autre solution, appellée « Moving Time Window », qui consiste à limiter l'optimisme de l'exécution pour empêcher les erreurs de trop se propager. Pour cela, une *fenêtre* fixant un horizon au delà duquel la tvl ne peut aller est fixée. Cette démarche est critiquée car il est difficile de déterminer la bonne taille de la fenêtre, mais aussi parce qu'un gel de la simulation peut se produire malgré une évolution correcte.
[Madisetti et al., 1988] propose une optimisation appellée « Wolf Call » qui consiste à envoyer un message spécial gelant tous les modèles qui ont été perturbés par l'erreur. Cette technique nécessite toutefois une transmission rapide mais aussi la maîtrise des délais de transmission en temps réel pour déterminer la sphère d'influence des messages erronés.

11.2.1.3 Approche hybride

Paul Reynolds soutient que la dichotomie dans le domaine de la simulation distribuée entre l'approche pessimiste et l'approche hybride est une erreur [Reynolds, 1988]. Il pense qu'il existe un grand nombre de choix dont les deux extrèmes sont les approches pessimiste et optimiste.
Une approche couramment utilisée est l'ajout du comportement opposé à celui choisi comme base.
Par exemple, sur une base pessimiste, il est possible de plaquer un certain optimisme en calculant les événements sûrs puis, par précaution, les événements moins sûrs. Dans ce groupe, on peut trouver :
– un algorithme proposé par Boris Lubachevsky, Adam Shwartz et Alan Weiss qui permet de traiter les événements non sûrs en fonction de connaissances données par l'utilisateur [Lubachevsky et al., 1989] ;

– une méthode de calculs spéculatifs qui sont réalisés sans envoi de message pouvant générer des erreurs. Les émissions sont alors éxécutées une fois la validation acquise [MEHL, 1991] ;
– une proposition de Paul REYNOLDS et Philip DICKENS appellée SRAD/LR et proche de la méthode précédente. Elle permet de limiter l'agressivité en calculant dynamiquement la limite supérieure d'optimisme [REYNOLDS et DICKENS, 1990] ;
– une approche par calculs spéculatifs avec émission de messages, mais avec une fenêtre limitée statistiquement [SOKOL et al., 1989].

D'autres propositions consistent à dégrader le modèle optimiste en calculant dynamiquement des fenêtres en fonction du comportement de chaque modèle.

Une approche originale, « COMPOSITE ELSA », consiste à rechercher a priori les parties de modèles pour lesquelles l'approche pssimiste est préférable et d'autres parties pour lesquelles l'optimisme serait plus adapté [ARVIND et al., 1991 ; ARVIND et SMART, 1992].

11.2.2 Choix d'une approche

Comme nous pouvons le voir clairement, la communauté scientifique de la simulation distribuée s'est largement penchée sur le problème de la synchronisation des modèles de simulation distribués. Pour notre part, nous ne désirons pas proposer une nouvelle approche. Toutefois, il reste nécessaire de synchroniser les modèles de simulation. Nous avons donc choisi un algorithme existant : notre choix s'est porté sur un algorithme pessimiste car il nous semble plus simple à implanter dans le cadre de notre maquette de développement. Notons toutefois que ce choix reste arbitraire. En effet, nous pensons qu'une étude plus complète devrait être réalisée pour savoir quel type de synchronisation est le mieux adaptée à la simulation distribuée avec des systèmes multi-agents. Notons l'existence d'environnements de simulation basés sur les SMA comme ARÉVI [DUVAL et al., 1997].

11.3 Cahier des charges de la phase de réalisation

Dans cette section, nous décrivons les principes généraux de l'implantation des objets du SMA de simulation, une architecture d'implantation à l'aide des logiciels ARENA® et VISUAL BASIC®, et l'implantation de l'exemple du groupement de trois entreprises déjà présenté dans les chapitres précédents.

L'un des objectifs de la phase de réalisation est d'obtenir un modèle exécutable correspondant au modèle de système multi-agents obtenu durant la phase de conception.

11.3.1 Concernant les objets de l'environnement

Comme nous l'avons vu dans le chapitre précédent, l'environnement est une composante majeure du SMA. L'un des objectifs de la réalisation est d'implanter l'ensemble des objets de l'environnement définis dans le modèle conceptuel en respectant les contraintes liées à ce dernier (unités de traitements, liens, ...) :

1. Les objets utilisés par un agent doivent obligatoirement proposer une interface de communication utilisable par ce dernier. Cette interface est basée sur l'échange de messages. Ainsi, pour chaque service de type `method` (resp. `event`), l'objet doit pouvoir traiter la réception (resp. l'émission) d'un message étiquetté par le nom du service. La syntaxe utilisée par les messages dépend des outils d'implantation choisis.
2. Les objets de l'environnement doivent implanter des comportements compatibles avec ceux attendus dans le modèle conceptuel.
3. Tout objet est indépendent des autres objets, sauf si un lien est explicitement décrit dans le modèle conceptuel. Dans ce cas, la plateforme d'implantation doit pouvoir supporter les liens et les relations entre les objets.
4. Les objets ne doivent pas mettre en danger le déroulement de la simulation. Ainsi, il doivent être conformes à la politique de synchronisation des modèles (les contraintes de causalité et de vivacité doivent être respectées).

Le choix de l'outil de réalisation pour l'environnement du SMA est libre. Toutefois, nous rappelons l'existence d'une contrainte particulière que nous avons posé dans la section 2.3.4. Elle consiste à utiliser, dans la mesure du possible, les outils existants de simulation. Par conséquent, nous devons étudier la compatibilité de ces outils avec l'implantation de l'environnement. Cette dernière peut être décrite comme suit :

- les outils de simulation sont fortement adaptés au support des objets représentant les artefacts du sous-système opérationnel. En effet, pour la plupart, ils se sont concentrés sur la modélisation en prenant le point de vue opérationnel comme fil conducteur. Par exemple, dans ARENA®, les unités de traitements, les files d'attentes et les générateurs d'entités peuvent être représentés par les objets DELAY, QUEUE et CREATE. De plus, les outils de simulation respectent localement les contraintes de causalité et de vivacité imposées par un algorithme de synchronisation.
- une partie des artefacts du sous-système informationnel sont naturellement supportés par les outils de simulation : définition d'entités (dans notre exemple avec ENTITIES), ...
- nous considérons que les outils existants de simulation ne sont pas adaptés au support du sous-système décisionnel. Toutefois, nous n'interdisons pas l'utilisation d'outils de simulation spécifiquement dédiés à ce sous-système.

Dans la section 11.4, nous proposons une architecture d'implantation permettant au logiciel ARENA® de s'intégrer dans notre environnement de simulation. Ceci nous permet d'illustrer l'intégration d'un logiciel existant de simulation en surchargeant une partie de ses fonctionnalités (en l'occurrence, en utilisant son interface VISUAL BASIC FOR APPLICATION®). L'intégration d'outils ne supportant pas ce type d'extension peut être réalisée mais demande une approche plus complexe. Nous considérons qu'il est à la charge du réalisateur du SMA de simulation d'implanter les procédures et les programmes permettant à l'outil qu'il veut utiliser de s'intégrer dans notre environnement de simulation. Nous imposons simplement que cette intégration respecte les contraintes exprimées ci-dessus.

11.3.2 Concernant les agents

Les agents sont directement issus du modèle conceptuel. Ainsi, les contraintes de réalisation sont les contraintes propres à tout système multi-agents :
– autonomie de raisonnement des agents ;
– capacités d'interactions ;
– distributivité informatique du système ;
– modélisation selon l'approche « Voyelles ».
À ces contraintes, nous ajoutons que les réalisations des agents doivent respecter leur définition conceptuelle : ils doivent proposer et utiliser exclusivement les même services. Ainsi, le passage du modèle conceptuel au modèle informatique est beaucoup plus simple à réaliser que pour les objets de l'environnement. Notons que les agents doivent respecter les protocoles et les formats de messages utilisés par les objets de l'environnement. D'autre part, nous utilisons la même notation pour décrire les messages échangés par les divers agents (qu'ils soient faciliteurs ou non).
Dans la section 11.5, nous proposons une implantation d'agent.

11.4 Implantation des objets de l'environnement avec Arena®

Nous avons décidé arbitrairement d'utiliser l'outil de simulation ARENA® parce qu'il peut s'intégrer, moyennant quelques adaptations, dans un environnement de simulation hétéroclite comme celui que nous voulons mettre en œuvre.
Dans un premier temps, nous proposons une table permettant de transformer les objet de l'environnement du SMA en artefacts de modélisation propres à ARENA®. La table 11.1 illustre une partie des correspondances que nous pouvons utiliser pour obtenir un modèle de simulation exécutable à partir du modèle conceptuel. Par exemple, les objets conceptuels de type `ProcessingUnit` (ou unité de traitement) sont traduits sous forme de DELAY dont la loi statistique correspond à celle utilisée

par l'objet conceptuel. L'ensemble de ces correspondances pourrait faire l'objet d'une bibliothèque de composants particuliers. En effet, cette perspective permettrait de faciliter la construction et la lecture des modèles Arena®.

Objet conceptuel	Objet Arena®
`ProcessingUnit`	DELAY
`Link`	LINK
`Queue`	QUEUE
`EntityGenerator`	CREATE et VBA ou CREATE
`EntryPoint`	VBA
`ExitPoint`	VBA et DESTOY
`SubModel`	SUBMODEL
`Jump`	STATION
`EntityDestructor`	DISPOSE
`RemoteModel`	VBA
`Resource`	RESOURCES
`Entity`	ENTITIES
...	

Tab. 11.1 – Du modèle conceptuel à Arena®

Certaines de ces relations donnent lieu à la création d'objet de type VBA. Ces derniers doivent être obligatoirement accompagnés de code Visual Basic for Application® qui implante le comportement attendu pour l'objet. Ce comportement est en général l'implantation des méthodes correspondant aux services proposés par l'objet conceptuel. Pour ce faire, nous proposons une interface permettant à Arena® de recevoir des appels à des services et d'émettre des événements. La figure 11.3 illustre l'architecture de cette interface.

Il s'agit d'un objet OCX (ou ActiveX®) qui peut être directement utilisé dans le code Visual Basic for Application® d'Arena®. Il propose des méthodes permettant d'envoyer des messages en utilisant un langage et un protocole reconnu par les agents. Il permet aussi de recevoir des messages. Chacun d'entre eux fera alors l'objet d'une génération d'un événement dans l'environnement Visual Basic®.

Soit s un service issu du modèle conceptuel défini à la page 160 comme le n-uplet $\langle id, type, \mathcal{P} \rangle$. L'objectif de l'interface étant de permettre l'utilisation des services proposés ou demandés par l'objet de l'environnement, nous définisons les messages envoyés et reçus par l'interface comme le n-uplet $\langle id, estampille, type, nom\_message, \mathcal{P}_e \rangle$ où :

- id est l'identificateur unique du message ;
- $estampille$ est la date de simulation à laquelle l'événement a été généré ;
- $type$ représente le type du message (`event` pour un événement, et `method` pour un appel de méthode) ;

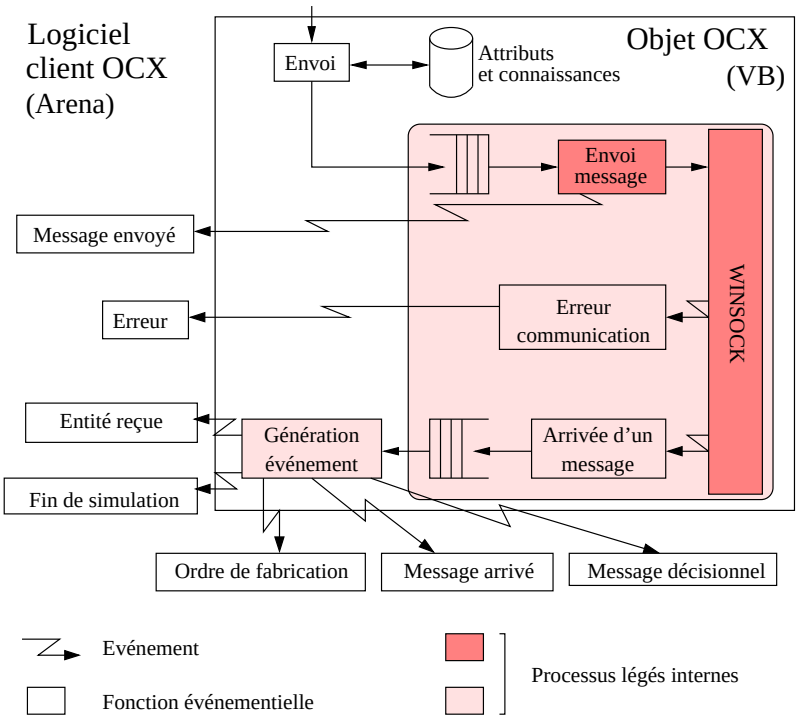


FIG. 11.3 – Interface entre ARENA® et le SMA

– *nom_événement* est le nom de l'événement généré ;
– $\mathcal{P}_e$ est l'ensemble des paramètres de l'événement. Chacun d'entre eux est défini par le doublet $\langle nom, valeur \rangle$ où *nom* est le nom du paramètre et *valeur* la valeur qui y est associée.

Notons que l'implantation réelle des interactions considère qu'un message est une chaîne de caractères. Par exemple, le message décrivant la génération d'une instance de l'événement `envoi_entité` avec en paramètre l'entité à envoyer est représentée par la chaîne `"[0001|13431|event|envoi_entité|{[entité|[entité0000293,{[quantité|5]}]]}]"`. Cet exemple nous permet de mettre en avant certaines caractéristiques des objets définis dans le modèle conceptuel. En effet, nous considérons que ces derniers peuvent toujours être représentés sous forme de chaînes de caractères. Dans notre exemple, une entité (définie dans le sous-système informationnel) est représentée par le doublet $\langle id, \mathcal{A} \rangle$ où *id* est l'identificateur de l'entité et $\mathcal{A}$ l'ensemble des attributs associés à cette entité. Nous utilisons le protocole suivant :

– les champs d'un n-uplet ou les membres d'un ensemble sont séparés par le caractère « | »,
– un n-uplet est encadré par des crochets,
– un ensemble est encadré par des accolades.

Dans le futur, nous pensons rendre les messages plus lisibles en utilisant le langage

« eXtended Markup Language » (XML) comme support de formalisation et par conséquent en définissant une grammaire sous forme de « Document Type Definition » (DTD) [W3 Consortium, 2001].

Reprenons l'exemple du groupement de trois entreprises et plus particulièrement le modèle conceptuel de l'entreprise E_1 présenté dans la section 10.3. La figure 11.4 illustre le modèle Arena® résultant de la transformation du modèle conceptuel.

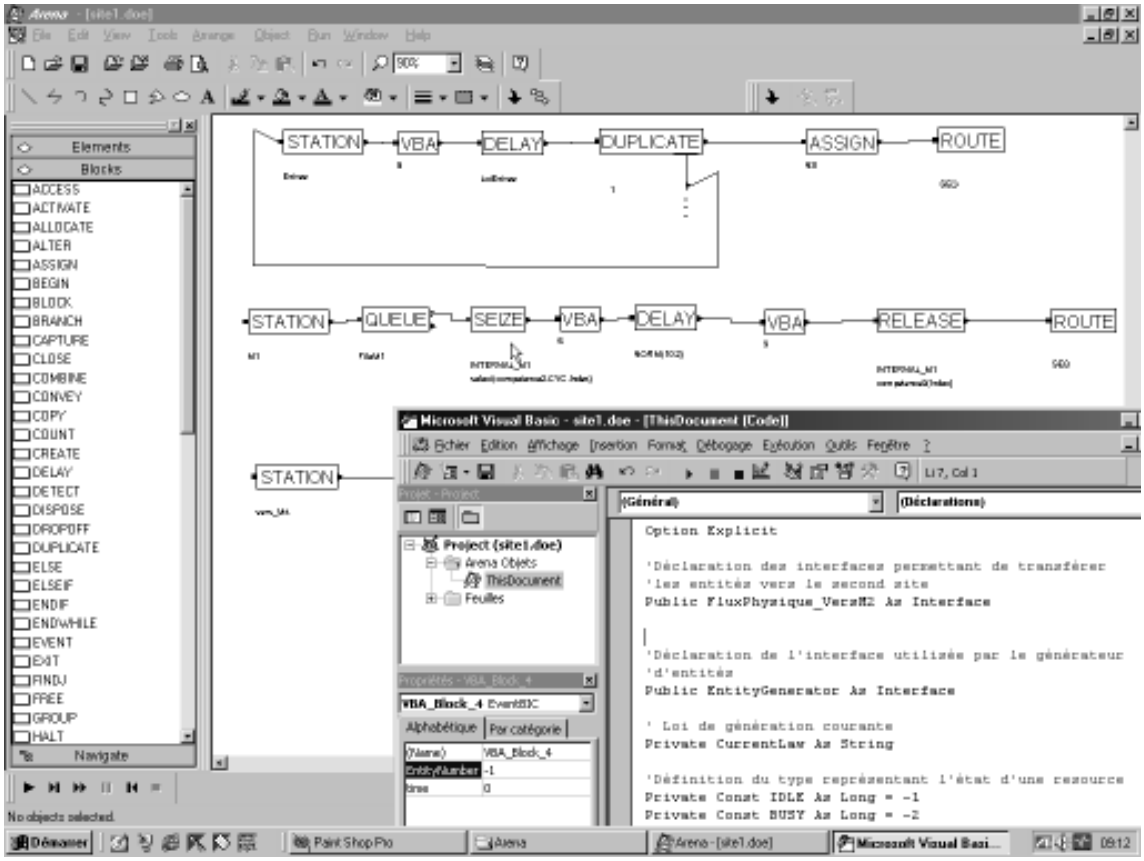


Fig. 11.4 – Exemple d'implantation de l'environnement sma

11.5 Implantation des agents

Dans cette section, nous décrivons les principes de l'implantation des agents. Nous avons décidé d'utiliser le langage Visual Basic® pour construire nos agents. En effet, il s'agit d'un environnement de programmation simple à mettre en œuvre et adapté à la réalisation de maquettes de logiciels. De plus, une partie des bibliothèques de gestion des messages qui ont été rédigées pour les interfaces Arena® ↔sma ont pu être réutilisées. La figure 11.5 illustre l'implantation d'un agent pour la simulation.

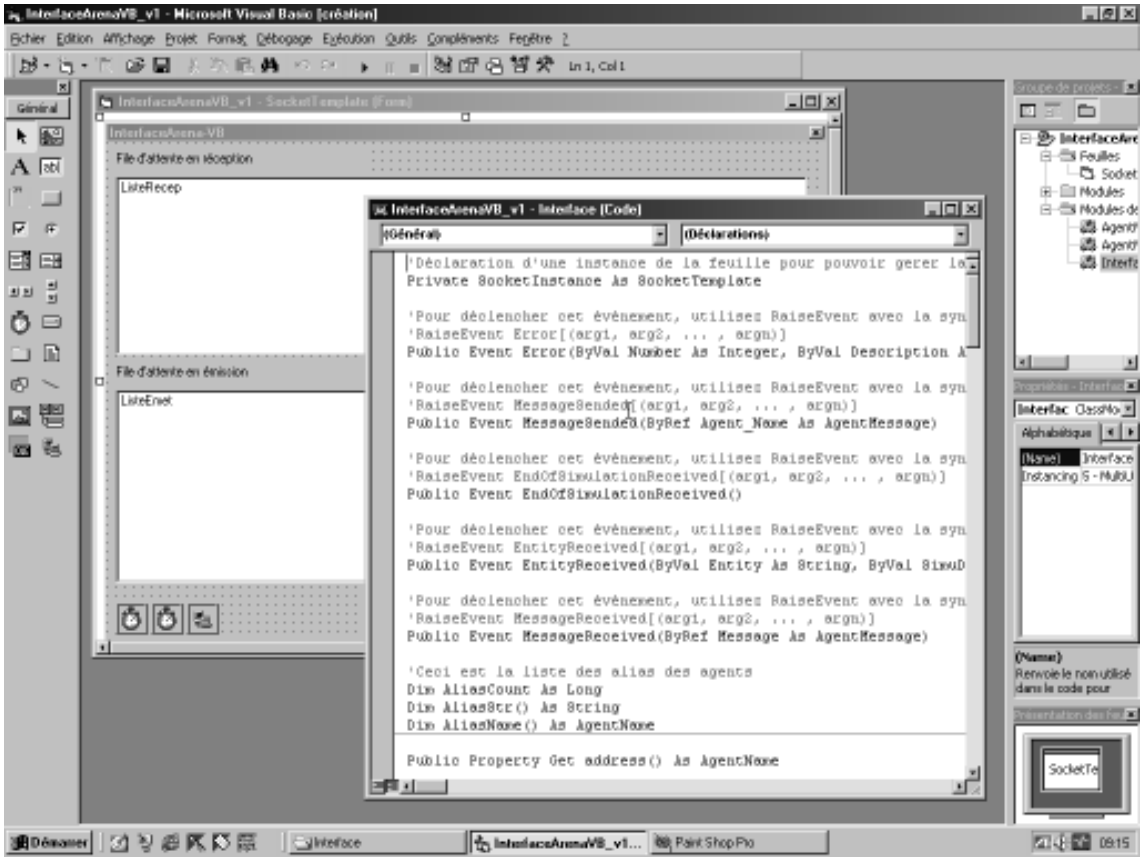


FIG. 11.5 – Exemple d'implantation d'un agent pour la simulation

11.6 Autres outils et plateformes

Nous sommes conscients que nos choix de réalisation sont très restrictifs. Toutefois, celui d'ARENA® nous a permis de mieux comprendre les capacités d'interaction d'une grande majorité des outils de simulation existant sur le marché. Cette approche peut être facilement adaptée pour intégrer des outils comme SIMPLE++® par exemple. Il reste cependant l'existence d'autres outils n'intégrant pas VISUAL BASIC FOR APPLICATION® (CSIM18, ...). Ces derniers doivent faire l'objet d'un travail spécifique. Comme nous nous imposons des contraintes d'intégration des outils de simulation, il est nécessaire de trouver un moyen de communication entre ces outils et notre système multi-agents. Par exemple, la création d'un fichier contenant les résultats de la simulation qui soit lu régulièrement par l'interface connectée au SMA.

Notre second choix de réalisation concerne l'utilisation de VISUAL BASIC® pour implanter l'ensemble de nos agents. Nous sommes conscients que l'utilisation d'une plateforme spécialisée comme MAST [BOISSIER et al., 1998] ou AALAADIN [FERBER et GUTKNECHT, 1998] aurait apporté plus de facilités pour traduire le modèle conceptuel en un modèle informatique exécutable. Toutefois, le

choix d'une plateforme multi-agents est contraint par la reconnaissance de l'approche « Voyelles » [Demazeau, 1997].

11.7 Conclusion

La phase d'implantation est la dernière phase majeure de notre approche méthodologique car nous ne nous sommes pas intéressés à l'analyse des résultats de simulation. Elle permet, à partir du modèle conceptuel issu de la phase précédente, d'obtenir un modèle informatique de simulation.

Nous considérons qu'un modèle informatique de simulation doit respecter certaines contraintes pour correspondre au modèle conceptuel de simulation. Par exemple, nous considérons que tous les objets de l'environnement du SMA doivent fournir les services décrits durant la phase de conception. D'autre part, l'une de nos contraintes de réalisation étant l'utilisation d'outils existants de simulation, nous proposons une architecture logicielle permettant à ces derniers de s'intégrer au sein de notre plateforme de simulation (architecture uniquement validée avec Arena®). Cette interface suppose que l'outil de simulation est capable d'utiliser des objets dynamiques (ActiveX® ou OCX) [Microsoft® Corporation, 1998]. Ainsi, l'interface propose des méthodes permettant d'envoyer des messages aux agents de notre environnement de simulation, et de générer des événements à chaque fois qu'un agent désire utiliser un des services associés aux objets de l'environnement. Ces derniers sont en effet directement transformés en objets correspondants dans l'outil de simulation choisi.

Du point de vue des agents, nous proposons une architecture d'implantation simple décrite en Visual Basic®. Ainsi, les agents implantent les comportements définis dans le modèle conceptuel. Cette architecture intègre également les protocoles et les formats de messages définis durant la phase de conception. Nous proposons deux architectures distinctes. La première correspond aux agents facilitateurs. Elle ne peut être changée par l'utilisation de $\mathcal{MAM}$A-S. La seconde architecture proposée est un canevas de modélisation dans lequel l'utilisateur devra remplir les parties non fournies (comportement interne des agents, réaction à des demandes de services, ...). Ainsi, par défaut, nous ne proposons que l'implantation des protocoles de communication entre les agents (facillitateurs ou non) et les AgF et les objets de l'environnement.

Pour résumer, nous avons donc réalisé :

- un objet permettant d'interfacer Arena® et notre système multi-agents ;
- deux prototypes réalisés en Visual Basic® représentant respectivement les agents facilitateurs et les agents pour la simulation ;
- l'implantation sur l'exemple simple présenté dans ce mémoire ;
- la réalisation d'un environnement de modélisation incluant :

- une version partielle d'un environnement de modélisation supportant notre métamodèle UML et basée sur la bibliothèque ARAKHNÊ [1] ,
- une implantation partielle (et non fonctionnelle) du système multi-agents de vérification,
- la génération d'un modèle conceptuel à partir du modèle conceptuel (réalisation partielle).

D'autre part, nous avons réalisé un modèle de simulation dans le cadre d'une application pédagogique [GALLAND et al., 2000c]. Cette implantation permet de faire communiquer plusieurs modèles ARENA® se trouvant sur des machines distantes. Dans ce cadre, nous avons vérifié le fonctionnement des mécanismes de base de notre système multi-agents pour la simulation.

Notons que si nos réalisations sont simples (voir même simplistes), elles nous permettent toutefois d'illustrer la faisabilité informatique de nos propositions. Nous regrettons que ce point de vue ne soit que trop rarement appliqué dans nombre de travaux ayant pour objectif de répondre à des problématiques identiques ou voisines de celles qui nous intéressent.

Nous venons de présenter nos derniers apports dans le cadre de $\mathcal{MAMA}$-$\mathcal{S}$. Nous allons maintenant conclure ce mémoire.

[1] `http://www.arakhne.org/`

Quatrième partie

Conclusion

Chapitre 12

Conclusion générale et perspectives

12.1 Problèmes posés

Conscientes de l'impossibilité de maîtriser l'ensemble de leurs métiers, les entreprises d'aujourd'hui ont tendance à se transformer, à se réorganiser en fonction des opportunités que leur offre leur environnement en terme de compétences. On assiste à un recentrage de certaines entreprises sur les métiers dans lesquels elles possèdent un avantage compétitif, et à une externalisation des métiers annexes. Dans ce contexte, ont émergé de nouvelles formes d'organisations d'entreprises parmi lesquelles nous trouvons les entreprises réseau ou les groupements d'entreprises.
L'un des moyens que possèdent les gestionnaires des systèmes de production pour évaluer les performances de ces derniers est l'utilisation de la simulation. Cette collection d'outils et de méthodes permet de reproduire totalement ou partiellement le comportement des systèmes réels. La mise en œuvre de ce processus de simulation sur les nouvelles organisations d'entreprises pose des problèmes particuliers. Plus spécifiquement, nous nous sommes intéressés aux problématiques de formalisation, de mise en évidence des flux et des sous-systèmes, de support à la réutilisation des modèles de simulation.
L'un des premiers problèmes rencontrés durant la phase de modélisation pour la simulation d'un système manufacturier est le manque de définition formelle des éléments constituant ces derniers. Ainsi, les modèles de base de la simulation tels que les réseaux de Pétri [David et Alla, 1992 ; Wang, 1998] ou la théorie des files d'attentes [Knuth, 1998] permettent de modéliser convenablement un système industriel [Zimmermann, 1994], mais ils restent difficiles à maîtriser et à utiliser. De plus, les approches plus intuitives comme celles proposées par des langages et des logiciels de simulation spécifiques (Arena®, QNAP, ...) ne répondent que partiellement à la problématique de formalisation. En effet, chacun de ces outils propose une

vision des systèmes industriels qui lui est propre. Ainsi, la qualité des modèles de simulation est dépendante des compétences et des connaissances du modélisateur, mais aussi des capacités de l'outil utilisé.

Le second problème qui nous intéresse est celui de la mise en évidence des sous-systèmes et des flux composant un système industriel. En effet, nous pensons que distinguer ces différentes parties est un moyen efficace pour limiter les problèmes de construction, de compréhension et de maintenance des modèles de simulation. Dans ce sens, Jean-Louis LE MOIGNE propose une approche systémique distinguant les sous-systèmes opérationnel, informationnel et décisionnel.

L'une des caractéristiques des nouvelles organisations d'entreprises est la forte distributivité de leurs composants et notamment de leurs centres de décisions. Ainsi, s'il est toujours possible de réaliser un seul et unique modèle de simulation (c-à-d centralisé), il est difficile de réaliser plusieurs modèles interagissant tout en tenant compte des contraintes spécifiquement liées aux systèmes distribués (confidentialité, dynamique de construction des systèmes, ...). Ainsi, si la simulation distribuée apporte des solutions quant à la modélisation et la simulation de plusieurs modèles, elle n'offre aucune structure spécifiquement destinée aux entreprises en réseaux ou groupement d'entreprises.

Enfin, le dernier problème sur lequel nous avons porté notre intérêt est la réutilisation des modèles de simulation. Cette notion est très proche des deux précédentes. Il s'agit de pouvoir construire un modèle suffisamment modulaire pour pouvoir être réutilisé pour construire un autre modèle de simulation (approche de modélisation par composants, ...).

12.2 Propositions

Nous nous sommes démarqués des approches méthodologiques existantes par la proposition d'un environnement de modélisation et de simulation supportant les nouvelles organisations et incluant une approche efficace de modélisation.

Nous pensons que l'utilisation d'une approche systémique de modélisation permettra d'améliorer le processus de modélisation. Ainsi, il est possible de distinguer les structures physiques, les centres pilotant ces dernières et l'ensemble des informations transitant dans le système. Cette dichotomie permet de modéliser un système industriel en s'intéressant particulièrement à l'un de ces aspects à la fois.

Nous nous sommes interrogé sur les moyens à mettre en œuvre pour modéliser et simuler les artefacts composant ces différents sous-systèmes. Si pour la partie opérationnelle, les outils existants de simulation conviennent parfaitement, le sous-système décisionnel nécessite une approche différente. Nous avons constaté la très forte adéquation entre les possibilités des systèmes multi-agents (SMA) et nos be-

soins en modélisation et en simulation des centres de pilotage (appelés aussi centres de décision). En effet, les SMA sont composés d'entités autonomes capables d'interagir pour atteindre un but commun. Plusieurs auteurs avant nous ont démontré les avantages de l'utilisation d'un SMA dans la cadre de la modélisation du sous-système décisionnel.

Plus précisement, nous proposons une approche méthodologique dont l'acronyme est $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$ (« Multi-Agent Methodological Approach for Simulation »). Cette approche est basée sur un cycle de vie des modèles désormais classique : une approche de modélisation en spirale présentée dans le chapitre 8. Ainsi, nous proposons d'étendre les phases de spécification, de conception et de réalisation afin de supporter la modélisation et la simulation de systèmes industriels distribués :

- La spécification est la première phase majeure du cycle de vie attaché à $\mathcal{M}_{\mathcal{A}}\mathcal{MA}$-$\mathcal{S}$. Elle consiste à construire un modèle de simulation de haut niveau, c'est-à-dire un modèle manipulant des artefacts de modélisation indépendants de toute plateforme de réalisation. Dans ce mémoire, nous proposons un langage de modélisation décrit par un métamodèle UML. Ce dernier est accompagné de mécanismes de vérification incluant la vérification locale à l'aide des contraintes imposées par le métamodèle, et un système multi-agents servant à vérifier la cohérence entre les différents modèles distribués.
- La conception est la phase dans laquelle nous introduisons les modèles de systèmes multi-agents. L'objectif de cette étape du cycle de vie est de transformer le modèle issu de la spécification en un modèle de SMA. Nous proposons un ensemble de règles non exhaustives qui permettent de partiellement automatiser cette transformation. D'autre part, nous proposons une architecture de système multi-agents basée sur la définition d'agents facilitateurs, d'agents pour la simulation et d'objets de l'environnement du SMA. Cette architecture nous permet de répondre aux problèmes de décentralisation des modèles, d'une partie de la modularité (modularité au niveau des modèles de simulation), de la mise en évidence des sous-systèmes du système industriel simulé. De plus, nous posons comme hypothèse que le choix des outils d'exécution (outil de simulation ou plateforme multi-agents) n'est pas réalisée durant cette phase. Pour respecter cette contrainte, nous utilisons l'approche de modélisation « Voyelles ». Elle définit un système multi-agents selon quatre facettes : la définition des agents, la définition de l'environnement, la spécification des interactions possibles et la définition de la structure organisationnelle. Cette approche nous semble la plus adaptée car elle ne favorise aucune des facettes. Il nous est alors possible d'aborder la conception du SMA pour la simulation selon le point de vue qui nous convient le mieux.
- La réalisation est la dernière phase à laquelle nous apportons des modifications. Ici, l'objectif est de transformer le modèle de système multi-agents précédent en

un modèle informatique exécutable. Pour cela, nous proposons une architecture logicielle basée sur l'utilisation d'ARENA® et de VISUAL BASIC®. Toutefois, nous ne voulons pas restreindre notre environnement à l'utilisation de ces deux outils. Pour cela, nous décrivons un ensemble de contraintes qui doivent être respectées par les outils existants de simulation et par les plateformes multi-agents pour pouvoir s'intégrer dans notre approche.

12.3 Perspectives

Conscient que le développement d'une méthodologie est un travail de longue haleine, nous avons limité nos travaux aux bases de celle-ci. Ainsi, nous proposons une architecture logicielle agrémentée d'artefacts de modélisation pour pouvoir l'utiliser. Nous avons laissé certains problèmes de côté afin de nous concentrer sur ce que nous avons décrit dans la section précédente.

Nos propositions n'incluent pas de démarches méthodologiques complètement formalisées. Ainsi, à l'instar de Merise, notre approche méthodologique pourrait intégrer des méthodes et des guides aidant l'utilisateur à construire pas à pas ses modèles de simulation. Cette problématique doit s'intéresser aux problèmes de modélisation de systèmes industriels distribués (domaine de la modélisation en entreprise et de la simulation), et de systèmes multi-agents (rappelons que l'approche « Voyelles » n'est pour l'instant qu'une démarche de modélisation et non pas une méthode ou une méthodologie).

Toujours du point de vue méthodologique, nous pensons que l'étude de l'équivalence entre deux modèles de simulation peut améliorer le processus de modélisation. En effet, ce type d'approche permettrait de créer des modèles par raffinements successifs. Mais, comme l'illustre le domaine des méthodes formelles, une approche par raffinements nécessite de « prouver » la validité de ces derniers. La notion de preuve formelle nous semble difficilement applicable dans le cadre de modèles de simulation. Mais, il s'agit tout de même là d'une voie intéressante qui permettrait à $\mathcal{MAMA}$-$\mathcal{S}$ de se démarquer complètement des méthodologies déjà existantes.

Les travaux présentés dans ce mémoire possèdent les limitations suivantes :

- Les aspects touchant au développement collectif de modèles : il nous semble intéressant de s'inspirer des travaux existants dans le domaine des travaux collaboratifs et du développement en groupe.
- Le développement des artefacts de modélisation utilisés durant la spécification : nous avons défini des objets canoniques mais les utilisateurs de méthodologies de simulation peuvent s'attendre à trouver des artefacts de plus haut niveau. Ainsi, il serait intéressant d'étendre le langage de spécification en introduisant des concepts comme les serveurs (composition d'une file d'attente, d'une unité de traitement

et de ressources). L'ensemble de ces nouveaux éléments de modélisation pourrait être regroupé au sein de bibliothèques d'objets métier.

De plus, nous avons laissé de côté certains artefacts qui nous semblent nécessaires à la simulation. Ainsi, la définition dès la phase de spécification des sources extérieures de données ou des données à récolter doit faire l'objet d'une étude plus approfondie.

- L'étude des problèmes de synchronisation au sein d'un système multi-agents pour la simulation de systèmes industriels distribués doit faire l'objet d'une étude plus approfondie. En effet, nous avons utilisé un algorithme pessimiste de synchronisation sans toutefois être absolument certain qu'il s'agit là de l'approche la plus adaptée à notre problématique.
- Nos propositions nécessitent d'être validées sur d'autres outils de simulation et plateformes multi-agents.

Malgré l'ensemble de ces limitations, nous pensons que notre contribution pose les bases d'une approche méthodologique plus complète. Rappelons que notre objectif n'était pas de développer $\mathcal{MAS}$-$\mathcal{S}$ dans sa globalité, mais plutôt de dégager les concepts et les infrastructures nécessaires à sa mise en œuvre et à son développement. Nous pensons que cet objectif a été atteint. $\mathcal{MAS}$-$\mathcal{S}$ est nécessairement destinée à évoluer et à s'étendre, mais nous proposons d'ores et déjà les bases conceptuelles et techniques de cette approche.

Enfin, notre approche méthodologique doit être rapprochée des efforts de standardisation se rapprochant de notre problématique :

- au niveau de la modélisation en entreprise, les travaux sur « Unified Enterprise Modelling Language » (UEML) nous semblent très intéressants. Comme nous l'avons présenté dans la section 4.7, ce projet veut proposer un langage unifié permettant de modéliser les entreprises ;
- dans le domaine de la simulation, un projet récent initié par le groupe d'étude « Discret-Event systems Specification » (DEVS) se rapproche considérablement de notre problématique [DEVS-SG, 2001]. Son objectif est d'étudier l'interopérabilité des modèles de simulation DEVS ;
- enfin, au niveau des systèmes multi-agents, un certain nombre d'efforts sont réalisés pour dégager des concepts et des méthodes pour construire un SMA. Notons plus particulièrement les spécifications proposées par la « Foundation for Intelligent Physical Agents » (FIPA) qui vise à définir l'ensemble des composants d'une plateforme de système multi-agents [FIPA, 2001b].

En conclusion, nous constatons que notre problématique de recherche, évoquée dans ce mémoire, est plus que jamais d'actualité. Nos travaux permettent de nous intégrer dans ces groupes de recherche scientifique et, ainsi, d'y apporter une contribution modeste, mais intéressante à la modélisation et la simulation de systèmes industriels.

Bibliographie

J. Abrial (1996), « *The B-Book : Assigning Programs to Meanings* ». Cambridge University Press.

J. Allen et C. Perrault (1980), « Analyzing intention in dialogues ». *Artificial Intelligence*, 15(3).

J. Allen, J. Hendler et A. Tate (1990), « *Readings in planning.* ». Morgan Kaufmann.

M. Allouche, O. Boissier et C. Sayettat (1997), « Supervision de systèmes évolutifs par une société d'agents ». In *Technique et science informatique*, pp. 1063-1084.

M. Allouche (1998), « *Une société d'agents temporels pour la supervision de systèmes industriels* ». Thèse de doctorat, École Nationale Supérieure des Mines de Saint Etienne.

AMICE (1993), « *CIMOSA : CIM Open System Architecture, 2nd revised and extended edition* ». Springer-Verlag.

D. Arvind, C. Smart et V. Rebello (1991), « Distributed Simulation of Parallel VLSI Architectures ». In *Workshop on Architecture, Algorithms and VLSI*. Bonas, France, pp. 285-298.

D. Arvind et C. Smart (1992), « Hierachical Parallel Discrete Event Simulation in Composite ELSA ». In *6th Workshop on Parallel and Distributed Simulation (PADS'92)*. Newport Beach, USA, pp. 147-158.

J. L. Austin (1975), « *How to Do Things with Words* ». Havard.

G. Babin, Z. Maamar et B. Chaib-draa (1997), « Metadatabase meets distributed AI ». In *First international workshop on Cooperative Information Agents (CIA-97)*.

J. Banks, J. Carson et B. Nelson (1996), « *Discrete Event System Simulation* ». Prentice-Hall.

J. Banks (1999), « Introduction to Simulation ». In *Winter Simulation Conference*, P. Farrington, H. Nembhard, D. Sturrock et G. Evans (Ed.), pp. 7-13.

M. Barbuceanu et M. S. Fox (1995), « COOL : A Language for Describing

Coordination in Multi-Agent Systems ». In *International Conference on Multi-Agent Systems*, V. LESSER (Ed.). San Francisco, pp. 17-24.

K. BARCHARD et A. HAKSTIAN (1997), « The robustness of confidence intervals for coefficient alpha under violations of the assumption of essential parallelism ». *Multivariate Behavioral Research*, 32, pp. 169-191.

B. BAUER, J. P. MÜLLER et J. ODELL (2001), « *Agent UML : A Formalism for Specifying Multiagent Interaction* ». Springer-Verlag, pp. 91-103.

F. BELLEGARDE, C. DARLOT, J. JULLIAND et O. KOUCHNARENKO (2000), « Reformulate Dynamic Properties during B Refinement and Forget Variants and Loop Invariants ». In *ZB 2000 - Lectures Notes in Computer Science*. York, UK, pp. 230-249.

C. BERCHET (2000), « *Modélisation pour la simulation d'un système d'aide au pilotage industriel* ». Thèse de doctorat, Université de Savoie.

O. BERRY (1986), « *Performance Evaluation of the Time Warp Distributed Simulation Mechanism* ». Thèse de doctorat, University of South California, Los Angeles, USA.

G. BIRTWISTLE (1979), « *Discrete Event Modelling on SIMULA* ». Macmillan Press.

B. BOEHM (1981), « *Software Engeneering Economics* ». Prentice-Hall.

O. BOISSIER et Y. DEMAZEAU (1994), « ASIC : an architecture for social and individual control and its application to computer vision ». In *Lectures Notes in artificial Intelligence*. MAAMAW workshop. Odense, Danemark, pp. 135-149.

O. BOISSIER, P. BEAUNE, H. PROTON, M. HANNOUN, T. CARRON, L. VERCOUTER et C. SAYETTAT (1998), « *The Multi-Agent System Toolkit* ». SIC/ENSM-SE. Rapport technique.

O. BOISSIER (1999), « *Approche Voyelles AEIO* ». DEA Coopération et Communication dans les Systèmes à Agents, Université de Savoie, École des Mines de Saint-Étienne, France.

G. BOOCH (1994), « *Object-Oriented Analysis and Design with Applications* ». Addison-Wesley.

G. BOOCH, I. JACOBSON, J. RUMBAUGH et al. (1997), « *Unified Modeling Language Specifications - version 1.1* ». UML consortium - Object Management Group. Rapport technique.

C. BORKOWF (2000), « A new nonparametric method for variance estimation and confidence interval construction for Spearman's rank correlation ». *Computational Statistics and Data Analysis*, 34, pp. 219-241.

C. BOURNEZ (2001), « *Une architecture multi-agents réflexive pour le contrôle*

de systèmes de production distribués et hétérogènes ». Thèse de doctorat, Institut National des Sciences Appliquées, Lyon.

P. Boutin (2001), « *Définition d'une méthodologie de mise en œuvre et de prototypage d'un progiciel de gestion d'entreprise (ERP)* ». Thèse de doctorat, École Nationale Supérieure des Mines de Saint-Étienne.

C. Brassac (1993), « Théorie des actes de langage et intelligence artificielle distribuée ». In *Journées SMA du PRC-GDR-IA*.

M. Bratman (1987), « *Intentions, Plans, and Practical Reason* ». Havard University Press.

M. Bratman, D. Israel et M. Pollack (1988), « Plans and resource-bounded practical reasoning ». *Computational Intelligence*, 4, pp. 349-355.

R. A. Brooks (1986), « A robust layered control system for a mobile robot ». *IEEE Journal of Robotics and Automation*, RA-2(1), pp. 14-23.

R. A. Brooks (1991a), « Intelligence without representation ». *Artificial Intelligence*, 47(1-3), pp. 139-159.

R. A. Brooks (1991b), « Intelligence without reason ». In *12th International Joint Conference on Artificial Intelligence*, R. Myopoulos et J. Reiter (Ed.). Sydney, Australia, pp. 569-595.

J. Browne, P. Sackett et J. Wortmann (1995), « Future Manufacturing systems – Toward the extended enterprise ». *Computer in industry*, 25, pp. 235-254.

R. Bryant (1979), « Simulation on a Distributed System ». In *First International Conference on Distributed Computing Systems (ICDCS)*. Alabama, USA, pp. 544-552.

R. Burkhart (1994), « The Swarm Multi-Agent Simulation System ». In *OOPSLA Workshop on "The Object Engine"*.

P. Burlat (1996), « *Contribution à l'Évaluation Économique des Organisations Productives : vers une modélisation de l'entreprise-compétences* ». Thèse de doctorat, Université Lyon 2.

P. Burlat, S. Peillon et L. Vincent (1998), « Quels modèles pour une firme sans frontières ? ». In *2ème Congrès International Franco-Québecois de Génie Industriel : Le génie industriel dans un monde sans frontières*.

F. Butera (1991), « *La métaphore de l'organisation : du château au réseau* ». Éditions d'organisation, Les nouveaux modèles d'entreprise : vers l'entreprise réseau.

CACI (2001), « *Simscript II.5* ». `http://www.caciasl.com/simscript.cfm`.

J. Campbell et M. D'Inverno (1990), « Knowledge Interchange Protocols ». In *First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, Y. Demazeau et J. P. Müller (Ed.). Amsterdam, pp. 63-80.

T. CARRON, H. PROTON et O. BOISSIER (1999), « A Temporal Agent communication language for dynamic Multi-Agent Systems ». In *Modelling Autonomous Agents in a Multi-Agent World*. Spain, pp. 115-127.

J. CARSON (1993), « Modeling and Simulation World Views ». In *Winter Simulation Conference*, G. EVANS, M. MOLLAGHASEMI, E. RUSSEL et W. BILES (Ed.), pp. 18-23.

CEN (1995), « *ENV 12 204 : Advanced Manufacturing Technology. Systems Architecture. Constructs for Enterprise Modelling* ». CEN. Rapport technique, Bruxelles.

B. CHAIB-DRAA (1994), « Distributed Artificial Intelligence : an overview ». In *Encyclopedia of Computer Science and Techonlogy*, A. KEN, J. WILLIAMS, C. HALL et R. KENT (Ed.), pp. 215-243.

B. CHAIB-DRAA (1995), « Industrial applications of distributed AI ». *Communications of the ACM*, 38(11), pp. 49-53.

B. CHAIB-DRAA (1996), « Interaction between agents in routine, familiar and unfamiliar situations ». *International Journal of Experimental and Theorical AI (JETAI)*, 1(5), pp. 7-20.

B. CHAIB-DRAA et D. VANDERVEKEN (1998), « Agent Communication Language : A Semantics Based on the Success, Satisfaction and Recursion ». In *Agent Theories, Architectures and Languages*.

B. CHAIB-DRAA, I. JARRAS et B. MOULIN (2001), « *Systèmes multi-agents : principes généraux et applications* ». Hermès, Agent et systèmes multi-agents. À paraître.

K. M. CHANDY et J. MISRA (1979), « Distributed Simulation : A Case Study in Design and Verification of Distributed Programs ». *IEEE Transaction on Software Engeneering*, 5(5), pp. 440-452.

K. M. CHANDY et J. MISRA (1981), « Asynchronous Distributed Simulation Via a Sequence of Parallel Computations ». *Communications of the ACM*, 24(11), pp. 198-206.

K. M. CHANDY et J. MISRA (1988), « *Parallel Program Design : A Foundation* ». Addison-Wesley.

P. CHEN (1976), « The Entity-Relationship Model, Towards a Unified View of Data ». *Communications of the ACM, Transaction on Database systems*, 1(1), pp. 9-36.

G. CHEN et B. K. SZYMANSKI (2001), « Component-Based Simulation ». In *15th European Simulation Multiconference*, E. KERCKHOFFS et M. SNOREK (Ed.). Pragues, pp. 68-75.

P. CHEVAILLIER, R. QUERREC et J. TISEAU (1997), « Modélisation Multi-agents d'une Cellule de Production ». In *2ème Colloque National de Productique*. Casablanca - Maroc, pp. 1-10.

W. K. Ching (2001), « *Iterative Methods for Queuing and Manufacturing Systems* ». Springer Monographs in Mathematics.

T. Christiansen et N. Torkington (2000), « *Perl en action* ». O'Reilly.

L. Cloutier (1999), « *Une approche multi-agents par conventions et contrats pour la coordination de l'entreprise manufacturière réseau* ». Thèse de doctorat.

P. Cohen et C. Perrault (1979), « Elements of plan based theory of pseech acts ». *Cognitive sciences*, 3, pp. 177-212.

P. Cohen et H. Levesque (1990), « Intention is choice with commitment ». *Artificial Intelligence*, 42, pp. 213-261.

P. Cohen et H. Levesque (1995), « Communicative Actions for Artificial Intelligence ». In *1st International Conference on Multi-Agent Systems (ICMAS)*, V. Lesser (Ed.). San Francisco.

D. Coleman (1994), « *Object-oriented Development : The Fusion Method* ». Prentice-Hall.

D. Corkill et V. Lesser (1983), « The distributed vehicule monitoring testbed : a tool for investigating distributed problem solving network ». *AI Magazine*, 3(4), pp. 15-33.

A. Couture et G. Loussararian (1999), « L'entreprise se transforme : de l'organisation mécanique et figée à l'oganisation réactive et vivante ». *Revue Française de Gestion Industrielle*, 18(2).

O. Damani et V. K. Garg (1998), « Fault-Tolerant Distributed Simulation ». In *12th Parallel and Distributed Simulation Workshop*. Banff, Alberta - Canada, pp. 38-45.

R. David et H. Alla (1992), « *Petri Nets and Grafcet – Tools for Modeling Discrete Event Systems* ». Prentice Hall.

J. De Rosnay (1975), « *Le macroscope* ».

K. S. Decker et V. R. Lesser (1993), « Quantitative modeling of complex environments ». *International Journal of Intelligent Systems in Accounting, Finance, and Management*, 2(4), pp. 215-234.

Y. Demazeau (1995), « From Interactions to Collective Behaviour in Agent-Based Systems ». In *European conference on cognitive science*. Saint-Malo, France.

Y. Demazeau (1997), « Steps Towards Multi-Agent Oriented Programming ». In *1st International Workshop on Multi-Agent Systems (IWMAS)*. Boston.

R. Descartes (1637), « *Discours de la méthode* ».

J. Devore (1995), « *Probability and Statistics for Engeneering and the Sciences* ». Wadsworth Inc.

L. DEVROYE (1985), « *Non-Uniform Random Variable Generation* ». Springer-Verlag.

DEVS-SG (2001), « *The Discrete-Event Systems Specifications (DEVS) Study Group* ». `http://www.sce.carleton.ca/faculty/wainer/devsgroup.htm`.

E. W. DIJKSTRA et C. S. SCHOLTEN (1980), « Termination Detection of Diffusing Computation ». *Informations Processing Letter*, pp. 217-219.

G. DOUMEINGTS (1984), « *Méthode GRAI : méthode de conception des systèmes productiques* ». Thèse de doctorat, Laboratoire d'Automatique et de Productique, Université Bordeaux I.

H. DREYFUS (1979), « *What computer can't do : a critique of artificial reason* ». Harper.

A. DROGOUL (1993), « *De la simulation multi-agents à la résolution collective de problèmes* ». Thèse de doctorat, Université Paris 6.

A. DROGOUL et J. GERBER (1994), « Multi-agent simulation as a tool for modeling societies : application to social differentiation in ant colonies ». In *Artificial Social Systems*, C. CASTELFRANCHI et E. WERNER (Ed.). Heidelberg, Allemagne, pp. 3-23.

L. DUPONT (1998), « *La gestion industrielle* ». Hermès.

D. DURAND (1975), « La systèmique ». In *Que sais-je ?*

T. DUVAL, S. MORVAN, P. REIGNIER, F. HARROUET et J. TISSEAU (1997), « ARéVi : Une Boîte à Outils 3D Pour Des Applications Coopératives ». *"La coopération"*, 9(2), pp. 239-250.

A. EL MHAMEDI, C. LERCH et M. SONNTAG (1997), « Modélisation des activités et des processus des systèmes de production : une approche interdisciplinaire ». *RAIRO - APII - JESA*, 31(4), pp. 669-693.

E. ENGELMORE et T. MORGAN (1988), « *Blackboard Systems* ». Addison-Wesley.

A. ERKOLLAR et A. FELFERNIG (1999), « Using UML as domain specific modeling-language for the construction and maintenance of simulation models ». In *Simulation in Industry - 11th European Simulation Symposium*, G. HORTON, D. MÖLLER et U. RÜDE (Ed.). Erlangen - Germany, pp. 31-35.

D. ETTINGHOFFER (1992), « *L'entreprise virtuelle ou les nouveaux modes de travail* ». Éditions Odile Jacob.

R. FAGIN, I. HALPERN et Y. MOSES (1995), « *Reasoning about knowledge* ». the MIT Press.

J. FERBER (1995), « *Les Systèmes Multi-Agents - Vers Une Intelligence Collective* ». InterEditions.

J. Ferber et O. Gutknecht (1998), « A meta-model of the analysis and design of organizations in multi-agent systems ». In *3rd International Conference on Multi-Agent systems (ICMAS)*, pp. 128-135.

R. Fikes et N. Nilsson (1971), « STRIPS : a new approach to the application of theorem proving to problem solving ». *Artificial Intelligence*, 2(3/4), pp. 189-208.

J.-M. Filloque (1992), « *Synchronisation Répartie sur une Machine Parallèle à Couche Logique Reconfigurable* ». Thèse de doctorat, Institut de Formation Supérieure en informatique et Communication - Université de Rennes 1.

T. Finin, R. Fritzon, D. McKay et R. McEntire (1994), « KQML as an Agent Communication Language ». In *3rd International Conference on Information and Knowledge Management*.

FIPA (1998), « *Agent Communication Language* ». Foundation for Intelligence Physical Agent. Rapport technique.

FIPA (2001a), « *FIPA Contract Net Interaction Protocol Specification* ». Foundation for Intelligence Physical Agent. Rapport technique.

FIPA (2001b), « *The official site of the Foundation for Intelligent Physical Agents (FIPA)* ». `http://www.fipa.org/`.

R. Foisel (1998), « *Modèle de réorganisation de systèmes multi-agents : une approche descriptive et opérationnelle* ». Thèse de doctorat, Université Henry Poincaré, Nancy I.

M. Fox et M. Gruninger (1995), « Enterprise Modelling ». In *AI Magazine*. Fall, pp. 109-121.

S. Fujii, A. Ogita, Y. Kidani et T. Kaihara (1999), « Synchornization Mechanisms for Integration of Distributed Manufacturing Simulation Systems ». *Simulation*, 72(1), pp. 187-197.

R. Fujimoto (1990a), « Parallel Discrete Event Simulation ». In *ACM*.

R. Fujimoto (1990b), « Optimistic Approachs to Parallel Discrete Event Simulation ». *Transactions of the Society for Computer Simulation*, 7(2), pp. 152-191.

R. Fujimoto (1993), « Parallel Discrete Event Simulation : Will the Field Survive ? ». *ORSA Journal of Computing*, 5(3).

A. Futschik et G. Pflug (1995), « Confidence sets for discrete stochastic optimization ». *Annals of Operations Research*, 56, pp. 95-108.

J. Gabay (1998), « *Merise, vers OMT et UML* ». InterÉditions, Masson.

A. Gafni (1988), « Roolback Mechanisms for Optimistic Distributed Simulation Systems ». In *SCS Multi-Conference on Distributed Simulation*. San Diego, USA, pp. 61-67.

S. GALLAND (1998), « *Classification de propriétés dynamiques et de leurs raffinements à partir de quelques études de cas* ». Mémoire scientifique, Laboratoire d'Informatique de Besançon, Besançon, France.

S. GALLAND, F. GRIMAUD, P. BEAUNE et J.-P. CAMPAGNE (1999), « Multi-Agent Methodological Approach for Distributed Simulation ». In *Simulation in Industry - 11th European Simulation Symposium*, G. HORTON, D. MÖLLER et U. RÜDE (Ed.). Erlangen - Germany, pp. 104-108.

S. GALLAND et F. GRIMAUD (2000a), « Methodological approach for distributed simulation : Life cycle of $\mathcal{MAS}$-$\mathcal{S}$ ». In *ASIM-workshop 20/21.3 2000 - Multiagent-systems and Individual-based simulation*, F. KLÜGL, F. PUPPE, P. SCHWARZ et H. SZCZERBICKA (Ed.). Institut für Informatik. Würzburg, Germany, pp. 83-93.

S. GALLAND, F. GRIMAUD et J.-P. CAMPAGNE (2000b), « Methodological approach for distributed simulation : General concepts for $\mathcal{MAS}$-$\mathcal{S}$ ». In *Simulation and Modelling : Enablers for a better quality of life – 14th European Simulation Multiconference*, R. VAN LANDEGHEM (Ed.). Ghent, Belgium, pp. 77-82.

S. GALLAND, F. GRIMAUD et J.-P. CAMPAGNE (2000c), « Multi-agent architecture for distributed simulation : Teaching application for industrial management ». In *Simulation and Modelling : Enablers for a better quality of life – 14th European Simulation Multiconference*, R. VAN LANDEGHEM (Ed.). Ghent, Belgium, pp. 756-762.

S. GALLAND, F. GRIMAUD et J.-P. CAMPAGNE (2001a), « $\mathcal{MAS}$-$\mathcal{S}$: Phase de spécification d'un modèle de simulation pour un système industriel distribué ». In *3ème conférence francophone de MOdélisation et SIMulation*, A. DOLGUI et F. VERNADAT (Ed.). Troyes, France, pp. 573-580.

S. GALLAND, F. GRIMAUD, P. BEAUNE et J.-P. CAMPAGNE (2001b), « Méthodologie pour la simulation de systèmes industriels complexes distribués au travers d'une étude de cas ». In *4ème Congrès International de Génie Industriel*, J.-C. BERTRAND et J.-P. KIEFFER (Ed.). Aix-en-Provence, Marseille, Ajaccio, pp. 673-684.

S. GALLAND, F. GRIMAUD et J.-P. CAMPAGNE (2001c), « How to specify an simulation model of an industrial system in $\mathcal{MAS}$-$\mathcal{S}$ ». In *Modelling and Simulation - 15th European Simulation Multiconferences*, E. KERCKHOFFS et M. SNOREK (Ed.). CTU Prague - Czech Republic, pp. 296-305.

S. GALLAND, F. GRIMAUD, P. BEAUNE et J.-P. CAMPAGNE (2001d), « A study case with a methodological approach for complex and distributed industrial system simulation ». In *International Conference on Industrial Engineering and Production Management*. Québec, Canada.

J. GALLIERS (1988), « *A theorical framework for computer models of cooperative dialogue, acknoledging multi-agent conflict* ». Thèse de doctorat, Open university, England.

L. GASSER, N. ROUQUETTE, R. HILL et J. LIEB (1989), « Representing and

Using Organizational Knowledge in DAI Systems ». *Distributed Artificial Intelligence*, pp. 55-79.

L. Gasser (1992), « An overview of DAI ». In *Distributed Artificial Intelligence, Theory and Praxis*, L. Gasser et N. Avouris (Ed.).

M. Georgeff (1988), « Communication and interaction in multi-agent planning ». In *Readings in Distributed Artificial Intelligence*, A. H. Bond et L. Gasser (Ed.), pp. 200-204.

S. Gerhart (1991), « Formal Methods : An International Perspective ». In *13th International Conference on Software Engineering*, pp. 36-37.

C. Ghezzi, M. Jazayeri et D. Mandrioli (1991), « *Fundamentals of software engeneering* ». Prentice-Hall International.

D. Goldsman et L. Schruben (1990), « New confidence interval estimators using standardized time series ». *Management Science*, 36, pp. 393-397.

T. Goranson, M. Johnson, A. Presly et H. Rogers (1997), « Metrics for the agile virtual enterprise case study ». In *6th Annual National Agility Conference.*

M. Gourgand (1984), « *Outils Logiciels Pour L'évaluation Des Performances Des Systèmes Informatiques* ». Thèse de doctorat, Université Blaise Pascal, Clermont-Ferrand, France.

F. Grimaud (1996), « *Conception D'une Base de Composants Logiciels Pour L'évaluation Des Performances Des Entreprises Manufacturières* ». Thèse de doctorat, Université Blaise Pascal, Clermont-Ferrand.

GRP GT 5 (1998), « La modélisation d'entreprise : le point de vue productique ». Action 8 «document de référence», version 1.1.

GRP GT 5 (2000), « Glossaire pour la modélisation d'entreprise ».

O. Gutknecht, J. Ferber et F. Michel (2000), « MadKit : Une expérience d'architecture de plate-forme multi-agent générique ». In *Journées Francophones IAD et SMA (JFIADSMA)*, S. Pesty et C. Sayettat-Fau (Ed.). Saint-Étienne, pp. 223-236.

J. Guttag et J. Horning (Ed.) (1995), « *Larch : Languages and Tools for Formal Specification* ». Springer-Verlag.

M. Hannoun, J. S. a. Sichman, O. Boissier et C. Sayettat (1998), « Dependence Relations Between Roles in a Multi-Agent System ». In *Multi-agent Systems and Agent-based Simulation, Lecture Notes in artificial Intelligence*, J. Sichman, R. Conte et N. Gilbert (Ed.). Berlin-Alemanha, pp. 169-182.

M. Hannoun, O. Boissier, J. S. Sichman et C. Sayettat (2000), « MOISE : An organizational Model for Multi-agent Systems ». In *Advances in Artificial Intelligence*, M. Monard et J. Sichman (Ed.). Brazil, pp. 156-165.

M. HANNOUN (2002), « MOISE *: un modèle organisationnel pour les systèmes multi-agents* ». Thèse de doctorat, École Nationale Supérieure des Mines, Saint-Étienne. À paraître.

B. HEULUY et F. VERNADAT (1997), « The CIMOSA Enterprise Ontology ». In *IFAC Worksop - MIM'97*. Vienna.

C. HEWITT (1977), « Viewing control structures as patterns of passing messages ». *Artificial Intelligence*, 8(3), pp. 323-364.

R. HOGG et A. CRAIG (1994), « *Introduction to mathematical statistics* ». Macmillan.

P. INGELS et M. RAYNAL (1990), « Simulation Répartie de Systèmes à Événements Discrets ». *TSI*, 9(5), pp. 383-397.

G. JACOB (1995), « *La refonte des systèmes d'information* ». Éditions Hermès.

D. JEFFERSON (1985a), « Virtual Time ». *ACM Transaction on Programming Languages and Systems*, 7(3), pp. 404-425.

D. JEFFERSON et H. SOWIZRAL (1985b), « Fast Concurrent Simulation Using the Time Warp Mechanism ». In *SCS Multi-Conference on Distributed Simulation*. San Diego, USA, pp. 63-69.

N. JENNINGS, K. SYCARA et M. WOOLRIDGE (1998a), « A roadmap of agent reseach and development ». *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1), pp. 7-38.

N. JENNINGS et M. WOOLRIDGE (Ed.) (1998b), « *Agent technology : foundations, applications and markets* ». Springer-Verlag.

N. JENNINGS (2000), « Agent methodology for software engineering ». *Communication of ACM*. À paraître.

B. JULLIEN (1991), « *Simulation des Systèmes de Production, tome 1* ». École Nationale Supérieure des Mines. Document pédagogique interne.

N. KABACHI (1999), « *Modélisation et Apprentissage de la prise de décision dans les organisations productives* ». Thèse de doctorat, Université Jean Monnet / ENSM.SE, Saint Etienne, France.

P. KELLERT (1992), « Définition et Mise En Oeuvre D'une Méthodologie Orientée Objets Pour la Modélisation Des Systèmes de Production ». In *Actes Du Congrès INFORSID*. Clermont-Ferrand, France, pp. 415-436.

P. KELLERT et C. FORCE (1998a), « Méthodologie de Modélisation Orientée Objets de Systèmes de Production – Application à nne Chaîne d'Étuvage de Bobines d'Allumage ». *Journal Européen des Systèmes Automatisés*, 32(1), pp. 107-131.

P. KELLERT et S. RUCH (1998b), « Méthodologie de Modélisation Orientée Objets de Systèmes de Production – Un Processus de Construction/Validation du Modèle

Générique Orienté Objets d'un Système de Production ». *Journal Européen des Systèmes Automatisés*, 32(1), pp. 51-105.

W. Kelton, R. P. Sadowski et D. A. Sadowski (1998), « *Simulation with Arena* ». McGraw-Hill.

D. Kesser (1997), « *La représentation des connaissances* ». Hermès.

R. Kilmer, A. Smithg et L. Schuman (1999), « Computing confidence intervals for stochastic simulation using neural network metamodels ». *Computers & Industrial Engineering*, 36, pp. 391-407.

D. E. Knuth (1998), « *The Art of Computer Programming* ». Addison-Wesley.

A. Kobsa (1989), « *User models in dialog systems* ». Springer-Verlag.

K. Kosanke et J. Nell (Ed.) (1997), Actes « *Enterprise Engineering and Integration : Building Intenrational Consensus* », Berlin, Springer-Verlag.

E. Laizé (1998), « *Méthodologie de Modélisation de Systèmes Manufacturiers de Pneumatiques : Spécification de la Connaissance* ». Thèse de doctorat, Université Blaise Pascal.

S. Lardon, P. Bommel, F. Bousquet, C. Le Page, T. Libourel, R. Lifran et P.-L. Osty (2000), « De la simulation d'une dynamique d'embrousaillement à un outil d'aide à la gestion de l'espace pastoral - un modèle de conception des transformations de l'espace ». In *Journées Francophones IAD et SMA (JFIADSMA)*, S. Pesty et C. Sayettat-Fau (Ed.). Saint-Étienne, pp. 357-374.

B. Latour (1989), « *La science en action* ». La Découverte.

A. Law et W. Kelton (1991), « *Simulation modeling and analysis* ». McGraw Hill.

J.-L. Le Moigne (1977), « *La théorie du système général. Théorie de la modélisation* ». Presses Universitaires de France.

J.-L. Le Moigne (1992), « *La modélisation des systèmes complexes* ». Editions Dunod.

J. Lee, M. Gruninger, Y. Jin, T. Malone, A. Tate et G. Yost (1998), « PIF : The Process Interchange Format ». In *Handbook of information systems*, P. Bernus, K. Mertins et G. Schmidt (Ed.). Berlin, pp. 209-241.

Y. Lespérance (1991), « *A formal theory of indexical knowledge and action* ». Thèse de doctorat, University of Toronto.

D. Lestel (1986), « *Contribution à l'étude du raisonnement expérimental dans un domaine sémantiquement riche* ». Thèse de doctorat, EHESS.

Y.-B. Lin et P. A. Fishwick (1996), « Asynchronous Parallel Discrete Event Simulation ». *IEEE transactions on systems, man and cybernetics*, 26(4).

M. LISSANDRE (1990), « *Maîtriser SADT* ». Colin.

N. LOPEZ, J. MIGUEIS et E. PICHON (1998), « *Intégrer UML dans vos projets* ». Eyrolles.

B. LUBACHEVSKY, A. SHWARTZ et A. WEISS (1989), « Roolback Sometimes Works ... If Filtered ». In *1989 Winter Simulation Conference*. Washington, USA, pp. 630-639.

V. MADISETTI, J. WALRAND et D. MESSERSCHMITT (1988), « WOLF : A Roolback Algorithm for Optimistic Distributed Simulation ». In *1988 Winter Simulation Conference*. San Diego, USA, pp. 296-305.

L. MAGNIN (1996), « *Modélisation de l'environnement dans les systèmes multi-agents (applications aux robots footballeurs)* ». Thèse de doctorat, Université Paris 6.

B. MAGNUSSON (1994), « *SIMULA runtime system overview in object-oriented environments : the Mjolner approach* ». Prentice-Hall, 25, pp. 378-387.

F. MARCOTTE (1995), « *Contribution à la modélisation des systèmes de production : extension du modèle GRAI* ». Thèse de doctorat, LAP, Université Bordeaux I.

R. J. MAYER (1992), « *Information Modeling (IDEF1) – A reconstruction of the original Air Force wright aeronautical laboratory technical report* ». Knowledge Based Systems, Inc.. Rapport technique AFWAL-TR-81-4023, College Station, Texas, USA.

H. MEHL (1991), « Speed-Up of Conservative Discrete Event Simulation Methods by Speculative Computing ». In *SCS Multi-Conference on Avances in Parallel and Dsitributed Simualtion*. Anaheim, USA, pp. 163-166.

H. MERTINS, H. EDELER, R. JOCHEM et J. HOFMANN (1995), « Object-oriented modelling and analysis of business processes ». In *Integrated Manufacturing Systems Engineering*, P. LADET et F. VERNADAT (Ed.). London, pp. 115-128.

M. MEZAROVIC, D. MACKO et T. TAKAHARA (1970), « *Theory of hierarchical, multilevel systems* ». Academic Press.

MICROSOFT® CORPORATION (1998), « *Microsoft® Visual Basic® 5.0 – Langages et Contrôles ActiveX* ». Microsoft® Press.

C. MORGAN (1989), « A survey of MS/OR surveys ». *Interfaces*, 19, pp. 95-103.

B. MOULIN et B. CHAIB-DRAA (1996), « An overview of distributed artificial intelligence ». In *Foundations of Distributed AI*, G. O'HARE et N. JENNINGS (Ed.). Chichester, England, pp. 3-54.

T. MOWBRAY et R. ZAHAVI (1995), « *The Essential CORBA* ». John Wiley & Sons.

P.-A. MULLER (1997), « *Modélisation objet avec UML* ». Eyrolles.

R. Nance (1981), « *Model Representation in Discrete Event Simulation : The Conical Methodology* ». Department of Computer Science, Virginia Polytechnic Institute and State University. Rapport technique CS-81003-R, Blackburg, USA.

R. Nance (1994a), « The Conical Methodology and the Evolution of Simulation Model Development ». *Annals of Operations Research*, 53, pp. 1-45.

R. Nance (1994b), « Conical Methodology : An Evolutionary Convergence of Systems and Software Engineering ». *Annals of operations research*, 53.

A. Negroponte (1995), « *Being digital* ». Hodder and Stoughton.

A. Newell, J. Shaw et H. Simon (1963), « Empirical explorations of the logic theory machine : a case study in heuristics ». In *Computers and Thought*, E. Feigenbaum et J. Feldman (Ed.), pp. 109-133.

NIST (2001), « *Process Specification Language* ». `http://www.mel.nist.gov/psl/`.

T. Norman et D. Long (1996), « Alarms : an implementation of motivated agency ». In *Intelligent Agents II (LNAI 1037)*, M. Woolridge, J. Müller et M. Tambe (Ed.). Heidelberg, Allemagne, pp. 21-234.

P. Nunes (1994), « *Formes PME et ogranisation en réseaux* ». INIST-CNRS.

H. Nwana et D. Ndumu (1999), « Agents of change in future communication systems ». *Applied AI Journal*.

H. Pattison, D. Corkill et V. Lesser (1987), « Instantiating descriptions of organizational structures. ». In *Distributed Artificial Intelligence*, M. Huhns (Ed.). Londre & San Mateo (CA), pp. 56-96.

C. D. Pegden, R. E. Shannon et R. P. Sadowski (1995), « *Introduction to simulation using SIMAN* ». McGraw-Hill.

H. Pierreval (1990), « *Les méthodes d'analyse et de conception des systèmes de production* ». Éditions Hermès.

P. Populaire, Y. Demazeau, O. Boissier et J. Sichman (1993), « Description et Implémentation de Protocoles de Communication En Univers Multi-Agents ». In *Premières Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents*.

D. Poulin, B. Montreuil et S. Gauvin (1994), « *L'entreprise réseau : bâtir aujourd'hui l'ogranisation de demain* ». Publi-Relais.

H. Proton (2002), « *Coordination des conversations dans un système multi-agents* ». Thèse de doctorat, École Nationale Supérieure des Mines, Saint-Étienne. À paraître.

A. S. Rao et M. P. Georgeff (1995), « DBI agents : from theory to practice ». In *1st International Conference on Multi-Agent Systems (ICMAS)*, V. Lesser (Ed.).

San Francisco, pp. 312-319.

J. RASMUSSEN et T. GEORGE (1978), « After 25 years : a survey of operations research alumni, case western reserve university ». *Interfaces*, 8, pp. 48-52.

P. REIHER (1990), « Parallel Simulation Using the Time Warp Operating System ». In *1990 Winter Simulation Conference*. New Orleans, USA, pp. 38-45.

P. REYNOLDS (1988), « A Spectrum of Options for Parallel Simulation ». In *1988 Winter Simulation Conference*. San Diego, USA, pp. 325-332.

P. REYNOLDS et P. DICKENS (1990), « SRADS with Local Roolback ». In *SCS Multi-Conference on Distributed Simulation*. San Diego, USA, pp. 161-164.

A. ROLSTADAS (1995), « Enterprise Modelling for Competitive Manfucaturing ». *Control Engineering Practice*, 1(1), pp. 43-50.

J. ROSENSCHEIN et M. GENESERETH (1985), « Delas Among Rational Agents ». In *9th International Joint Conference on Artificial Intelligence (IJCAI)*. Los Angeles, CA, pp. 91-99.

J. RUMBAUGH et al. (1991), « *Object Oriented Modeling and Design* ». Prentice Hall.

D. SADEK (1991), « *Attitudes mentales et interaction rationnelle : vers une théorie formelle de la communication* ». Thèse de doctorat, Université Rennes I.

B. SAMADI (1985), « *Distributed Simulation, Performance and Analysis* ». Thèse de doctorat, University of California, Los Angeles, USA.

V. SANDOVAL (1994), « *Les techniques de reengineering* ». Éditions Hermès.

SANTA FEE INSTITUTE (1994), « *The Swarm Multi-Agent Simulation System* ». Document non publié.

J. SEARLE (1969), « *Speech acts : an essay in the philosophy of language* ». Cambridge University Press.

J. SEARLE (1972), « *Les actes de langage* ».

J. SEARLE (1991), « *The rediscovery of the mind* ». The MIT Press.

R. SHANNON, S. LONG et B. BUCKLES (1980), « Operations Research Methodologies in Industrial Engineering ». *AIIE Trans*, 12, pp. 364-367.

Y. SHOHAM (1993a), « Agent-oriented programming ». *Artificial Intelligence*, 60(1), pp. 51-92.

Y. SHOHAM et M. TENNENHOLTZ (1993b), « On social Laws for Artificial Agent societies ». *Artificial Intelligence*, 73.

J. S. SICHMAN, Y. DEMAZEAU, R. CONTE et C. CASTELFRANCHI (1994), « Un Mécanisme de Raisonnement Social Fondé sur des Réseaux de Dépendance ». In *IA*

Distribuée et Systèmes Multi-Agents – 2ème JFIADSMA. Voiron, pp. 131-142.

J. S. Sichman et Y. Demazeau (1995), « Exploiting social reasoning to deal with agency level inconsistency ». In *First International Conference on Multi-Agent Systems (ICMAS)*. San Francisco, pp. 352-359.

J. Siegel (1996), « *CORBA - Fundamentals and Programming* ». John Wiley & Sons.

R. G. Smith (1980), « The Contract Net Protocol : High-Level Communication and Control in a Distributed Problem Solder ». *IEEE Transaction on computers*, C-29(12), pp. 1104-1113.

L. Sokol, D. Briscoe et A. Wieland (1988), « MTW : Strategy for Scheduling Discrete Event Simulation Events for Concurrent Execution ». In *SCS Multi-Conference on Distributed Simulation*. San Diego, USA, pp. 34-43.

L. Sokol, B. Stucky et V. Hwang (1989), « MTW : A Control Mechanism for Parallel Discrete Simulation ». In *1989 International Conference on Parallel Processing*. Pen State university, Pennsylvanie, USA, pp. 250-254.

I. Sommerville (1998), « *Le Génie Logiciel et ses applications* ». InterEditions.

G. Spur, K. Mertins et R. Jochem (1996), « *Integrated Enterprise Modelling* ». Springer-Verlag.

Y. M. Tan (1994), « *Formal Specification Techniques for Promoting Software Modularity, Enhancing Documentation, and Testing Specifications* ». Thèse de doctorat, MIT/LCS/TR-619.

H. Tardieu, A. Rochfeld et R. Colleti (1985), « *La Méthode MERISE - Tome I : Principes et Outils* ». Edition d'Organisation.

Tecnomatix® (2001), « Simple++® ». `http://www.tecnomatix.com/`.

The RAISE Language Group (Ed.) (1992), « *The RAISE Specification Language* ». Prentice Hall International.

The RAISE Method Group (Ed.) (1995), « *The RAISE Development Method* ». Prentice Hall International.

G. Thomas et J. DaCosta (1979), « A sample survey of corporate operations research ». *Interfaces*, 9, pp. 102-111.

S. Thomas (1993), « *PLACA, an agent oriented programming language* ». Thèse de doctorat, computer Science Departement of Standford Univsersity.

K. Trivedi (2001), « *Probability and Statistics with Reliability, Queuing, and Computer Science Applications* ». Wiley.

US Air Force (1993a), « *Integrated Computer Aided Manufactured Definition Language (IDEF methods)* ». Department of Commerce, National Institute of Stan-

dards and Technology, Computer Systems Laboratory. Rapport technique, Gaithersburg, USA.

US AIR FORCE (1993b), « *Integration Definition for Function Modeling (IDEF0)* ». Department of Commerce, National Institute of Standards and Technology, Computer Systems Laboratory. Rapport technique, Gaithersburg, USA.

US AIR FORCE (1993c), « *Process Flow and Object State Description Capture Method (IDEF3)* ». Department of Commerce, National Institute of Standards and Technology, Computer Systems Laboratory. Rapport technique, Gaithersburg, USA.

US DEPARTMENT OF DEFENSE (1996), « *High Level Architecture Federation Development and Execution Process (FEDEP) Model, version 1.0* ». Defense Modeling and simulation Office. Rapport technique.

M. VERAN et D. POTIER (1984), « QNAP 2 : A Portable Environment for Queueing System Modelling ». In *Colloque international sur la modélisation et les outils d'analyse de performance*. Paris.

L. VERCOUTER (2000), « *Conception et mise en œuvre de systèmes multi-agents ouverts et distribués* ». Thèse de doctorat, École Nationale Supérieure des Mines, Saint-Étienne, France.

F. VERNADAT (1996), « *Enterprise Modeling and Integration : Principles and Applications* ». Chapman & Hall.

F. VERNADAT (1997), « Enterprise Modeling Languages ». In *Enterprise Engineering and Integration - International Conference on Enterprise Integration and Modeling Technology*, K. KOSANKE et J. NELL (Ed.), pp. 212-224.

F. VERNADAT (1998), « La modélisation d'entreprise par la méthodologie CIMOSA ». In *Modélisation d'entreprise (CNRS PROSPER «Systèmes de production»)*. Roisy-en-France.

F. VERNADAT (1999), « *Techniques de modélisation en entreprise : application aux processus opérationnels* ». Editions Economica.

F. VERNADAT (2000), « Enterprise Modeling and Integration : Myth or reality ? ». In *2nd Intenrational Conference on Management and Control of Production and Logistic*. Grenoble, France.

F. VERNADAT (2001), « UEML : Towards a Unified Entreprise Modelling Language ». In *3ème conférence francophone de MOdélisation et de SIMulation*, A. DOLGUI et F. VERNADAT (Ed.). Troyes, France, pp. 3-10.

W3 CONSORTIUM (2001), « *eXtended Markup Language* ». `http://www.w3.org/XML/`.

J. WANG (1998), « *Timed Petri Nets, Theory and Application* ». Kluwer Academic Publishers.

G. Weiss (Ed.) (1999), « *Multiagent Systems – A modern Approach to Distributed Artificial Intelligence* ». The MIT Press.

D. Wilkins (1988), « *Practical planning : extending the AI planning paradigm* ». Morgan Kaufmann.

T. Williams (1994), « The Purdue Enterprise Reference Architecture ». *Computers in Industry*, pp. 141-158.

K. R. Wood (1993), « A Practical Approach to Software Engineering Using Z and the Refinement Calculus ». In *ACM SIGSOFT – Symposium on the Foundations of Software Engineering*, pp. 79-88.

M. Woolridge et N. Jennings (1995), « Intelligent agents : theory and practice ». *The Knowledge Engineering Review*, 10(2), pp. 115-152.

X. Ye (1994), « *Modélisation et simulation des systèmes de production : une approche orientée-objets* ». Thèse de doctorat, École Nationale Supérieure des Mines de Saint Etienne.

M. Zelm, F. Vernadat et K. Kosande (1995), « The CIMOSA business modelling process ». *Computers in industry*, 21(2). Special issue : Validation of CIMOSA.

A. Zimmermann (1994), « A modeling method for flexible manufacturing systems based on colored Petri nets ». In *International Workshop on new directions of control and manufacturing*, pp. 147-154.

Cinquième partie

Annexes

Annexe A

Métamodèle UML

Les dernières modifications sur ce métamodèle sont datées du 5 Juin 2001 (version 0.3).

Cette annexe décrit l'ensemble du métamodèle Uml utilisé durant la phase de spécification de $\mathcal{MAMA}$-$\mathcal{S}$. Elle est composée d'un ensemble de bibliothèques correspondant à la modélisation des sous-systèmes composant une entreprise, et d'une bibliothèque plus générale contenant un ensemble de concepts communs aux trois sous-systèmes.

Chaque bibliothèque est décrites grâce au protocole de notation suivant :

- **Syntaxe abstraite :** elle présente un diagramme Uml du métamodèle. Certaines règles de bonne forme y sont exprimée, notamment les multiplicités et les relations entre objets. Cette section se termine par une brève description informelle des éléments du diagramme.
- **Règles de construction :** certaines propriétés sémantiques statiques ne peuvent être exprimées par la syntaxe abstraite. Chaque contrainte est exprimée formellement grâce à la logique des prédicats du premier ordre. Nous n'utilisons pas l'OCL (« Object Constraint Language ») qui est défini dans les spécifications d'Uml [Booch et al., 1997]. Nous pensons que la notation logique est d'un abord plus aisé que l'OCL ;

Les attributs et les méthodes peuvent être associées aux clauses de visibilité suivantes :

- **public :** représentée par ⟨+⟩, cette clause autorise l'accès à l'attribut ou la méthode par n'importe quel objet ;
- **protected :** représentée par ⟨#⟩, restreint l'accès au membre qui lui est associé à l'objet courant et à ses descendants ;
- **privé :** représentée par un ⟨-⟩, n'autorise l'accès qu'à l'objet courant.

Le métamodèle décrit ici est succeptible de fortement évoluer durant le développement de l'approche méthodologique $\mathcal{MAMA}$-$\mathcal{S}$.

A.1 Bibliothèque *Common*

La bibliothèque *Common* contient les objets les plus fondamentaux nécessaire à la mise en œuvre de modèles de simulation. Les constructions comme SIMULATIONELEMENT et REMOTEOBJECT sont communes aux autres bibliothèques de *Simulation*. De plus nous y définissons les concepts de modèles. La section suivante décrit la syntaxe abstraite de la bibliothèque *Common*.

A.1.1 Syntaxe abstraite

La figure A.1 illustre la syntaxe abstraite décrite dans cette section.

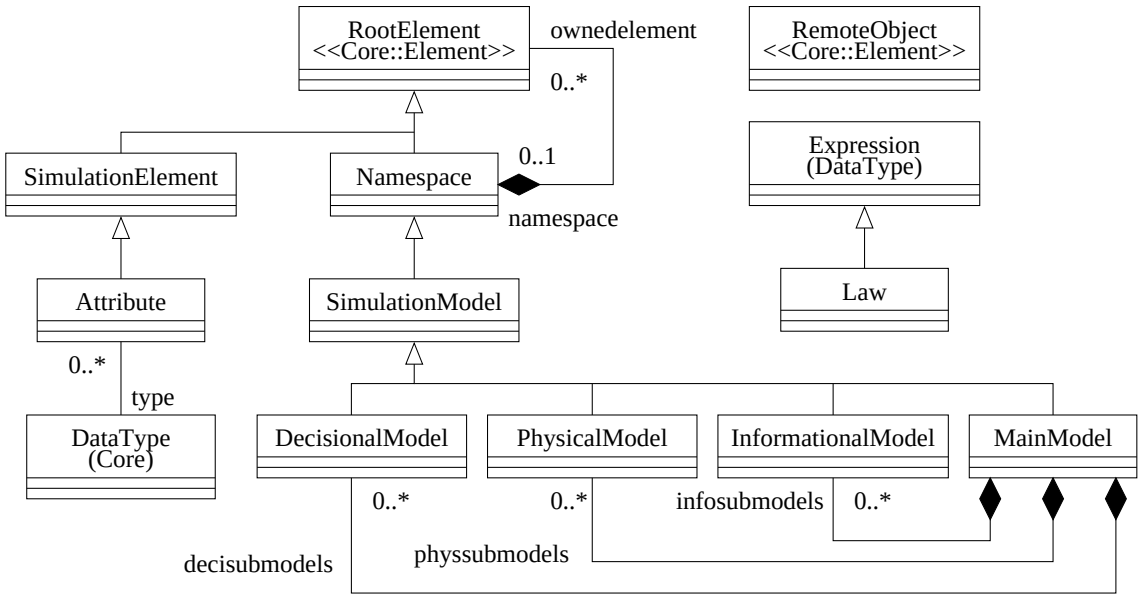


FIG. A.1 – Paquetage *Simulation*::*Common*

ATTRIBUTE

Un attribut est une valeur nommée. ATTRIBUTE associe à l'identificateur hérité *name* une valeur. Cette dernière possède un type particulier représenté par l'association avec DATATYPE. Lors de la construction d'un modèle, la seule valeur intéressante est celle qui est utilisée pour initialiser un attribut.

Attributs

+	initialvalue : Expression		Représente la valeur initiale de l'attribut.

Associations

entity: Correspond à une entité partiellement décrite par l'attribut (figure A.11 page 230).

product: Produit dont l'attribut est un composant de la description (figure A.15 page 250).

type: Désigne le type de la valeur représentée par l'attribut (figure A.1).

DecisionalModel

Le modèle des processus décisionnels représente l'ensemble des algorithmes et des capacités de prise de décisions d'entités au sein d'un système industriel. Chaque instance de cette classe appartient obligatoirement à une instance de la classe MainModel.

Associations

decisionalsubmodel: Un modèle décisionnel compose un modèle global (figure A.1 page précédente).

InformationalModel

Le modèle des flux informationnels est une représentation de l'ensemble des éléments composant le sous-système du même nom. Chaque instance de cette classe appartient obligatoirement à une instance de la classe MainModel.

Associations

informationnalsubmodel: Un modèle informationnelle compose un modèle global (figure A.1 page précédente).

nomenclaturemodel: Correspond au modèle des nomenclatures composant le sous-système informationnel (figure A.13 page 246).

routingmodel: Représente le modèle des gammes (figure A.13 page 246).

Law

Law représente une expression statistique utilisée par les modèles de simulation.

Associations

activatedstate:	Représente l'état d'une unité de production dont l'activation est déterminée par une loi (figure A.8 page 228).
branch:	Désigne le lien de composition entre une loi et un élément de modélisation d'un branchement (figure A.7 page 228).
durationstate:	Correspond à un état d'une unité de production dont la durée de validité est déterminée par une loi (figure A.7 page 228).
generator:	Représente le générateur d'entité qui utilise une loi statistique pour créer celles-ci (figure A.1 page précédente).
PROCESSINGLAW*:*	Désigne les unités de traitement des entités qui utilise des lois pour traiter ces dernières (figure A.11 page 230).
routingstation:	Une loi est utilisée par une station de traitement du modèle des gammes (figure A.17 page 252).

Méthodes

+	sameAs(x : LAW)	Indique si l'expression de la loi courante est équivalente à l'expression de la loi passée en paramètre.

MAINMODEL

Un modèle de simulation représentant un processus industriel doit inclure les modèles des sous-systèmes physiques, informationnels et décisionnels pour être complet. MAINMODEL désigne ce modèle global. Son rôle est de réaliser un lien direct de composition entre les trois autres catégories de modèles.

Associations

decisionalsubmodel:	Représente le sous-système décisionnel inclu dans le modèle courant (figure A.1 page 218).
informationalsubmodel:	Correspond au sous-système informationnel inclu dans le modèle courant (figure A.1 page 218).
physicalsubmodel:	Représente une partie du modèle global modélisant le sous-système physique (figure A.1 page 218).

NAMESPACE

Un espace de nom est une partie d'un modèle dans laquelle chaque nom des éléments de modélisation appartenant à cet espace est unique. NAMESPACE représente l'espace de noms où se trouves des objets de type ROOTELEMENT.

Associations

ownedelement:	Désigne l'ensemble des éléments de modélisation contenus dans l'espace de noms courant (figure A.1 page 218).

PhysicalModel

Le modèle des flux physiques est une représentation de l'ensemble des éléments composant le sous-système physique. Chaque instance de cette classe appartient obligatoirement à une instance de la classe MainModel.

Associations

appartient:	Désigne l'ensemble des composants du modèle de simulation physique (figure A.2 page 225).
defining:	Correspond à la définition d'un sous-modèle par un modèle physique (figure A.5 page 227).
physicalsubmodel:	Un modèle physique compose un modèle global (figure A.1 page 218).

RemoteObject

Un objet distant est une instance existant sur une machine accessible par l'intermédiaire d'un réseau informatique. RemoteObject est une classe abstraite mettant en place une interface pour l'utilisation de ce type d'objets. Afin de permettre cette communication, il est nécessaire de spécifier la localisation de l'objet distant.

Attributs

+	remotelocation : Uninterpreted	Description de la localisation de l'objet distant.

RootElement

Un "élément racine" est un constituant atomique du formalisme permettant de créer des modèles de simulation. Dans le métamodèle, RootElement est la classe abstraite mère de toute la hiérarchie. Chaque élément de modélisation possède un identificateur unique dans son espace de noms.

Attributs

-	name : Name	L'identificateur de l'élément de modélisation au sein de son espace de nom.

Méthodes

+	getName()	Renvoi le nom de l'élément de modélisation *i.e.*, l'attribut *name*.
+	setName()	Change l'identification de l'élément de modélisation *i.e.*, l'attribut *name*.

Associations

namespace: Désigne l'espace de noms dans lequel se trouve l'élément de modélisation. Tous les éléments de modélisation à l'exception de RemoteObject doivent être contenus dans un espace de noms (figure A.1 page 218).

SimulationElement

Un élément de simulation est un composant d'un modèle de simulation des sous-systèmes physiques, informationnels et décisionnels d'une entité industrielle. SimulationElement est une classe abstraite de laquelle doit être dérivés tous les éléments de simulation utilisés pour la construction du modèle de simulation. SimulationElement peut être distribuable *i.e.*, partagé avec des modèles de simulation distants. Par défaut nous considérerons qu'il est utilisable uniquement localement.

Attributs

#	distributable	: Boolean	Permet d'autoriser ou d'interdire le partage du composant au niveau du meta-modèle.
+	shared	: Boolean	Permet d'autoriser ou d'interdire le partage d'un élément de modélisation. Cet attribut peut être modifé lors de l'utilisation du méta-modèle et il est très forte adéquation avec *distributable*.

SimulationModel

Un modèle de simulation est une représentation des sous-systèmes physiques, informationnelles et décisionnels d'une entreprise industrielle. SimulationModel représente l'abstraction commune aux différentes types de modèles utilisés pour définir un processus industriel. Un modèle de simulation peut être partagé avec des modèles distants.

Attributs

#	distributable	: Boolean	Désigne l'autorisation ou l'interdiction de partager un modèle de simulation au niveau du méta-modèle.
+	shared	: Boolean	Permet d'autoriser ou d'interdire le partage d'un élément de modélisation. Cet attribut peut être modifé lors de l'utilisation du méta-modèle et il est très forte adéquation avec *distributable*.

A.1.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

Attribute

[1] Le type de l'attribut *initialvalue* doit être le même que celui associé à Attribute par *type*:
$\forall x / x \in \text{Attribute}, x.\ initialvalue \in x.type$

MainModel

[1] Il doit exister au moins une instance d'un modèle physique, informationnel ou décisionnel dans la composition d'un modèle global MainModel:
$\forall x / x \in \text{MainModel},$
$\text{card}(x.physicalsubmodel) > 0 \vee$
$\text{card}(x.decisionalsubmodel) > 0 \vee$
$\text{card}(x.informationalsubmodel) > 0$

NameSpace

[1] Le nom d'un élément de modélisation est unique dans l'espace de noms auquel il appartient:
$\forall x, element_1, element_2 /$
$(x, element_1, element_2) \in \text{NameSpace} \times (x.\text{allContents})^2,$
$element_1.\ name = element_2.\ name \Rightarrow element_1 = element_2$

Opérations additionnelles:

[2] L'opération allContents a pour résultat l'ensemble des éléments de modélisation contenus dans l'espace de noms:
$\text{allContents} = \{x / x \in self.ownedelement\}$

RootElement

[1] Le nom d'un élément de modélisation ne peut être vide:
$\forall x / x \in \text{RootElement}, x.\ name \neq$ ""

SimulationElement

[1] L'attribut *shared* peut avoir la valeur ***true*** seulement si *distributable* possède aussi la valeur ***true***:
$\forall x / x \in \text{SimulationElement}$,
$x.\ shared \Rightarrow x.\ distributable$

SimulationModel

[1] Un modèle de simulation ne peut contenir que des composants de modélisation:
$\forall x, y / (x, y) \in \text{SimulationModel} \times \text{RootElement}$,
$y \in x.\text{allContents} \Rightarrow y \in \text{SimulationElement}$

[2] L'attribut *shared* peut avoir la valeur ***true*** seulement si *distributable* possède aussi la valeur ***true***:
$\forall x / x \in \text{SimulationModel}$,
$x.\ shared \Rightarrow x.\ distributable$

A.2 Bibliothèque *PhysicalFlow*

La bibliothèque *PhysicalFlow* contient l'ensemble des structures nécessaires pour la construction d'un modèle représentant le sous-système physique d'un système de production. Nous y trouvons notamment les unités de productions, les transports, les ressources ou encore les files d'attente.

Nous définissons des concepts de base selon les briques disponibles dans la section 9 : entité (Entity), transition (Transition), ressource (Resource), modèle (PhysicalModel), sous-modèle (SubModelElement), queue (Queue), station (Station), unité de traitement (ProcessingUnit), ... Ces éléments de modélisations sont décrits dans la description Uml du paquetage *PhysicalFlow*.

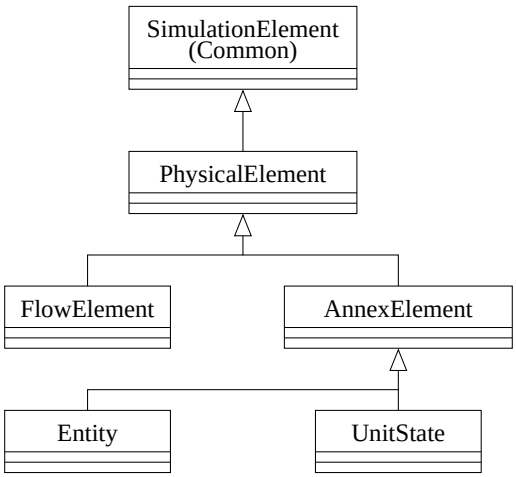


Fig. A.2 – *Simulation*::*PhysicalFlow* : éléments de modélisation

A.2.1 Syntaxe abstraite

Les figures A.2 à A.11 illustrent la syntaxe abstraite de ce paquetage.

AnnexElement

Cette classe d'objets est une abstraction de toutes les structures composant le modèle qui, à la différence de FlowElement, ne réalise aucun traitement sur les entités.

ArrivalElement

Pour que les entités puissent parcourir le modèle physique, il est nécessaire de les introduire dans ce dernier. La classe ArrivalElement correspond à un point d'entrée. Deux grandes méthodes sont utilisées pour faire apparaître des entités : la génération « spontanée » (*cf.* objet EntityGenerator) et l'arrivée depuis d'autres modèles (*cf.* objet IncomingFlow).

Attributs

#	::incomingcount	: Integer = 0	Une entité ne peut pas sortir du modèle par une porte d'entrée.

Associations

routingelement: Désigne un élément de modélisation des gammes associée à un élément de création d'entité (figure A.19 page 254).

Branch

Cette classe permet de modéliser les jonctions et les séparations de flux.

Attributs

+	branchcount	: Integer = $+\infty$	Nombre de branches gérées par cet objet.

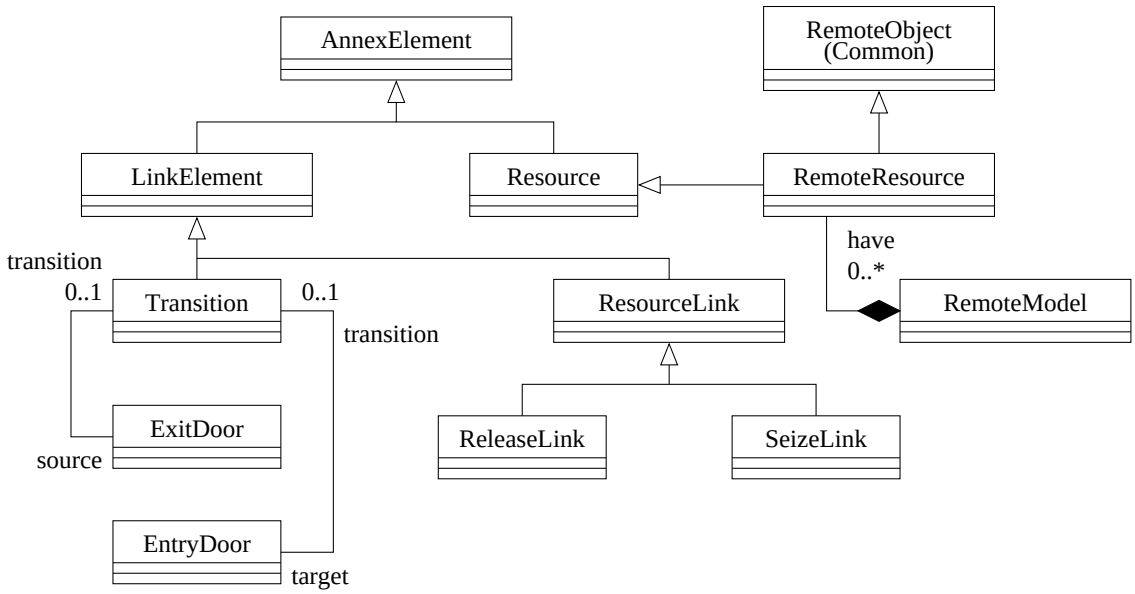


FIG. A.3 – *Simulation*::*PhysicalFlow* : liens et ressources

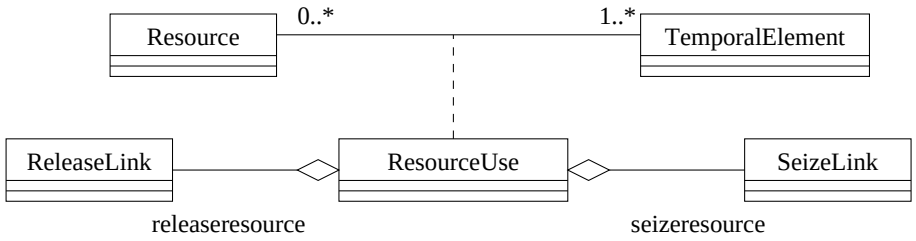


FIG. A.4 – *Simulation*::*PhysicalFlow* : Allocation et libération de ressources

Associations

branchlaw: Représente l'ensemble des lois utilisées par l'objet de branchement pour répartir (resp. recevoir) des entités dans (resp. depuis) des flux physiques (figure A.7 page 228).

BRANCHIN

La classe BRANCHIN permet de modéliser une structure de rassemblement de flux.

Attributs

+	::incomingcount : Integer = 2	Par défaut une jonction est réalisée à partir de deux flux. Cet attribut hérité change de portée en passant dans le domaine public.

BRANCHOUT

BRANCHOUT permet de modéliser une structure de dispersion des flux.

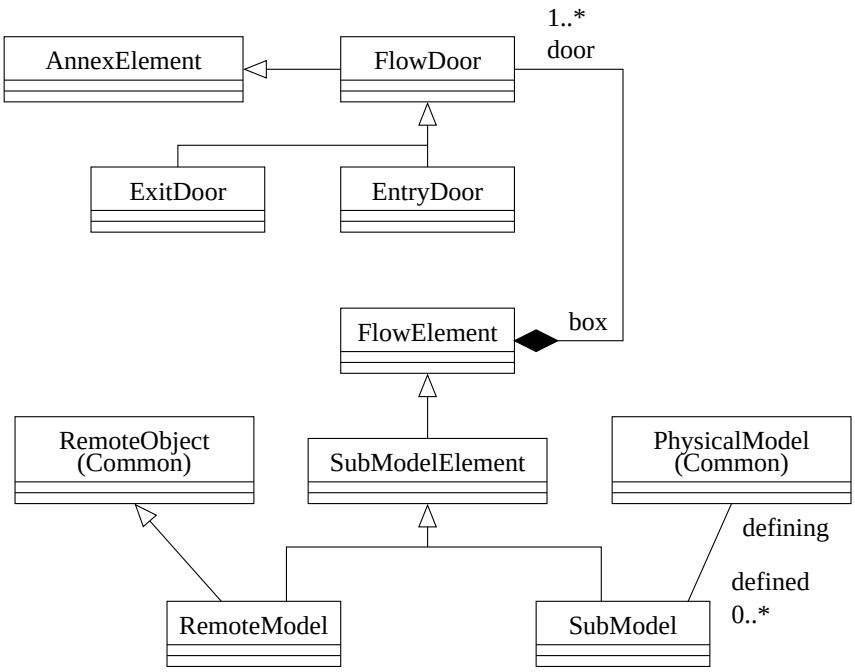


Fig. A.5 – *Simulation*::*PhysicalFlow* : Sous-modèles

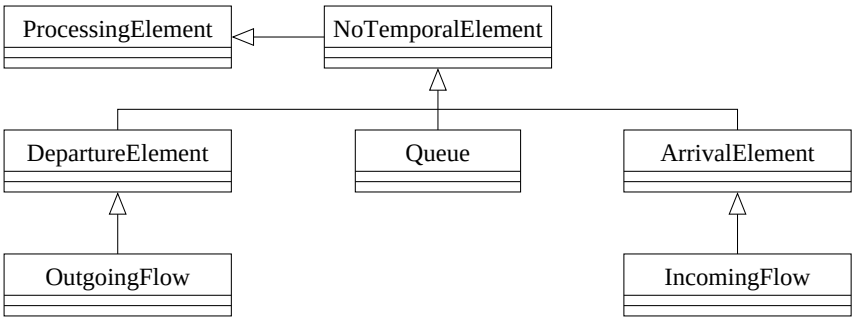


Fig. A.6 – *Simulation*::*PhysicalFlow* : Eléments avec traitement immédiat

Attributs

+	::outgoingcount : Integer = 2	Par défaut une disjonction est réalisée à partir de deux flux. Cet attribut hérité change de portée en passant dans le domaine public.

Convoyer

Un convoyeur est un dispositif de transport avec une capacité limité de chargement. Ainsi les entités ne pourront être transportées que si elles sont en nombre suffisant. La particularité d'un convoyeur est qu'il est toujours disponible *e.g.*, un escalator est un dispositif de transport qui ne peut prendre en charge que deux personnes à la fois (entités) mais que l'on peut utiliser sans délai d'attente.

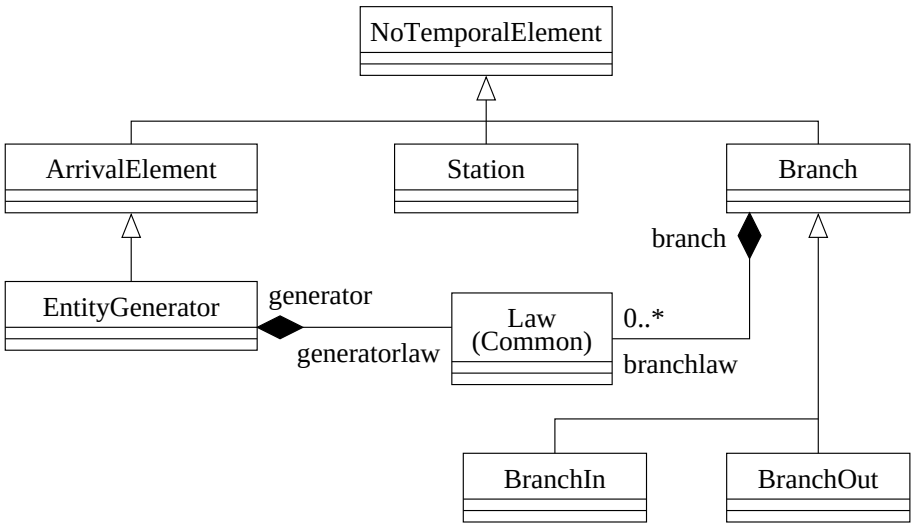


FIG. A.7 – *Simulation*::*PhysicalFlow* : Eléments avec traitement immédiat (2)

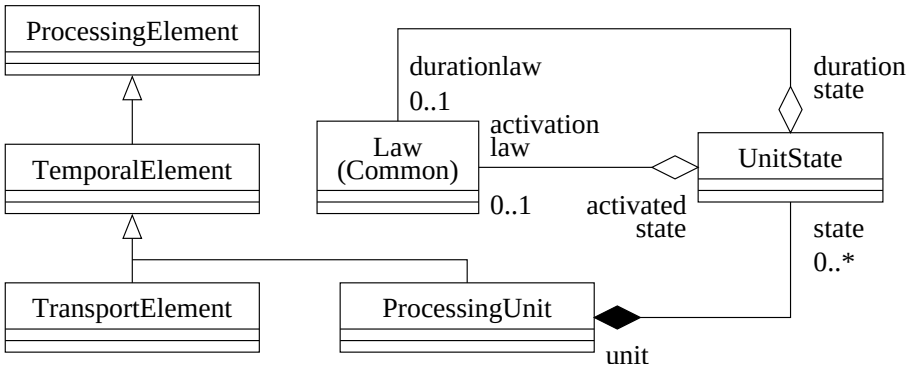


FIG. A.8 – *Simulation*::*PhysicalFlow* : Eléments avec traitement temporisé

Attributs

+	WaitForLimit : Boolean = true	Cet attribut représente la nécessité pour les entités d'être en nombre suffisant pour pouvoir utiliser le convoyeur. Si *WaitForLimit* est faux alors les entités utilisent le dispositif de transport immédiatement.

DEPARTUREELEMENT

Une fois le modèle parcouru, les entités doivent le quitter. La classe DEPARTUREELEMENT a le rôle de détruire les entités qui lui parviennent.

Attributs

#	::outgoingcount : Integer = 0	Aucune entité ne peut échapper à la destruction.

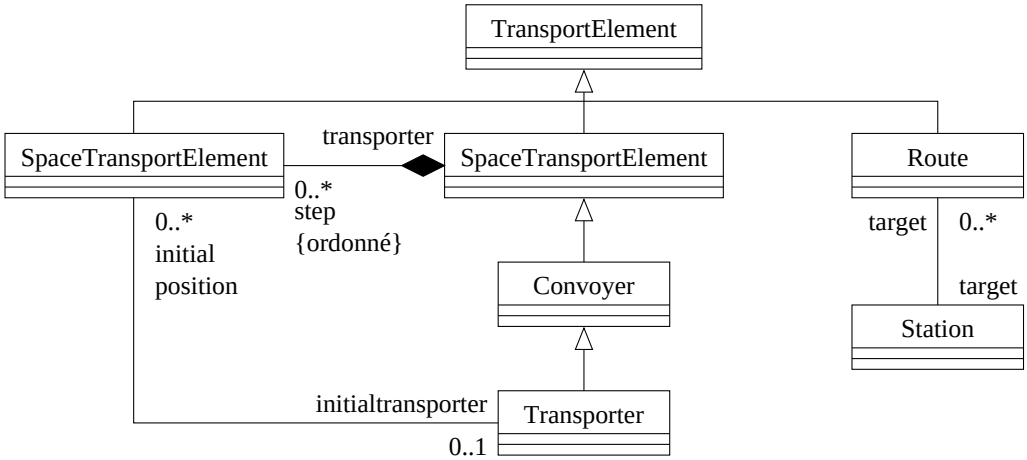


Fig. A.9 – *Simulation*::*PhysicalFlow* : Eléments de modélisation des transports

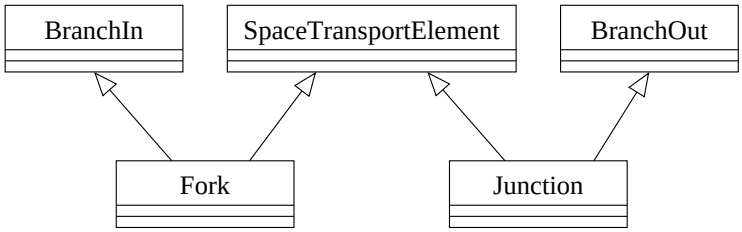


Fig. A.10 – *Simulation*::*PhysicalFlow* : Eléments de modélisation des transports (2)

Associations

routingelement: Désigne un élément de modélisation des gammes associée à un élément de destruction d'entité (figure A.19 page 254).

Entity

Une entité est une matière première, un produit semi-fini ou fini transitant par les différentes unités de production du sous-système physique. Lors du processus de simulation, les entités transites dans le modèle au travers des unités de traitements, des moyens de transports et des transitions. La classe Entity est incorporée au modèle conceptuel afin de permettre à l'analyste la description de certaines propriétés des entités.

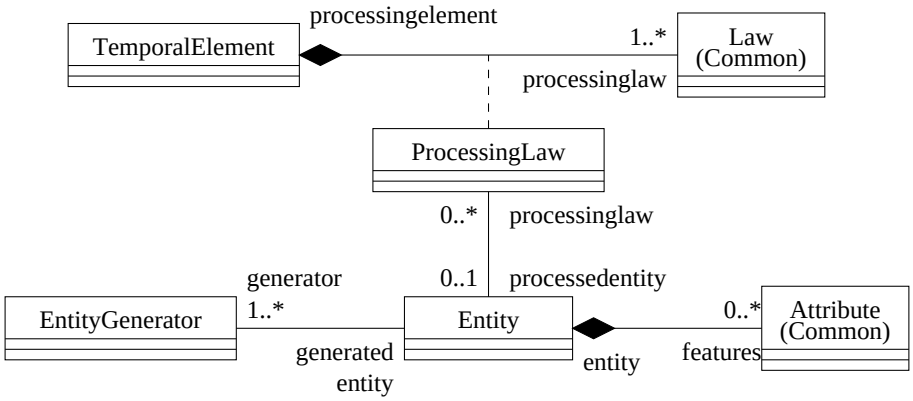


FIG. A.11 – *Simulation*::*PhysicalFlow* : Relations avec les entités

Associations

features:	Correspond à un ensemble de propriétés d'une entité définis par le modélisateur (figure A.11 page suivante).
generator:	Représente l'ensemble des générateurs capables de créer cette entité (figure A.11 page suivante).
informationalelement:	Une entité peut être associée à un produit appartenant à un modèle des nomenclatures (figure A.15 page 250).
processinglaw:	Désigne la loi utilisée par une unité de traitement pour cette entité (figure A.11 page suivante).

ENTITYGENERATOR

Cet objet est de la classe des créateurs d'entités (ARRIVALELEMENT). Il permet de générer des entités selon une loi statistique donnée par l'association *generatorlaw*.

Attributs

+	entitycount : Integer = 1	Propriété représentant le nombre d'entités générées en une seule fois *i.e.*, la quantité d'entité générée durant chaque phase de génération.

Associations

generatedentity:	Représente l'entité créée par le générateur (figure A.11).
generatorlaw:	Désigne la loi utilisée pour générer les entités (figure A.7 page 228).

ENTRYDOOR

Cette classe est une spécialisation d'une "porte" définie par FLOWDOOR. Elle représente les points d'entrée dans un élément de modélisation.

Associations

transition: Désigne la transition alimentant cette porte d'entrée (figure A.3 page 226).

ExitDoor

Cette classe est une spécialisation d'une "porte" définie par FlowDoor. Elle représente les points de sortie d'un élément de modélisation.

Associations

transition: Chaque porte de sortie peut être associée à une transition (figure A.3 page 226).

FlowDoor

Les composants représentant des sous-modèles ont besoin d'une représentation précise des flux entrants et sortants. La classe FlowDoor est un point de passage obligé d'un de ces flux. Ces "portes" composent les éléments de modélisation SubModelElement.

Associations

box: Désigne la nécessité d'appartenance d'une "porte" à une représentation d'un modèle (figure A.5 page 227).

FlowElement

La classe FlowElement permet d'avoir une abstraction pour toutes les structures réalisant des traitements sur les entités transitantes. L'ensemble des structures de ce type sont reliées par des transitions. Les entités passent d'une instance de FlowElement vers une autre grâce aux transitions. Le nombre de transitions entrantes et sortantes autorisées dépend du type réel de l'élément du flux.

Attributs

#	***constante*** incomingcount : Integer = 1	Le nombre de transitions que l'élément du flux peut avoir en entrée.
#	***constante*** outgoingcount : Integer = 1	Le nombre de transitions que l'élément peut avoir en sortie.

Associations

door: Représente l'ensemble des connecteurs d'entrée et de sortie attachés à un élément de modélisation (figure A.5 page 227).

FORK

Les dispositifs de transports peuvent parcourir des chemins comprenant des fouchètes *i.e.*, des séparations de voies. La modélisation de ce type d'artefact est réalisée par FORK. Cet classe à la particularité d'hériter son comportement à la fois de TRANSPORTCHECKPOINT mais aussi de BRANCHOUT. Ainsi, une fourchète est un point de passage pour les dispositifs de transports et une séparation d'un flux (ici un transport d'entités) en plusieurs parties.

INCOMINGFLOW

Les entités peuvent quitter dans un modèle englobant pour arriver dans un sous-modèle. La classe INCOMINGFLOW correspond à ce point d'entrée et génère une instance de l'entité arrivante dans le sous modèle.

JUNCTION

Les chemins parcourus par des dispositifs de transport peuvent faire l'objet d'un agglomérat qui correspond à la formation d'une voie unique à partir d'un nombre strictement positifs d'autres chemins de transports. La classe JUNCTION permet de modélisation cette jonction des flux de transport d'entités. Elle hérite son comportement à la fois de TRANSPORTCHECKPOINT et de BRANCHIN. Dans le premier cas, une jonction peut être considérée comme un point de passage pour un dispositif de transport. Dans le second cas, la jonction hérite le comportement d'une jonction de flux de BRANCHIN.

LINKELEMENT

LINKELEMENT est une classe abstraite commune à toutes les transitions composant le modèle de simulation du sous-système physique. Une transition est un lien entre deux FLOWELEMENT et ne sert qu'au transport instantané des entités.

NOTEMPORALELEMENT

Cette classe abstraite représente les éléments du modèle n'utilisant pas de loi statistique pour déterminer les temps de traitement. La sémantique de la loi est spécifique à chaque spécialisation de NOTEMPORALELEMENT.

OUTGOINGFLOW

Les entités peuvent quitter un modèle pour arriver dans un modèle englobant. La classe OUTGOINGFLOW détruit l'entité dans le modèle courant pour la recrée dans le modèle engobant.

PhysicalElement

Une élément de modélisation physique est la base de tous les éléments utilisés pour construire un modèle de simulation des flux physiques. La règle 1 page 241 permet de contraindre l'utilisation de cet classe d'objets.

ProcessingElement

Cette classe représente les éléments de modélisation réalisant des traitements sur les entités.

ProcessingLaw

Chaque élément de modélisation de type TemporalElement possède un ensemble de règles de traitement représentées par ProcessingLaw. Une loi de traitement peut être un couple ⟨entité, loi⟩ où la loi est utilisée pour calculer le temps de traitement de l'entité. Elle peut aussi être l'uplet ⟨loi⟩ qui correspond à la loi utilisée pour toutes les entités.

Associations

processedentity: Désigne l'entité attachée à une loi de traitement (figure A.11 page 230).

ProcessingUnit

ProcessingUnit représente les unités de production (machines, ...).

Associations

manager: Représente l'entité décisionnelle supervisant cette unité de production.

routingunit: Désigne un éventuel lien entre l'unité de production du flux physique et une unité de traitement des produits du modèle des gammes (figure A.17 page 252).

state: Une unité de production peut être dans différents états durant sa vie *e.g.*, panne, maintenance, travail, repos. Cette association permet de représenter l'ensemble des états possible d'une unité de production (figure A.8 page 228).

Queue

Les files d'attente sont des structures traditionnelles de la modélisation de systèmes de production. Le choix du type de gestion de la file (FIFO, LIFO, *etc.*) est réalisée grâce à une spécialisation de cette classe.

Attributs

+	maximalsize : Integer	Nombre d'entités pouvant être stockées dans la file d'attente

RELEASELINK

Cette spécialisation d'une liaison avec une ressource est spécifique à la libération d'une ressource. Elle est utilisée pour indiquer qu'une unité de production n'a plus besoin d'une ressource.

Associations

releaseressource:	Désigne la transition représentant la libération d'une ressource (figure A.4 page 226).

REMOTEMODEL

Cette structure permet d'inclure la représentation d'un modèle distant. Ainsi, lorsque les entités entreront dans une instance de REMOTEMODEL, elles seront automatiquement envoyée au modèle distant. Et toutes les entités sortantes peuvent être utilisées par le modèle local.

Associations

have:	Un modèle distant peut partager un ensemble de ressources lui appartenant (figure A.3 page 226).
routingmodel:	Désigne le lien pouvant exister entre un modèle distant des flux physiques et une station distante de traitement des produits d'un modèle des gammes (figure A.17 page 252).

REMOTERESOURCE

Il est parfois nécessaire à l'analyse d'utiliser une ressource commune à plusieurs sites. Pour cela, nous introduisons la notion de ressource distante. Il s'agit de l'utilisation d'une ressource distribuable appartenant à un autre modèle comme s'il s'agissait d'une ressource locale.

Attributs

#	::distributable : Boolean	Une ressource distante est par défaut non distribuable.

Associations

in:	Une ressource distante peut être déclarée comme appartenant à un modèle distant spécifique. Ainsi, lors de l'utilisation de ce dernier, les ressources déclarées pourront être utilisées localement (figure A.3 page 226).

Resource

Certaines ressources sont utilisées dans le modèle physique *e.g.*, les ouvriers et les outils. La classe Resource a été introduite afin de pouvoir représenter les besoins des unités de production.

Attributs

#	::distributable	: Boolean	Une ressource est par défaut distribuable.
+	initialquantity	: Integer	Nombre de ressources disponibles pour une instance de cet classe.

Associations

ResourceUse:	Désigne le lien existant entre une ressource et une unité de traitement (figure A.4 page 226). *cf.* objet ResourceUse.
routingressource:	Représente une ressource du modèle des gammes associée à une ressource du modèle physique (figure A.17 page 252).

ResourceLink

ResourceLink est la représentation du lien entre une unité de traitement et une ressource. Elle peut être soit une libération soit une réservation d'une ressource.

ResourceUse

Cette classe d'association permet de représenter l'utilisation d'une ressource par une unité de production. Elle comprend le nombre de ressources à utiliser ainsi que les liaisons d'allocation et de libération. Une utilisation de ressource est l'aggrégation de deux transitions spécifiques : l'allocation et la libération de la ressource par une unité de traitement.

Attributs

+	usingquantity	: Integer = 1	Nombre de ressources nécessaires pour l'allocation

Associations

releaseressource:	L'utilisation d'une ressource doit obligatoirement contenir par une liaison de libération entre la ressource et l'unité de production (figure A.4 page 226).
seizeressource:	idem pour l'allocation de la ressource (figure A.4 page 226).

Route

Une route est un moyen de transfert d'entités entre des éléments de modélisation. Elle autorise la prise en compte d'un délai de transfert. La loi héritée de TransportElement correspond à l'expression de ce délai.

Associations

target: Désigne la station où doit aboutir la route.

SEIZELINK

Cette liaison entre une unité de production et une ressource est spécifique à l'allocation de cette dernière par la première.

Associations

seizeressource: Désigne la transition représentant l'allocation d'une ressource (figure A.4 page 226).

SPACETRANSPORTELEMENT

Certains dispositifs de transport doivent prendre en compte des données spatiales. SPACETRANSPORTELEMENT représente ces éléments. Il prend en compte deux nouveaux concepts : la place disponible sur le dispositif de transport et le chemin de transport. Le premier est simplement une spécification du nombre d'entités pouvant être transportées selon leurs types. Le second concept inclu une donnée spatiale et une donnée temporelle : la longueur du chemin et la durée de transport. La loi héritée de TRANSPORTELEMENT possède une sémantique particulière en fonction du type de dispositif de transport "spatiale" utilisé.

Attributs

+ defaultsize : Integer = 1 — Désigne la quantité par défaut d'entités pouvant être transportées.

Associations

step: Désigne l'ensemble des étapes devant être utilisées pour construire le trajet de transport (figure A.9 page 229).

STATION

Une station est un objet par lequel les entités transites sans perdre de temps. Elle permet de nommer des points précis du flux.

Associations

road: Désigne les moyens de transport de type ROUTE aboutissant à cette station.

SUBMODEL

Pour facilité la modularité et le développement des modèles, nous introduisons la notion de sous-modèle. Celui-ci correspond à un modèle dont les flux d'entrée et de sortie correspondent à des flux entrant et sortant d'une boîte "noire" dans un modèle englobant.

Associations

defined: Désigne la définition d'un modèle physique utilisable comme un sous modèle (figure A.5 page 227).

SubModelElement

La classe SubModelElement est une abstraction représentant la notion de modularité des modèles. Chaque instance de cette classe possède un ensemble de flux entrant et sortants qui permettent à un modèle englobant de l'inclure.

Attributs

#	::incomingcount	: Integer = $+\infty$	Le nombre de points d'entrée dans un sous-modèle n'a pas de limite.
#	::outgoingcount	: Integer = $+\infty$	Le nombre de points de sortie d'un sous-modèle n'a pas de limite.

TemporalElement

TemporalElement est une abstraction d'une unité de traitement utilisant une loi statistique pour calculer le temps de traitement d'une entité.

Associations

ProcessingLaw*:* Représente les lois utilisées par le dispositif pour traiter les entités (figure A.11 page 230).

ResourceUse*:* Désigne l'utilisation d'une ressource par TemporalElement pour réaliser son travail (figure A.4 page 226). *cf.* objet ResourceUse.

Transition

Une transition est un lien entre deux FlowElement. Elle permet le déplacement des entités d'une unité de traitement vers une autre sans délai.

Associations

target: Représente la destination vers laquelle les entités passant par cette transition devront être envoyées (figure A.3 page 226).

source: Correspond à la source par laquelle les entités arrivent (figure A.3 page 226).

TRANSPORTCHECKPOINT

Un chemin de transport peut être composé d'un ou plusieurs sous-chemins. Ceux-ci sont séparés et liés par des points de passage (*checkpoint*). TRANSPORTCHECKPOINT est la représentation d'une de ces étapes. La loi statistique héritée de TRANSPORTELEMENT est utilisée pour calculer le temps de trajet jusqu'à cette étape.

Attributs

+	length : Integer = 1	Longueur spatiale pour parcourir le chemin jusqu'à cette étape.

Associations

initial-transporter:	Représente le lien entre un transporteur et sa position initiale dans le système (figure A.9 page 229).
transporter:	Désigne l'élément de transport contenant cette étape (figure A.9 page 229).

TRANSPORTELEMENT

Les entités ont parfois le besoin d'être transportées d'une partie du modèle vers un autre. Nous introduisons les éléments de transport qui permettent ce transfert.

TRANSPORTER

Un transporteur est un mécanisme permettant de transférer des entités d'un point vers un autre. Il correspond à une spécialisation d'un convoyeur (CONVOYER). En effet, un transporteur possède une position dans l'espace. Les entités doivent non seulement attendre que leur nombre soit suffisant mais doivent aussi attendre qu'un dispositif de transport soit disponible et présent géographiquement près d'elles.

Attributs

+	quantity : Integer = 1	Nombre de transporteurs disponibles pour ce même chemin.

Associations

initialposition:	Désigne la position initiale du transporteur le long de son chemin (figure A.9 page 229).

UnitState

Chaque unité de production peut posséder des états différents (panne, maintenance, repos, *etc.*). Cette classe d'objets permet de décrire chacun d'entre eux. Un état est caractérisé par une loi d'activation permettant de déterminer les instants où la machine doit changer d'état, ou par une loi statistique représentant la durée d'activation.

Associations

activationlaw: Représente la loi d'activation de l'état *i.e.*, la loi statistique déterminant la périodicité d'activation de l'état (figure A.8 page 228).

durationlaw: Désigne le lien existant entre un état et la définition d'une loi déterminant la durée d'activation d'un état *e.g.*, l'état de "panne" a une durée déterminée par une loi particulière (figure A.8 page 228).

unit: Désigne l'unité de production à laquelle l'état appartient (figure A.8 page 228).

Attributs

+ entityprocessing : Boolean = false — L'état d'une unité de traitement peut correspondre au traitement "normal" des entités (utilisation des lois de traitement héritées de TemporalElement). Par défaut, un état n'autorise pas la prise en compte des entités *e.g.*, l'unité de traitement est en état de panne.

A.2.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

BRANCHIN

[1] Un élément de modélisation pour la jonction de flux n'a de sens que s'il possède au moins deux flux entrants:
$\forall x / x \in$ BRANCHIN,
$x.\ incomingcount \geq 2$

[2] Le nombre de lois composant un élément de branchement doit être égal au nombre de flux entrants:
$\forall x / x \in$ BRANCHIN,
$\text{card}(x.branchlaw) = x.\ incomingcount$

[3] Le nombre de branches doit être égal à l'attribut hérité:
$\forall x / x \in$ BRANCHIN,
$x.\ branchcount = x.\ incomingcount$

BRANCHOUT

[1] Un élément de modélisation pour la séparation de flux n'a de sens que s'il possède au moins deux flux sortants:
$\forall x / x \in$ BRANCHIN,
$x.\ outgoingcount \geq 2$

[2] Le nombre de lois composant un élément de branchement doit être égal au nombre de flux sortants:
$\forall x / x \in$ BRANCHIN,
$\text{card}(x.branchlaw) = x.\ outgoingcount$

[3] Le nombre de branches doit être égal à l'attribut hérité:
$\forall x / x \in$ BRANCHIN,
$x.\ branchcount = x.\ outgoingcount$

ENTITY

Opérations additionnelles:

[1] L'opération attributNames permet de construire l'ensemble des noms des attributs associés au produit:
attributNames $= \{n / \forall x \exists n, x \in self.features \wedge n = x.\ name\}$

EntityGenerator

[1] Un générateur d'entité doit produire des paquets non vides d'entités:
$\forall x / x \in \text{EntityGenerator}$,
$x.\ entitycount \geq 1$

FlowElement

[1] Les nombres de transitions pouvant arriver ou partir d'un élément de modélisation du flux physique sont positifs ou nuls:
$\forall x / x \in \text{FlowElement}$,
$x.\ incomingcount \geq 0 \wedge x.\ outgoing \geq 0$

[2] Le nombre de transitions arrivant à un élément de modélisation du flux physique ne doit pas dépasser le nombre maximale d'entrées possibles:
$\forall x / x \in \text{FlowElement}$,
$\text{card}(\text{EntryDoor} \cap x.door) \leq x.\ incomingcount$

[3] Le nombre de transitions partant d'un élément de modélisation du flux physique ne doit pas dépasser le nombre maximale de sorties possibles:
$\forall x / x \in \text{FlowElement}$,
$\text{card}(\text{ExitDoor} \cap x.door) \leq x.\ outgoingcount$

[4] Les portes appartiennent à l'espace de nom défini par la classe FlowElement:
$\forall x, y / (x, y) \in \text{FlowElement} \times \text{FlowDoor}$,
$y \in x.door \Rightarrow y \in x.\text{allContents}$

PhysicalElement

[1] Un élément de modélisation de type PhysicalElement ne peut être présent qu'à l'intérieur d'un modèle des flux physiques:
$\forall x, y / (x, y) \in \text{PhysicalElement} \times \text{NameSpace}$,
$y \in x.namespace \Rightarrow y \in \text{PhysicalModel}$

ProcessingLaw

Opérations additionnelles:

[1] L'opération isDefaultLaw permet de déterminer si une loi de traitement est une loi par défaut ou non:
isDefaultLaw = (card(*self.processedentity*) = 0))

Queue

[1] Le nombre d'entités pouvant être stockées dans la file d'attente doit être strictement positive:
$\forall x / x \in$ Queue,
$x.\ maximalsize \geq 1$

Resource

[1] Une instance de Resource doit avoir initialement au moins une ressource disponible:
$\forall x / x \in$ Resource,
$x.initialquantity \geq 1$

SpaceTransportElement

[1] La quantité par défaut doit être strictement positif:
$\forall x / x \in$ SpaceTransportElement,
$x.\ defaultsize \geq 1$

[2] Les points de passage d'un élément de transport sont compris dans l'espace de noms de ce dernier:
$\forall x, y / (x, y) \in$ SpaceTransportElement $\times$ TransportCheckPoint,
$y \in x.step \Rightarrow y \in x$.allContents

SubModel

[1] Chaque porte d'entrée de l'élément de modélisation correspond à une instance d'IncomingFlow dans le sous-modèle associé:
$\forall w, x, y \exists z /$
$(w, x, y, z) \in \text{SubModel} \times \text{PhysicalModel} \times \text{IncomingFlow} \times \text{EntryDoor}$,
$$\begin{pmatrix} y \in x.\textit{defining} \wedge \\ z \in y.\text{allContents} \wedge \\ w \in x.\textit{door} \end{pmatrix} \Rightarrow w.\ \textit{name} = z.\ \textit{name}$$

[2] Chaque porte de sortie de l'élément de modélisation correspond à une instance d'OutgoingFlow dans le sous-modèle associé:
$\forall w, x, y \exists z /$
$(w, x, y, z) \in \text{SubModel} \times \text{PhysicalModel} \times \text{OutgoingFlow} \times \text{ExitDoor}$,
$$\begin{pmatrix} y \in x.\textit{defining} \wedge \\ z \in y.\text{allContents} \wedge \\ w \in x.\textit{door} \end{pmatrix} \Rightarrow w.\ \textit{name} = z.\ \textit{name}$$

TemporalElement

[1] Une seule loi par défaut est autorisée:
$\forall x, y, z / (x, y, z) \in \text{TemporalElement} \times \text{ProcessingLaw}^2$,
$$\begin{pmatrix} y \in x.\textit{processinglaw} \wedge z \in x.\textit{processinglaw} \wedge \\ y.\text{isDefaultLaw} \wedge y \neq z \end{pmatrix} \Rightarrow \neg z.\text{isDefaultLaw}$$

Opérations additionnelles:

[2] L'opération allLaws retourne l'ensemble des fonctions $\langle \text{entité} \rangle \rightarrow \langle \text{loi} \rangle$ associées à l'unité de traitement temporisé:
$$\text{allLaws} = \left\{ x \rightarrow y \;\middle|\; \begin{array}{c} x \in \text{Entity} \cup \{\text{nil}\} \wedge z \in \textit{self}.\text{ProcessingLaw} \wedge \\ \left(\begin{array}{c} (z.\text{isDefaultLaw} \wedge x = \text{nil}) \vee \\ (\neg z.\text{isDefaultLaw} \wedge x \in z.\textit{processedentity}) \end{array} \right) \wedge \\ y = z.\textit{processinglaw} \end{array} \right\}$$

[3] L'opération getResources construit l'ensemble des ressources utilisée par une machine:
$\text{getResources} = \{x / \forall y \in \textit{self}.\text{ResourceUse} \wedge x = y.\text{Resource}\}$

TRANSPORTCHECKPOINT

[1] Si un point de passage est aussi un point de départ d'un dispositif de transport, alors il fait partie uniquement du chemin de celui-ci.:
$\forall x, y/(x, y) \in$ TRANSPORTCHECKPOINT $\times$ TRANSPORTER,
$y \in x.$*initial-transporter* $\Rightarrow y \in x.$*transporter*

[2] La distance pour atteindre un point de passage n'a de sens que si elle est positive:
$\forall x/x \in$ TRANSPORTCHECKPOINT,
$x.$ *length* ≥ 1

TRANSPORTER

[1] Le nombre de ressources d'un transporteur doit être strictement positif:
$\forall x/x \in$ TRANSPORTER,
$x.$ *quantity* ≥ 1

[2] Le nombre de points de départs particuliers *i.e.*, différents du début du chemin de transport, doit être inférieur ou égale au nombre de ressources:
$\forall x/x \in$ TRANSPORTER,
$\text{card}(x.$*initialposition*$) \leq x.$ *quantity*

[3] Chaque point de départ d'une ressource doit faire partie du chemin de transport:
$\forall x, y/(x, y) \in$ TRANSPORTER $\times$ TRANSPORTCHECKPOINT,
$y \in x.$*initialposition* $\Rightarrow y \in x.$*step*

UNITSTATE

[1] Si une loi de durée est associée à un état, il y a toujours une loi d'activation associée à ce même état:
$\forall x/x \in$ UNITSTATE,
$\text{card}(x.$*durationlaw*$) > 0 \Rightarrow \text{card}(x.$*activationlaw*$) > 0$

—Hht]

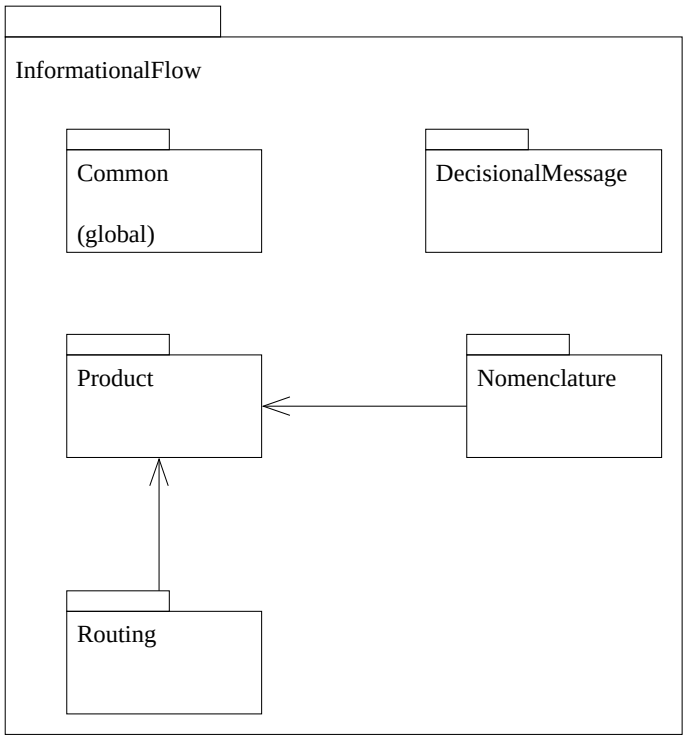


Fig. A.12 – Paquetage *InformationalFlow*

A.3 Bibliothèque *InformationalFlow*

La bibliothèque *InformationalFlow* contient toutes les structures nécessaire à la construction de modèles du sous-système informationnel. Nous y trouvons les produits, les gammes, les nomenclatures ainsi que la définition des messages décisionnels. Les paquetages *Common*, *DecisionalMessage*, *Nomenclature*, *Product* et *Routing* sont présentés dans les sections suivantes (*cf.* figure A.12).

A.3.1 Paquetage *InformationalFlow*::*Common*

Cette bibliothèque contient l'ensemble des structures présente dans l'ensemble du modèle de simulation. Nous y définissons les éléments basiques pour la modélisation du sous-système décisionnel ainsi que les sous-modèles composant celui-ci.

A.3.1.1 Syntaxe abstraite

La figure A.13 illustre la syntaxe abstraite de cette bibliothèque.

InformationalElement

Cette classe représente les éléments de modélisation composant le modèle informationnel.

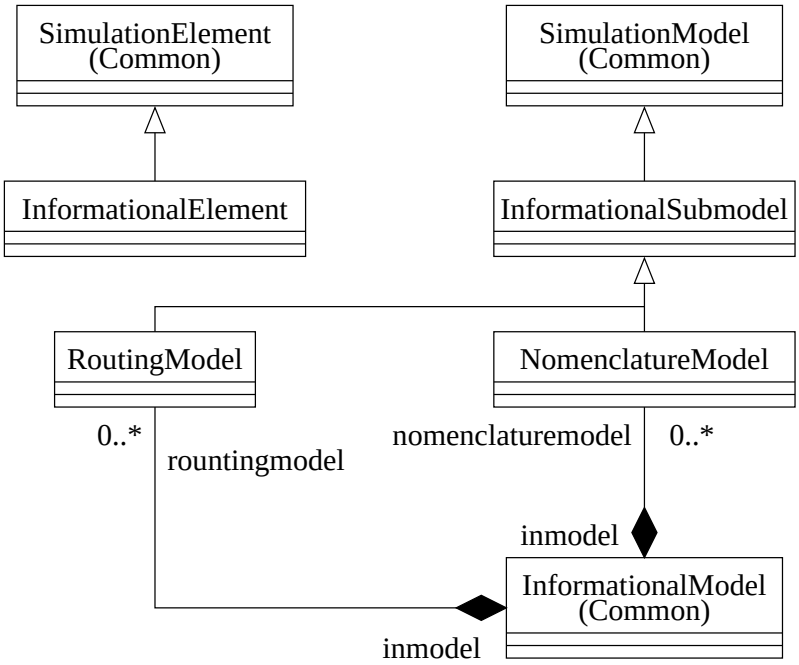


FIG. A.13 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*Common*

INFORMATIONALSUBMODEL

Cette classe est l'abstraction correspondant aux modèles composant un modèle informationnel.

NOMENCLATUREMODEL

Le modèle des nomenclatures contient l'ensemble des possibilités de composition des produits. Chaque instance de NOMENCLATUREMODEL est contenu dans un seul modèle informationnel global.

Associations

inmodel: Chaque modèle des nomenclatures est compris dans un modèle informationnel global (figure A.13).

ROUTINGMODEL

Le modèle des gammes contient l'ensemble des possibilités de fabrication des produits. Chaque instance de ROUTINGMODEL est contenu dans un seul modèle informationnel global.

Associations

inmodel: Chaque modèle des gammes est compris dans un modèle informationnel global (figure A.13).

A.3.1.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

InformationalSubmodel

[1] Les éléments appartenant à un sous-modèle informationnel ne peuvent être que des éléments de modélisation informationnels:
$\forall x, y/(x, y) \in$ InformationalSubmodel $\times$ SimulationElement,
$y \in x$.allContents $\Rightarrow y \in$ InformationalElement

NomenclatureModel

[1] Un modèle de simulation ne peut contenir que des éléments de modélisation de nomenclatures ou des produits.:
$\forall x, y/(x, y) \in$ NomenclatureModel $\times$ SimulationElement,
$y \in x$.allContents $\Rightarrow y \in$ NomenclatureElement $\vee\, y \in$ Product

RoutingModel

[1] Un modèle de simulation ne peut contenir que des éléments de modélisation des gammes.:
$\forall x, y/(x, y) \in$ RoutingModel $\times$ SimulationElement,
$y \in x$.allContents $\Rightarrow y \in$ RoutingElement

A.3.2 Paquetage *InformationalFlow*::*Nomenclature*

Le paquetage *Nomenclature* contient l'ensemble des définitions des structures utilisées pour modéliser les nomenclatures.

A.3.2.1 Syntaxe abstraite

La figure A.14 illustre la syntaxe abstraite de cette bibliothèque.

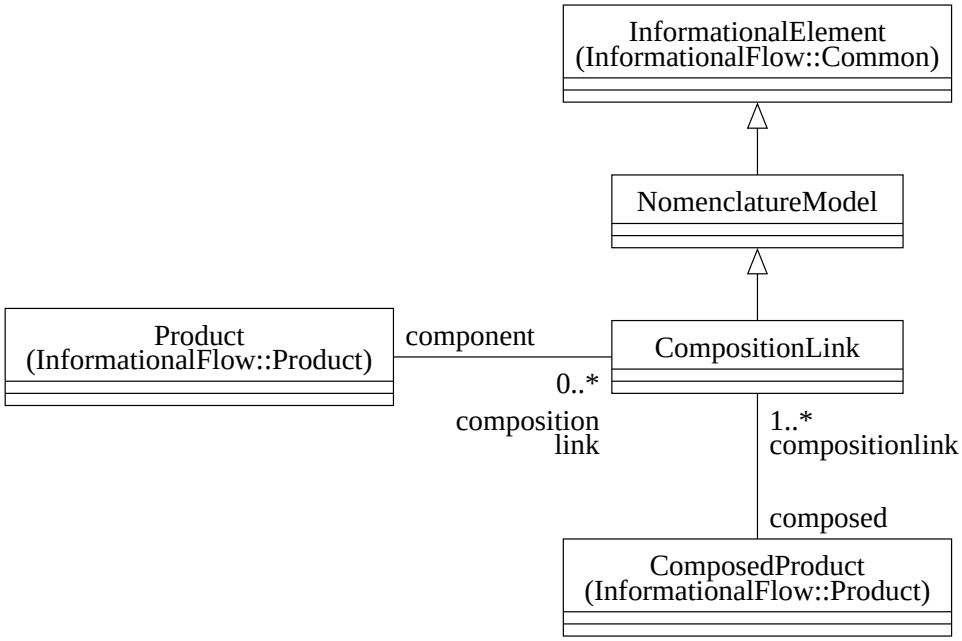


FIG. A.14 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*Nomenclature*

COMPOSITIONLINK

Les produits et leurs composants sont reliés par des transitions afin de schématiser les différentes constructions de produits possibles. Une transition part d'un produit décomposable pour aboutir à un produit quelconque.

Associations

component: Une transition abouti à un produit quelconque composant le produit de départ (figure A.14 page suivante).

composed: Une transition part d'un produit décomposable (figure A.14 page suivante).

NOMENCLATUREELEMENT

Cette classe défini l'élément de base du modèle des nomenclatures. Chaque structure composant ce dernier doit obligatoirement être une spécialisation de NOMENCLATUREELEMENT (sauf pour les produits).

A.3.2.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

CompositionLink

[1] Les deux produits d'un lien de composition doivent appartenir au même modèle des nomenclatures que l'instance de CompositionLink:
$\forall w, x, y, z /$
$(w, x, y, z) \in$ NomenclatureModel $\times$ CompositionLink $\times$ Product^2,
$(x \in w.\text{allContents} \wedge y \in x.component \wedge z \in x.composed)$
$\Rightarrow (y \in w.\text{allContents} \wedge z \in w.\text{allContents})$

[2] Les deux produits d'un lien de composition doivent être différents:
$\forall w, x, y, z /$
$(w, x, y, z) \in$ NomenclatureModel $\times$ CompositionLink $\times$ Product^2,
$(x \in w.\text{allContents} \wedge y \in x.component \wedge z \in x.composed)$
$\Rightarrow y \neq z$

NomenclatureElement

[1] Un élément de modélisation de type NomenclatureElement appartient uniquement à un modèle des nomenclatures:
$\forall x, y / (x, y) \in$ NomenclatureElement $\times$ SimulationModel,
$x \in y.\text{allContents} \Rightarrow y \in$ NomenclatureModel

A.3.3 Paquetage *InformationalFlow*::*Product*

La bibliothèque *Product* contient l'ensemble des structures définissant les produits.

A.3.3.1 Syntaxe abstraite

La figure A.15 illustre la syntaxe abstraite de cette bibliothèque.

ComposedProduct

Cette classe d'objets représente les produits composés à partir d'autres produits. Il s'agit de la seule définition d'un produit pouvant être partagée avec d'autres modèles informationnels.

Attributs

+	::distributable : Integer = true		Un produit composé peut être partagé avec d'autres modèles d'informations.

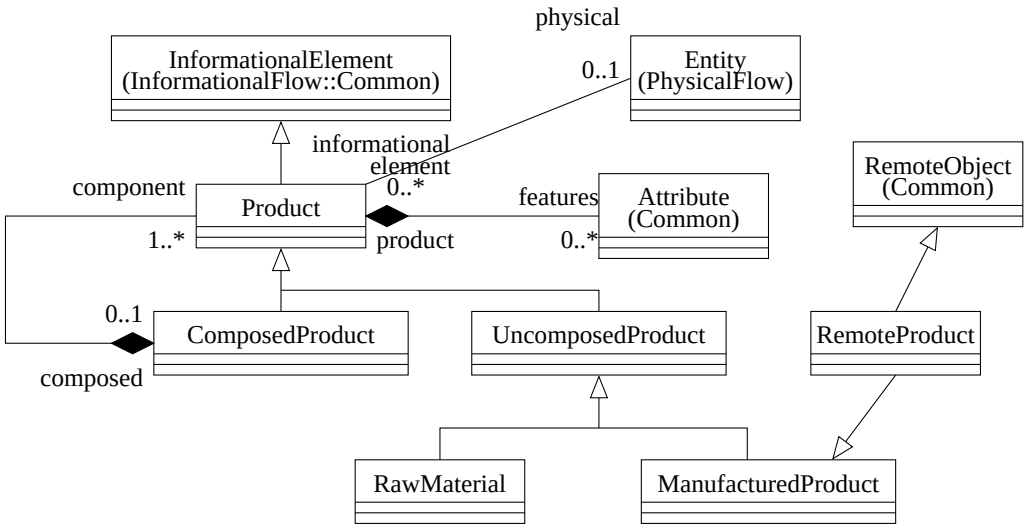


FIG. A.15 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*Product*

Associations

component: Un produit composé est construit à partir d'autres produits (figure A.15 page suivante).

compositionlink: Un produit composé fait l'objet d'une définition de composition à travers un COMPOSITIONLINK (figure A.15 page suivante).

MANUFACTUREDPRODUCT

MANUFACTUREDPRODUCT est une classe permettant de modéliser les produits manufacturés *i.e.*, les produits dont la décomposition n'est pas connue ou non intéressante.

PRODUCT

Un produit est la source et le résultat d'un processus de production. Il peut être de la matière première, des produits semi-finis ou finis. La classe PRODUCT est une abstraction qui est spécialisée selon le type du produit.

Associations

composed: Un produit peut composer un autre produit (figure A.15).

features: Correspond à l'ensemble des attributs attachés à un produit (figure A.15).

physicalelement: Désigne une entité représentant le produit dans le modèle physique (figure A.15).

usingtransition: Un produit peut être associé à une transition afin d'illustrer les mouvements de produits au sein d'une gamme (figure A.19 page 254).

RawMaterial

Cette classe permet de modéliser les matières premières.

RemoteProduct

RemoteProduct est une classe permettant la modélisation et l'utilisation d'une définition distante d'un produit.

UncomposedProduct

La classe d'objets UncomposedProduct représente les produits qui ne sont pas décomposables ou dont la décomposition n'est pas intéressante.

A.3.3.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

ComposedProduct

[1] Les produits composants doivent appartenir au même modèle des nomenclatures:
$\forall x, y, z /$
$(x, y, z) \in$ ComposedProduct $\times$ Product $\times$ NomenclatureModel,
$(y \in x.component \wedge z \in x.inmodel) \Rightarrow z \in y.inmodel$

Product

[1] L'ensemble des attributs d'un produit doit existé dans ceux de l'entité physique associée:
$\forall x, y / (x, y) \in$ Product $\times$ Entity,
$y \in x.physicalelement \Rightarrow x.$attributNames $\subseteq y.$attributNames

Opérations additionnelles:

[2] L'opération attributNames permet de construire l'ensemble des noms des attributs associés au produit:
attributNames $= \{n / \forall x \exists n, x \in self.features \wedge n = x.\ name\}$

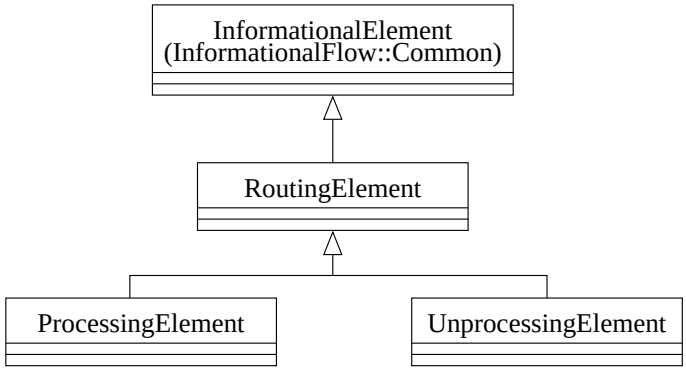


FIG. A.16 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*Routing*

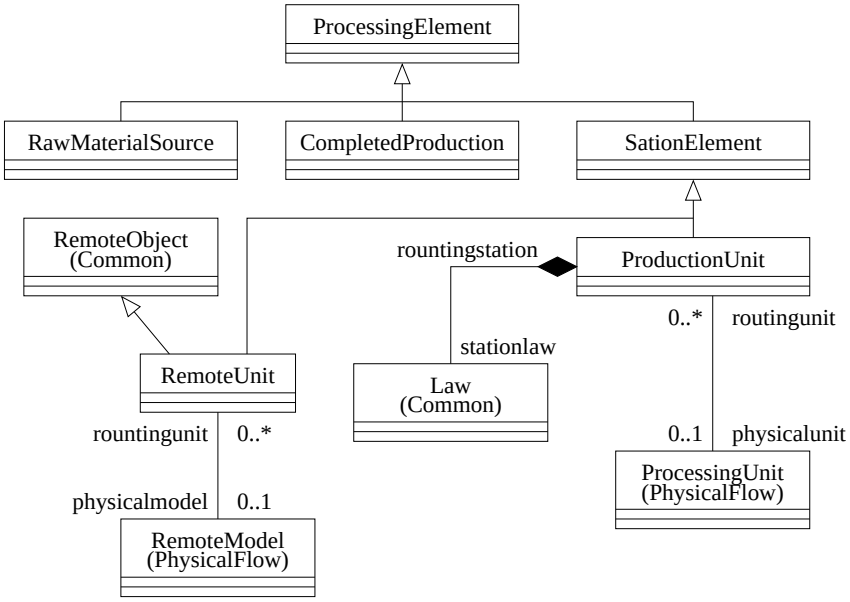


FIG. A.17 – *Simulation*::*InformationalFlow*::*Routing* : éléments de traitement

A.3.4 Paquetage *InformationalFlow*::*Routing*

La bibliothèque *Routing* regroupe les structures permettant la modélisation de gammes.

A.3.4.1 Syntaxe abstraite

Les figures A.16 à A.20 illustrent la syntaxe abstraite de cette bibliothèque.

CompletedProduction
La classe CompletedProduction permet de placer un élément modélisant la fin de la gamme *i.e.*, le moment où le produit fini est terminé.

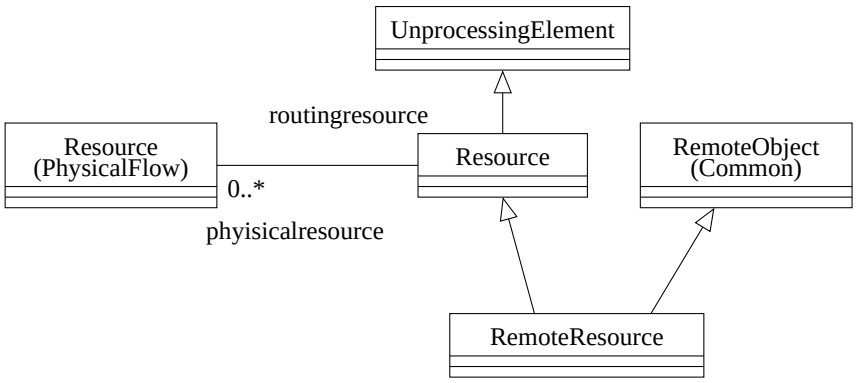


Fig. A.18 – *Simulation*::*InformationalFlow*::*Routing* : gestion des ressources

Associations

sender: L'élément de modélisation de la fin d'une gamme est reliée à une station par une transition de type LastTransition.

FirstTransition

La classe FirstTransition est le première transition dans la gamme. Elle permet de reliée les sources de matières premières avec les unités de traitement.

Associations

physicalelement: Désigne l'élément de modélisation du flux physique correspondant à un élément du modèle des gammes.

InterTransition

Cette classe permet de modéliser les transitions entre deux unités de traitement.

LastTransition

La classe LastTransition permet la modélisation de la transition entre la dernière unité de traitement de la gamme et une instance de CompletedProduction.

Associations

physicalelement: Désigne l'élément de modélisation du flux physique correspondant à un élément du modèle des gammes.

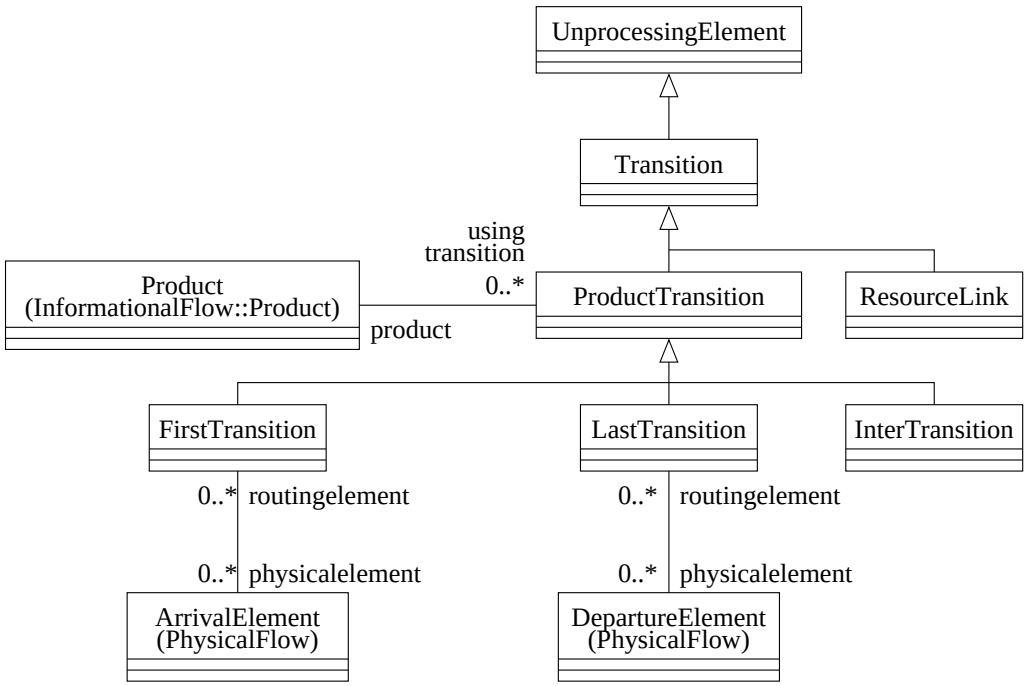


FIG. A.19 – *Simulation*::*InformationalFlow*::*Routing* : transitions

PROCESSINGELEMENT

Cette classe représente tous les éléments du modèle réalisant un traitement sur les produits entrant. Chaque PROCESSINGELEMENT est relié par un ensemble de transitions représentants les déplacements de produits.

PRODUCTIONUNIT

PRODUCTIONUNIT est une classe représentant les unités de transformation des produits. Une loi est utilisée pour le calcul de la durée de traitement d'un produit.

Attributs

+	::distributable : Boolean = true	Une unité de traitement des produit peut être partagée avec des modèles distants de gammes.

Associations

physicalunit:	Désigne une unité de traitement du flux physique correspondant à l'unité dans le modèle des gammes (figure A.16 page 252).
ressource:	une station a la possiblité d'utiliser une ressource pour réaliser sa tâche.
stationlaw:	Représente une loi statistique utilisée par la station de traitement (figure A.17 page 252).

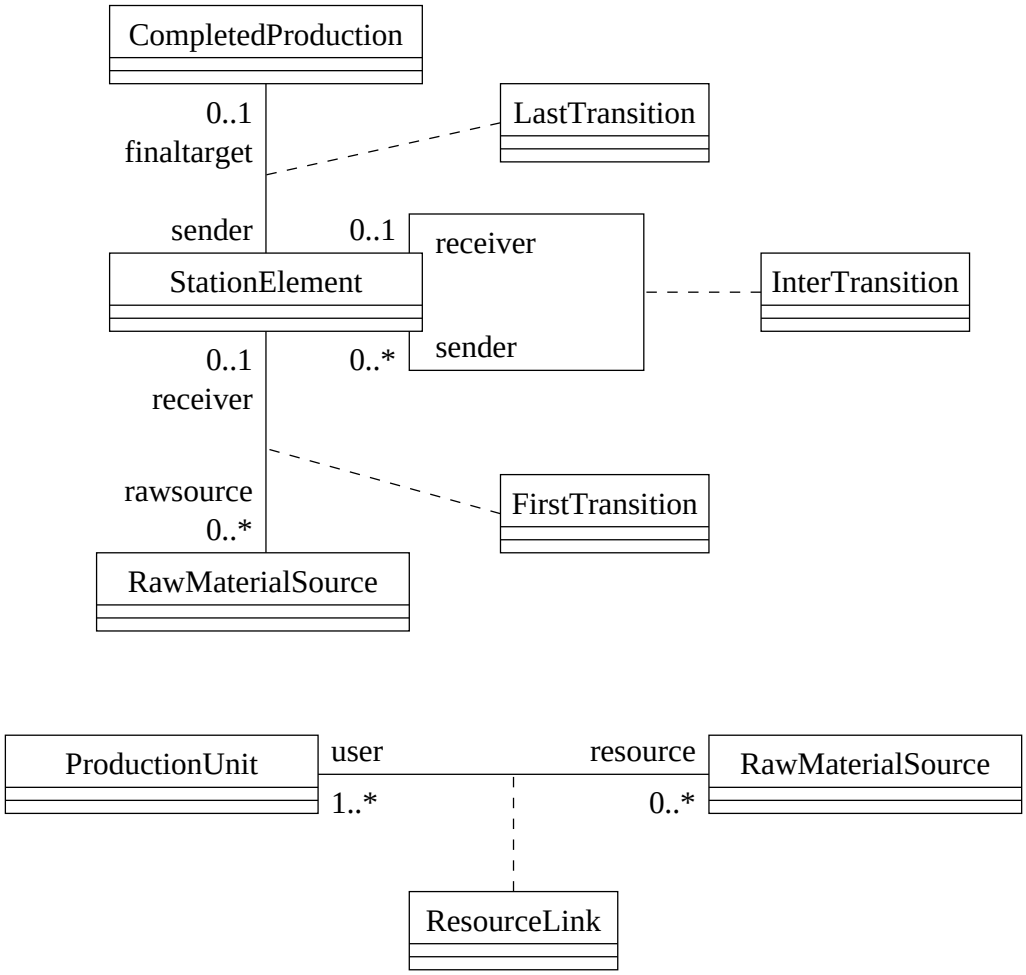


Fig. A.20 – *Simulation*::*InformationalFlow*::*Routing* : Liaisons

ProductTransition

Une catégorie de transitions existe entre les unités de traitement du modèle des gammes. Elle permettent de représenter les mouvements des produits. La classe ProductTransition correspond à la classe abstraite des transitions transportant un produit.

Attributs

+ batchsize : Integer = 1 — Taille du lot de produits consommés par l'entité où arrive la transition.

Associations

product: Désigne le produit attaché à la transition *i.e.*, le produit transitant par cette élément de modélisation.

RawMaterialSource

Cette classe permet de modéliser les sources de matières premières. Toute gamme doit obligatoirement débuter par une instance de RawMaterialSource.

Associations

receiver: une source de matière première est lié à une station de traitement où elle envoi ces matières.

RemoteUnit

Une station de traitement peut être distante. La classe RemoteUnit permet d'utiliser cette catégorie de station dans un modèle des gammes.

Associations

physicalmodel: Une station distante de traitement de produits peut être associée à un modèle distant du modèle des flux physiques (figure A.17 page 252).

RemoteResource

La classe RemoteResource permet la modélisation de ressources se trouvant dans des modèles de gammes distants.

Resource

Resource représente le type d'une ressource nécessaire pour la réalisation d'une étape de la gamme.

Associations

physicalressource: une ressource peut être associée à une ou plusieurs ressources appartenant au modèle des flux physiques.

user: une ressource doit être utilisée par au moins une station de traitement.

ResourceLink

Une transition de ressource est un lien entre une ressource et une unité de traitement. Elle permet de modéliser la nécessité d'utilisation d'un moyen particulier.

RoutingElement

Cette classe représente l'abstraction commune à toutes les éléments du modèle des gammes.

StationElement

Cette classe est une généralisation des unités de traitements des produits.

Associations

finaltarget: Une station peut être réliée à une élément de modélisation de la fin de la gamme.

rawsource: Désigne l'un des type de sources de produit pour une station : les matières premières.

receiver: une station peut émettre un produit vers une autre station.

sender: une station peut recevoir un produit d'une autre station.

Transition

Une transition est un lien entre des éléments particuliers du modèle : unités de traitement, source de matière première et indication de la fin de la gamme.

UnprocessingElement

Cette classe abstraite représente les éléments du modèle ne réalisant pas traitement sur les produits.

A.3.4.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

FirstTransition

[1] La taille du lot d'entités est strictement positive:
$\forall x / x \in \text{FirstTransition}$,
$x.\ batchsize \geq 1$

[2] La première transition d'un modèle de gammes et son créateur d'entité doivent être respectivement associés à un produit et une entité équivalents:
$\forall w, x, y, z /$
$(w, x, y, z) \in \text{FirstTransition} \times \text{ArrivalElement} \times \text{Entity} \times \text{Product}$,
$$\begin{pmatrix} x \in w.physicalelement \wedge \\ y \in x.generatedentity \wedge \\ z \in w.product \end{pmatrix} \Rightarrow y \in z.physicalelement$$

INTERTRANSITION

[1] La taille du lot d'entités est strictement positive:
$\forall x / x \in$ FIRSTTRANSITION,
$x.\ batchsize \geq 1$

LASTTRANSITION

[1] La taille du lot d'entités est égale à un car il n'y a pas de consommateur d'entités au point d'arriver de cette transition.:
$\forall x / x \in$ LASTTRANSITION,
$x.\ batchsize = 1$

PRODUCTIONUNIT

[1] La loi utilisée dans le modèle des gammes doit être similaire à celle proposée dans le modèle physique:
$\forall v, w, x, y, \exists z /$
$(v, w, x, y, z) \in$ PRODUCTIONUNIT $\times$ PROCESSINGUNIT$\times$
ENTITY $\times$ LAW^2,

$$\begin{pmatrix} v.physicalelement = \{w\} \wedge \\ y \in v.stationlaw \wedge \\ x \in v.\text{translateProducts} \wedge \\ (x \mapsto z) \in w.\text{allLaws} \end{pmatrix} \Rightarrow (y.\text{sameAs}(z) \wedge z.\text{sameAs}(y))$$

[2] Les ressources utilisées par une unité de production sont similaires à celles utilisées par l'unité physique:
$\forall w, x, y, z /$
$(w, x, y, z) \in$ PRODUCTIONUNIT $\times$ PROCESSINGUNIT$\times$
RESOURCE $\times$ *PhysicalFlow* :: RESOURCE,

$$\begin{pmatrix} w.physicalunit = \{x\} \wedge \\ y \in w.ressource \wedge \\ z \in x.\text{getResources} \end{pmatrix} \Rightarrow z \in y.physicalressource$$

RoutingElement

[1] Un élément de modélisation de type RoutingElement ne peut appartenir qu'a un modèle des gammes:
$\forall x, y/(x, y) \in \text{RoutingElement} \times \text{SimulationModel}$,
$x \in y.\text{allContents} \Rightarrow y \in \text{RoutingModel}$

StationElement

[1] Une station de traitement ne peut être la source que d'une seule transition:
$\forall x/x \in \text{StationElement}$,
$\text{card}(x.\textit{finaltarget}) > \text{card}(x.\textit{receiver}) \vee \text{card}(x.\textit{finaltarget}) < \text{card}(x.\textit{receiver})$

[2] Au moins une transition arrive à une station de traitement:
$\forall x/x \in \text{StationElement}$,
$\text{card}(x.\textit{rawsource}) + \text{card}(x.\textit{sender}) \geq 1$

Opérations additionnelles:

[3] L'opération getArrivals retourne l'ensemble des transitions arrivant à une station de traitement:

$$\text{getArrivals} = \left\{ x \;\middle|\; \begin{array}{c} x \in \text{ProductTransition} \wedge \\ \left(\begin{array}{c} x \in \textit{self.rawsource} \vee \\ x \in \textit{self.sender} \end{array} \right) \end{array} \right\}$$

[4] L'opération getProducts retourne le produit traité par l'unité de traitement.:
$\text{getProducts} = \{x/x \in \text{Product} \wedge \forall y \in \textit{self}.\text{getArrivals} \wedge x = y.\textit{product}\}$

[5] L'opération translateProducts retourne l'ensemble des entités associées aux produits traités par l'instance d'StationElement:

$$\text{translateProducts} = \left\{ x \;\middle|\; \begin{array}{l} x \in \text{Entity} \wedge \\ \forall y \in \textit{self}.\text{getProducts} \wedge \\ y.\textit{physicalelement} = \{x\} \end{array} \right\}$$

A.3.5 Paquetage *InformationalFlow*::*DecisionalMessage*

Le paquetage *DecisionalMessage* contient l'ensemble des définitions des structures utilisées pour modéliser les messages à caractère décisionnel.

A.3.5.1 Syntaxe abstraite

La figure A.21 illustre la syntaxe abstraite de cette bibliothèque.

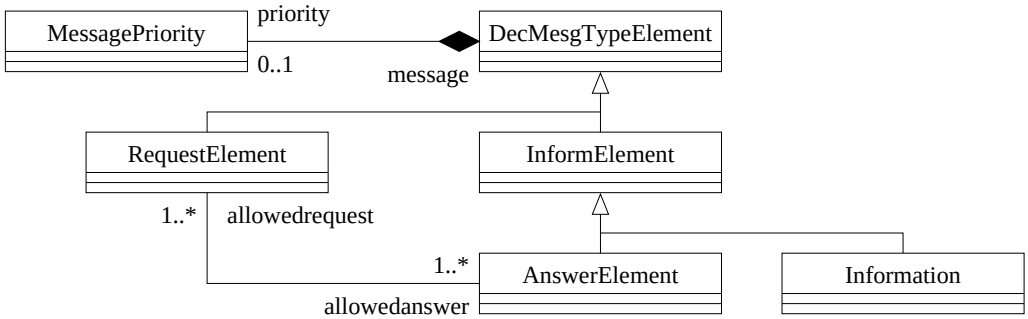


FIG. A.21 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*DecisionalMessage* : Type de messages

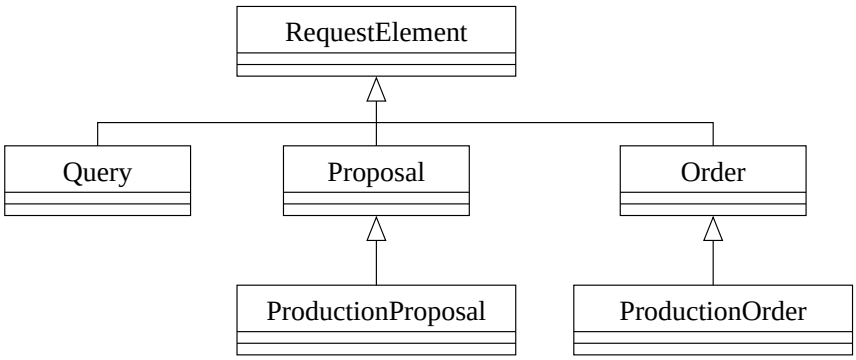


FIG. A.22 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*DecisionalMessage* : Requêtes

AcceptanceNotification

Cette classe représente les notifications d'acceptation.

DecMesgTypeElement

Cette classe d'objets représente l'abstraction maximale décrivant les types de messages à caractères décisionnels.

Associations

priority: Il est possible, par l'intermédiaire de cette association, de donner une priorité aux messages

InformElement

Cette classe représente l'ensemble des messages à caractère informatif.

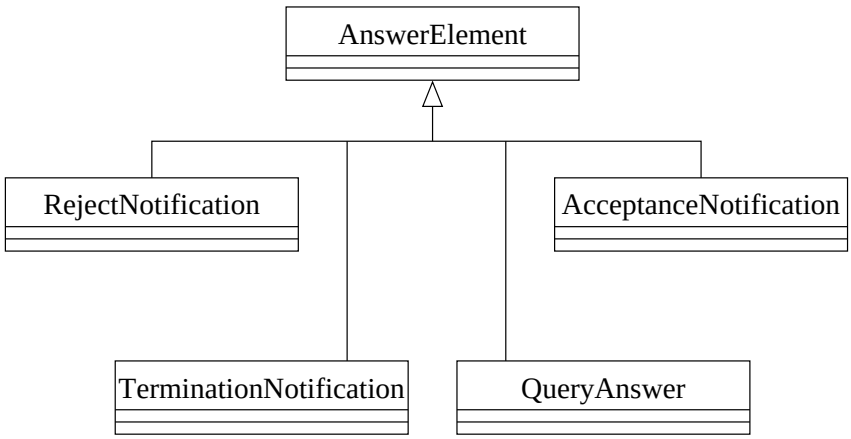


Fig. A.23 – Syntaxe abstraite de *Simulation*::*InformationalFlow*::*DecisionalMessage* : Informations

MessagePriority

Les messages à caractère décisionnel peuvent posséder un indicateur de priorité. La classe MessagePriority permet de modéliser les différents niveaux de priorité.

Associations

message: Représente le message possédant cette propriété.

Méthodes

+	priorTo(MessagePriority)	Méthode permettant de comparer deux instances de MessagePriority. Elle renvoi `true` si l'instance courante est plus prioritaire que l'instance passée en paramètre.

Order

Cette classe représente l'ensemble des messages correspondant à des ordres. Un ordre ne peut être refusé par le destinataire du message. Si toutefois cela se produit, ce dernier peut faire l'objet de l'application d'un malus.

ProductionOrder

Les ordres de production sont transmis grâce à ce type de message. Un ordre de production est soumis à la sémantique héritée de Order.

ProductionProposal

Les propositions de production sont transmis grâce à ce type de message. Cette proposition est soumis à la sémantique héritée de Proposal.

PROPOSAL

Cette classe représente l'ensemble des messages correspondant à des propositions. Une proposition peut être acceptée ou refussée par le destinataire du message.

QUERY

Cette classe représente l'ensemble des messages correspondant à des demandes d'informations.

QUERYANSWER

Cette classe représente les réponses à des demandes d'informations.

REJECTNOTIFICATION

Cette classe représente les notifications de rejet.

REQUESTELEMENT

La classe REQUESTELEMENT représente l'ensemble des messages contenant des requêtes. Ces dernières sont définies par les classes dérivées d'REQUESTELEMENT

Associations

allowedanswer: Correspond à l'ensemble des réponses attendues par cette requête.

TERMNIATIONNOTIFICATION

Cette classe représente les notifications d'accomplissement d'une requête.

A.4 Bibliothèque *DecisionalProcess*

La bibliothèque *DecisionalProcess* contient toutes les structures nécessaire à la construction de modèles du sous-système décisionnel.
Les paquetages *Organization* et *Behaviour* sont présentés dans les sections suivantes (*cf.* figure A.24 page suivante).

A.4.1 Bibliothèque *DecisionalProcess*::*Organization*

La bibliothèque *Organization* contient toutes les structures nécessaire à la construction de modèles du sous-système décisionel.

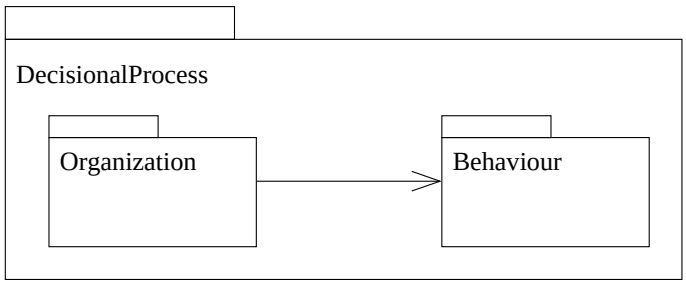


Fig. A.24 – Paquetage *DecisionalProcess*

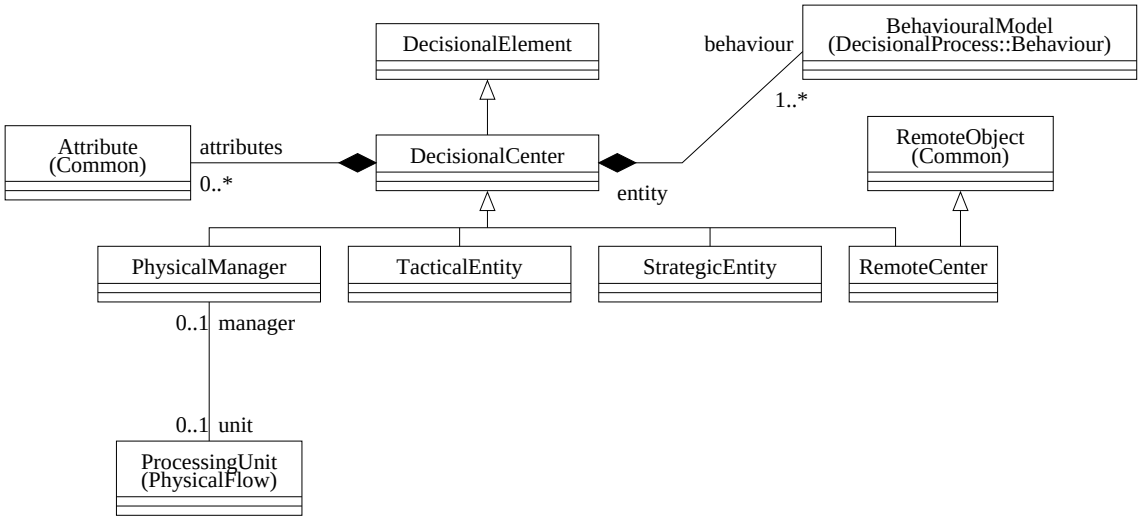


Fig. A.25 – *Simulation*::*DecisionalProcess*::*Organization* : Entités décisionnelles

A.4.1.1 Syntaxe abstraite

Les figures A.25 et A.26 illustrent la syntaxe abstraite de ce paquetage.

AuthorityLink

La classe AuthorityLink correspond aux liens d'autorités entre deux entités décisionnelles

DecisionalElement

Cette classe est l'abstraction de tous les éléments pouvant composer un modèle organisationel

DecisionalCenter

Les éléments de modélisation représentant les entités décisionnelles sont des classes héritiaires de DecisionalCenter

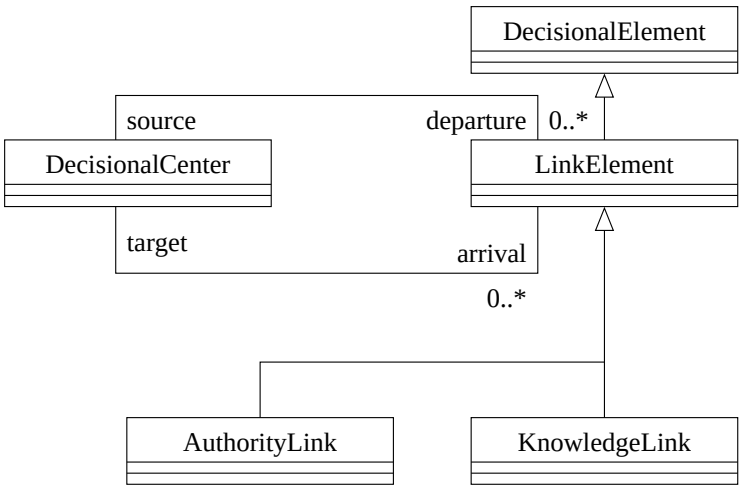


FIG. A.26 – *Simulation*::*DecisionalProcess*::*Organization* : Liens décisionnels

Associations

arrival: Correspond à l'ensemble des liens décisionnels entrants.

attributes: Représente l'ensemble de propriétés définies par le modélisateur pour une entité décisionnelle.

behaviour: Une entité décisionnelle peut avoir plusieurs comportements. Ces derniers sont représentés par cette association.

departure: Représente l'ensemble des liens décisionnels sortants.

KNOWLEDGELINK

La classe KNOWLEDGELINK correspond aux liens entre deux entités décisionnelles en exluant tout notion d'autorité. La direction du lien n'a plus aucune importance.

LINKELEMENT

Cette classe est l'abstraction des liens entre deux entités décisionnelles. Pour simplifier le formalisme, les liens ont une direction

Associations

source: entité décisionnelle d'où part le lien.

target: entité décisionnelle où abouti le lien.

PHYSICALMANAGER

Cette classe représente les entités décisionnelles directement responsables d'unités de productions. Dans la vision organisationnelle d'un système industrielle, cette classe correspond à une vision à court terme

Associations

unit: Représente l'unité physique supervisée.

RemoteCenter

La classe RemoteCenter correspond à l'utilisation d'une entité décisionnelle définie dans un modèle distant

StrategicEntity

Cette classe représente les entités décisionnelles ayant une vision stratégique de la gestion du système industriel

TacticalEntity

Cette classe représente les entités décisionnelles ayant une vision tactique de la gestion du système industriel

A.4.1.2 Règles de construction

Dans cette section nous décrivons formellement les règles de construction n'apparaissant pas dans la syntaxe abstraite.

DecisionalElement

[1] Un élément de modélisation de type DecisionalElement ne peut être présent qu'à l'intérieur d'un modèle des processus décisionnels:
$\forall x, y/(x, y) \in$ DecisionalElement $\times$ NameSpace,
$y \in x.namespace \Rightarrow y \in$ DecisionalModel

LinkElement

[1] Un lien doit relier deux entités décisionnelles différentes:
$\forall x, y, z/(x, y, z) \in$ LinkElement $\times$ DecisionalElement^2,
$y \in x.target \wedge z \in x.source \Rightarrow y \neq z$

A.4.2 Bibliothèque *DecisionalProcess*::*Behaviour*

La bibliothèque *Behaviour* contient toutes les structures nécessaire à la construction de modèles comportementaux pour le sous-système décisionel.

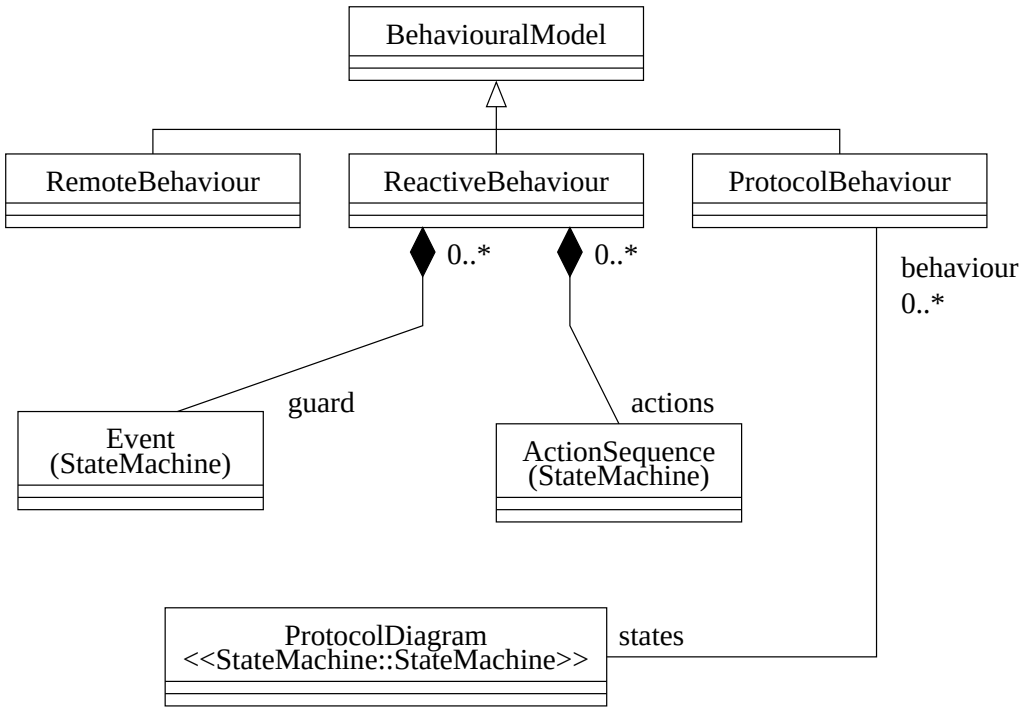


FIG. A.27 – *Simulation*::*DecisionalProcess*::*Behaviour*

A.4.2.1 Syntaxe abstraite

La figure A.27 illustre la syntaxe abstraite de ce paquetage.

BEHAVIOURALMODEL

Cette classe représente l'abstraction de tous les modèles comportementaux pouvant être utilisés dans le sous-système décisionnel.

Associations

entity: Correspond à l'entité décisionnelle utilisant ce modèle comportemental.

PROTOCOLDIAGRAM

Les instances de cette classes sont des définitions de protocoles de communication ou de coopération. Il s'agit d'un stéréotype d'une machine états-transitions.

Associations

behaviour: représente les modèles comportementaux utilisant ce protocole.

PROTOCOLBEHAVIOUR

La classe PROTOCOLBEHAVIOUR correspond à l'utilisation d'un comportement protocolaire *e.g.*, appels d'offres.

Associations

states: correspond à la définition du protocole utilisé.

ReactiveBehaviour

Cette classe représente les ocmportements réactifs *i.e.*, les comportements basés sur la réalisation d'une suite d'actions à chaque fois qu'une condition (guarde) est vérifiée.

Associations

actions: représente l'ensemble des actions associée à un modèle comportemental réactif.

guard: condition nécessaire pour que les actions soit réalisées.

RemoteBehaviour

Grâce à cette classe, il est possible d'utiliser un modèle comportemental défini dans un modèle décisionnel distant.

Annexe B

Grammaire pour les comportements réactifs

Cette annexe est un document de travail daté du 14 Février 2001.

Cette annexe décrit la grammaire utilisée pour décrire les comportements réactifs des centres de prise de décision (*cf.* section 9.2.3).

B.1 Grammaire

La grammaire est la suivante :

```
comportement ::= "CONTEXT" ident "WHEN" événement
                 "THEN" code "END"

événement ::= "EACH_TIME" '(' date ')'
              |
              "RECEIVE" ident [ "FROM" partenaire  ]
              [ "WITH PARAMS" paramètres  ]
              |
              condition

code ::= instruction ';' code |vide

paramètres ::= ident [ ',' paramètres  ]

partenaire ::= ident
               |
               "OPERATIONAL" ident
```

```
date ::= délai
       |
       heure
       |
       "INFORMATION" '(' ident expressions ')'

instruction ::= "SEND TO" ident "TYPE" type
                "NAMED" ident "DATA" expression
                |
                "FOREACH" '(' expression [ "AS" ident  ]  ')'
                "DO" code "DONE"
                |
                "WHILE" expression "DO" code "DONE"
                |
                "IF" expression "THEN" code [ "ELSE" code  ]
                "END IF"
                |
                "CALL" ident '(' ident expressions ')'
                |
                "OPERATION" '(' ident expressions ')'
                |
                "INFORMATION" '(' ident expressions ')'
                |
                ident '=' expression
                |
                vide

expressions ::= ',' expression expressions
                |
                vide

expression ::= ident
               |
               "OPERATION" '(' ident expressions ')'
               |
               "INFORMATION" '(' ident expressions ')'
               |
               code OCL

type ::= "ORDER"
       |
       "COMMUNICATION"
```

`condition ::= code OCL`

`ident ::= chaîne`

B.2 Sémantique

Dans cette section, nous décrivons brièvement la sémantique des éléments de langage appartenant à la grammaire décrite ci-dessus.

B.2.1 Entête

Chaque morceau de pseudo-code respectant la grammaire doit appartenir à un contexte. Un contexte est un centre de prise de décision. En dehor de celui-ci, le comportement décrit n'existe pas.
Ainsi, l'identificateur `ident` après le mot clef `CONTEXT` doit être l'identificateur d'un centre de décision. Cette définition est nécessaire et permet d'avoir accès aux informations et aux services attachées au centre (notamment utilisé par l'intermédiaire du mot clef `ATTACHED`).

B.2.2 Événement

Cette grammaire permettant de décrire un comportement réactif, il est nécessaire de pouvoir décrir un événement qui autorisera le déclenchement de l'exécution de la partie `code`.
Actuellement, nous considérons trois types d'événements :
- à interval ou à une date précise (`EACH_TIME`). La date spécifiée peut être un délai, une heure, ou une loi statistique définie dans le sous-système informationnel ;
- lorsqu'un message particulier est reçu par le centre de prise de décision (`RECEIVE`) ;
- lorsqu'une condition particulière est vérifiée (`condition`).

B.2.2.1 Expression d'une date

Une date peut être représentée de trois manières différentes :
- par un délai : `10j`, `+34`, ...
- par une heure précise : `Tue Oct 9 01 :06 :19 2001`, ...
- par l'utilisation d'une loi statistique définie dans le sous-système informationel. La syntaxe est la suivante :
  `INFORMATION( ident paramètres )` où `ident` est l'identificateur de la loi statistique dans le sous-système informationnel et `paramètres` sont nécessaires à l'obtention de ce service.

B.2.2.2 Description d'un message

La réception d'un message est définie selon la forme suivante :
`RECEIVE ident FROM partenaire WITH PARAMS paramètres` où
- `ident` est l'identificateur du message,
- `partenaire` est la description de l'émetteur du message. Il soit d'un identificateur d'un centre de décision, soit d'un objet appartenant au sous système opérationnel,
- `paramètres` sont les identificateurs des paramètres pour le message.

B.2.3 Code

Le code est la description du comportement du centre de décision. Il s'agit d'une séquence d'instruction pouvant être prises dans les suivantes :
- `ident = expression` est l'affectation d'une valeur calculée à une variable.
- `FOREACH`, `WHILE` et `IF` sont les instructions dérivées du langage Perl [CHRISTIANSEN et TORKINGTON, 2000].
- `CALL proc ( service epxressions )` est un appel de procédure extérieur que nous ne détaillerons pas ici.
- `OPERATION` et `INFORMATION` sont des instructions permettant d'avoir accès aux objets des sous-systèmes physiques et informationnel.
- `SEND TO recv TYPE type NAMED name DATA expressions` où :
  - `recv` est l'identificateur du destinataire du message,
  - `type` est le type du message : ordre ou coopération,
  - `name` est l'identificateur du message,
  - `expressions` sont des expressions calculées et passées comme paramètres au récepteur.

La théorie des types et de définition des variables est dériviée de celle utilisée dans le langage Perl [CHRISTIANSEN et TORKINGTON, 2000]. Nous n'entrerons pas plus dans les détails dans cette annexe.

Annexe C

Règles de conception

Cette annexe est un document de travail daté du 25 Février 2002.

Dans cette annexe, nous présentons un certain nombre de règles permettant d'obtenir un modèle de simulation multi-agent à partir d'un modèle abstrait. La définition de ces règles est précisée dans la section 10.2. Nous décrivons les règles que nous avons utilisées pour les sous-systèmes physiques et décisionnel. Le sous-système informationnel est transformé pour aboutir à la fois dans l'environnement SMA (sous forme d'objets comme les entités, les gammes opératoires, ...) et dans les agents (essentiellement dans les bases de connaissances). Les règles présentées respectent la syntaxe décrite dans la section 10.2.

C.1 Sous-système physique

Soit $\Re$ un comportement de passerelle, c'est-à-dire que chaque appel de procédure d'un agent correspond à un appel de service d'un objet de l'environnement.

RÈGLE C.1 : ENTITYGENERATOR

Un générateur d'entité fait l'objet d'une traduction sous forme d'un objet de l'environnement. Comme l'artefact de spécification est piloté par un centre opérationnel de décision, aucune loi stochastique n'est attachée à l'objet de l'environnement.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{EntityGenerator} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \\ \left\{ \langle \texttt{generate\_entity}, method, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \right\}, \\ \texttt{\_.getAttributes()} \end{array} \right\rangle \right\}$$

$$\texttt{AGENT} \leftarrow \texttt{EntityGenerator} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \emptyset, \\ \{\texttt{EntityGenerator}\}, \\ \left\{ \langle \texttt{generate\_entity}, method, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \right\}, \\ \texttt{\_.getAttributes()}, \Re, \emptyset \end{array} \right\rangle \right\}$$

RÈGLE C.2 : INCOMINGFLOW

Le point d'entrée doit être transformé en un objet de l'environnement et en un agent pilotant ce dernier.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{IncomingFlow} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \left\{ \langle \texttt{entity\_received}, event, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \right\}, \\ \texttt{\_.getAttributes()} \end{array} \right\rangle \right\}$$

$$\texttt{AGENT} \leftarrow \texttt{IncomingFlow} = \left\{ \langle \texttt{\_.name}, \emptyset, \{\texttt{\_.name}\}, \emptyset, \texttt{\_.getAttributes()}, \texttt{\_.getBehavior()} \rangle \right\}$$

Règle C.3 : OutgoingFlow

Le point de sortie doit être transformé en un objet de l'environnement et en un agent pilotant ce dernier.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{OutgoingFlow} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \left\{ \langle \texttt{forward\_entity}, event, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \right\}, \\ \texttt{\_.getAttributes()} \end{array} \right\rangle \right\}$$

$$\texttt{AGENT} \leftarrow \texttt{OutgoingFlow} = \left\{ \langle \texttt{\_.name}, \emptyset, \{\texttt{\_.name}\}, \emptyset, \texttt{\_.getAttributes()}, \texttt{\_.getBehavior()} \rangle \right\}$$

Règle C.4 : Resource

Une resource est un objet placé dans l'environnement SMA *pour être utilisé par les outils de simulation. Son allocation et sa libération sont réalisées par l'intermédiaire d'un agent.*

$$\texttt{ENVIRONMENT} \leftarrow \texttt{Resource} = \left\{ \left\langle \begin{array}{l} \texttt{\_.name}, \left\{ \begin{array}{c} \langle \texttt{seized}, event, \emptyset \rangle, \\ \langle \texttt{released}, event, \emptyset \rangle, \\ \langle \texttt{seize}, method, \emptyset \rangle, \\ \langle \texttt{release}, method, \emptyset \rangle, \end{array} \right\}, \\ \texttt{\_.getAttributes()} \end{array} \right\rangle \right\}$$

$$\texttt{AGENT} \leftarrow \texttt{Resource} = \left\{ \begin{array}{c} \langle \texttt{\_.name}, \emptyset, \{\texttt{\_.name}\}, \left\{ \begin{array}{c} \langle \texttt{seized}, event, \emptyset \rangle, \\ \langle \texttt{released}, event, \emptyset \rangle, \\ \langle \texttt{seize}, method, \emptyset \rangle, \\ \langle \texttt{release}, method, \emptyset \rangle, \end{array} \right\}, \\ \texttt{\_.getAttributes()}, \texttt{\_.getBehavior().toPseudoCode()} \rangle \end{array} \right\}$$

RÈGLE C.5 : REMOTERESOURCE

Une resource distante est un objet placé dans l'environnement SMA *pour être utilisé par les outils de simulation. Son allocation et sa libération sont réalisées par l'intermédiaire d'un agent distant proposant le service* `RemoteResource.name`.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{RemoteResource} = \left\{ \left\langle \texttt{\_.name}, \left\{ \begin{array}{c} \langle \texttt{seized}, event, \emptyset \rangle, \\ \langle \texttt{released}, event, \emptyset \rangle, \\ \langle \texttt{seize}, method, \emptyset \rangle, \\ \langle \texttt{release}, method, \emptyset \rangle, \end{array} \right\}, \texttt{\_.getAttributes()} \right\rangle \right\}$$

Danger *: cette règle doit être vérifier et réécrite pour formaliser l'enregistrement d'une ressource auprès d'un agent distant.*

RÈGLE C.6 : PROCESSINGUNIT

Nous considérons que les unités de traitement du sous-système opérationnel permet d'être entièrement générées dans l'environnement SMA.

$$\texttt{ENVIRONMENT} \leftarrow \texttt{ProcessingUnit} = \left\{ \left\langle \texttt{\_.name}, \left\{ \begin{array}{c} \langle \texttt{getState}, method, \emptyset \rangle, \\ \langle \texttt{beginProcess}, event, \{\langle \texttt{entité}, \texttt{entité}, \alpha \rangle\} \rangle, \\ \langle \texttt{endProcess}, event, \{\langle \texttt{entité}, \texttt{entité}, \gamma \rangle\} \rangle, \end{array} \right\}, \texttt{\_.getAttributes()} \cup \left\{ \begin{array}{c} \langle \texttt{delay}, Law, \texttt{"Law\_for\_"+\_.name} \rangle, \\ \langle \texttt{fail}, Law, \texttt{"Fail\_Law\_for\_"+\_.name} \rangle \end{array} \right\} \right\rangle \right\}$$

RÈGLE C.7 :

Soit α un élément de modélisation n'étant pas pris en compte par les règles précédentes. Un élément de modélisation du sous-système physique est automatiquement placé dans l'environnement du système multi-agents.

$$\texttt{ENVIRONMENT} \leftarrow \alpha = \left\{ \left\langle \texttt{\_.name}, \emptyset, \texttt{\_.getAttributes()} \right\rangle \right\}$$

C.2 Sous-système décisionnel

Règle C.8 : OperationalCenter

Chaque centre de prise de décision opérationnel est traduit en un agent de simulation ayant le même comportement.

```
AGENT [ OperationalCenter ]← OperationalCenter =
  A {
    IF OperationalCenter.behavior INSTANCEOF
       AlgorithmicBehavior THEN
      Behaviour = OperationalCenter.behavior.toPseudoCode
    END IF
  } ∧
  E {
    IF OperationalCenter.behavior INSTANCEOF
       ReactiveBehavior THEN
       Behaviour = OperationalCenter.behavior.toPseudoCode
    END IF
  } ∧
  I {
    IF OperationalCenter.behavior INSTANCEOF ProtocolBehavior
    THEN
      Behaviour = OperationalCenter.behavior.toPseudoCode
    END IF
  } ∧
  O { } ∧
  X {
    S = {x / x ∈ S_OperationalCenter.PhysicalElement}
    ⟨OperationalCenter.name,
      S,
      { OperationalCenter.PhysicalElement },
      {service_x / x ∈ S},
      OperationalCenter.getAttributes()⟩
  }
}
```

RÈGLE C.9 : DECISIONALCENTER

Chaque centre de prise de décision est traduit en un agent de simulation ayant le même comportement.

```
AGENT [ DecisionalCenter ]← DecisionalCenter =
  A {
    IF _.behavior INSTANCEOF
       AlgorithmicBehavior THEN
      Behaviour = _.behavior.toPseudoCode
    END IF
  } ∧
  E {
    IF _.behavior INSTANCEOF
       ReactiveBehavior THEN
       Behaviour = _.behavior.toPseudoCode
    END IF
  } ∧
  I {
    IF _.behavior INSTANCEOF ProtocolBehavior
    THEN
      Behaviour = _.behavior.toPseudoCode
    END IF
  } ∧
  O { } ∧
  X {
    ⟨DecisionalCenter.name,
      ∅, ∅, ∅,
      _.getAttributes()⟩
  }
}
```

Les liens entre les centres de décisions sont traduits en relations conformes à MOISE [HANNOUN et al., 2000].