Spécification, Synthèse et Qualité des Modèles de Simulation à Événements Discrets

Mamadou Kaba Traoré

Spécification, synthèse et qualité des modèles de simulation discrète

Mamadou Kaba Traoré

Sommaire

Introduction

La simulation informatique, maintenant identifiée dans la communauté scientifique sous le label M&S (pour Modélisation & Simulation), est utilisée pour accroître les capacités d'aide à la décision, surtout pour les problèmes complexes, peu manipulables par des moyens autres que les ordinateurs. Il s'agit d'approcher le comportement d'un système dynamique réel ou virtuel par celui d'un modèle digital, d'utiliser ce dernier comme source d'expérimentations, et d'en interpréter les résultats pour prendre des décisions sur le système. Une compilation de définitions historiques est donnée dans [Pritsker 1979].

Ayant été longtemps dirigée surtout par la pratique, le champ de la M&S est restée en quête d'une théorie générale, unificatrice des diverses approches adoptées, via une vision universelle des modèles transversale à tous les domaines d'application, et une interprétation cohérente de toutes les transformations qui caractérisent l'évolution d'un modèle dans son cycle de vie. La très grande hétérogénéité, à la fois des méthodes de M&S et des systèmes considérés, est un verrou à cette unification.

Cette quête est souvent envisagée de deux manières, duales l'une de l'autre :

- par une perspective que nous appellerons « Problème-centrée », i.e. en unifiant des concepts et méthodes propres à un domaine d'étude donné (évaluation des performances, vérification et validation, optimisation...) de manière transversale aux types de systèmes à étudier ; ou
- par une perspective que nous appellerons « Système-centrée », i.e. en construisant un paradigme autour d'une classe (parfois la plus large possible, parfois ciblée à dessein) de systèmes (manufacturiers, économiques, écologiques, continus, discrets, hybrides…), et ce de manière transversale aux types de problèmes à résoudre.

La littérature regorge de démarches Système-centrées. Certaines sont de type ascendant, où à partir de l'expérience acquise par modélisation de plusieurs systèmes, apparaît la nécessité de formaliser un plus grand dénominateur commun à ces systèmes, pour lequel il est possible de définir des outils et des

méthodes. Des exemples notables sont : [Cellier 1991] pour les systèmes continus, [Adiga & Glassey 1991], [Basnet & Mize 1995] ou [Kellert et al. 1997] pour les systèmes manufacturiers, [Coquillard & Hill 1997] pour les écosystèmes, [Young & Rato 2008] pour les systèmes environnementaux, [Mahzer 2000] pour les systèmes sociaux. D'autres sont de type descendant, où l'analyse du domaine permet d'offrir d'emblée une perspective commune pour une vaste famille de systèmes. Des exemples notables sont : [Arango & Prieto-Diaz 1991] pour les systèmes de production, et [Gregoriades & Karakostas 2004] pour les processus métiers. Un pas important a été franchi avec l'avènement du paradigme DEVS (*Discrete EVent systems Specification*), qui a jeté des bases solides pour l'émergence d'une véritable théorie de la M&S [Zeigler 1976]. DEVS propose une unification des 3 grands paradigmes de modélisation pour simulation que sont : la spécification continue des systèmes continus (DESS), la spécification discrète des systèmes discrets (DEVS initial), et la spécification synchronisée des systèmes (DTSS). Les éléments théoriques établis à partir de cette unification ont permis d'inclure 3 autres paradigmes de modélisation : la spécification unifiée (discrète et continue intégrées) des systèmes hybrides, i.e., incluant à la fois des aspects continus et discrets (DEV&DESS), la spécification combinée des systèmes hybrides (par couplages de spécifications de type DEVS, DTSS et/ou DESS), et la spécification discrète des systèmes continus, soit par discrétisation du continu (DESS vers DTSS), soit par quantization du continu (DESS vers DEVS). L'approche SES/MB (*System Entity Structure/Model Base*) est la méthode associée permettant de définir la hiérarchie entre tous les composants du système à décrire [Zeigler 1984].

De manière similaire, les démarches Problème-centrées peuvent être de type ascendant ou descendant. Par exemple, dans la perspective Problème-centrée d'optimisation par simulation, la technique de surface de réponse [Box & Wilson 1951], qui a pour but d'explorer les relations entre les variables dépendantes et indépendantes impliquées dans une expérience, et ce de manière indifférenciée selon les types de système considérés, est descendante. Par contre, les efforts de généralisation de pratiques liées à la simulation (en particulier, l'évaluation des performances) procèdent d'une démarche Problème-centrée ascendante [Hill 1993], [Cubert & Fishwick 1998]. Un panorama des techniques de simulation orientée objet est donné par [Roberts & Dessouky 1998]. La Table 1.1 résume l'essence de ces différentes approches d'unification.

Table 1. Démarches d'unification en M&S

	Démarche ascendante	**Démarche descendante**
Perspective Problème-centrée	Identification et définition de motifs récurrents de résolution, transversaux à tout type de système	Construction de schéma générique de résolution d'un problème spécifique
Perspective Système-centrée	Identification et définition d'une architecture commune de système dans un domaine donné, servant de référence pour l'application de toute méthode de résolution	Construction de méta-modèle de système, comme description générique utilisable par toute méthode de résolution

Nous nous intéressons, dans cet ouvrage, aux démarches descendantes. Nos préoccupations s'articulent autour des trois questions suivantes :

- Quels sont les concepts nécessaires à la bonne conduite d'un processus de M&S, et ce de manière transversale aux problèmes et aux systèmes ?
- Quels produits logiciels faut-il construire in fine ?
- Quelles en sont les propriétés algébriques et opérationnelles ?

L'objectif est d'aller vers une compréhension générique des modèles de simulation dans l'intégralité de leur cycle de vie, pour une meilleure conception et un usage de meilleure qualité. Il nous apparaitra, plus loin, que ces trois questions nous mènent respectivement à des problématiques de spécification, de synthèse de code, et de qualité (aussi bien de la spécification que de son code).

Chapitre 1.
Cycle de vie de M&S

Par système dynamique, la théorie des systèmes entend une entité dotée d'une structure et d'un comportement (Figure 1). La structure exprime, d'une part l'état dans lequel l'entité se trouve et les lois internes qui le gouvernent, et d'autre part, l'interface par laquelle elle interagit avec son environnement par réception de stimuli et émission de réponses. Le comportement se traduit par les traces d'évolution dans le temps de cette interface et de cet état.

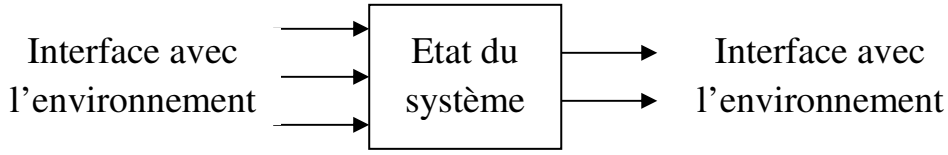


Figure 1. Système dynamique et environnement

1.1. Comprendre les modèles dans leur cycle de vie

Un modèle de simulation est, in fine, un produit logiciel. Son élaboration et son utilisation s'inscrivent donc dans un cycle de vie, dans le sens du Génie Logiciel (i.e., une succession d'étapes que traverse un produit dans le temps, de la gestation à la maturation, voire au déclin). Une approche rationnelle à la compréhension des modèles de simulation est donc celle de ce cycle lui-même.

Le cycle de vie que nous proposons en Figure 2 (en réalité, seulement la première partie de notre cycle complet) est une mise en perspective de la double finalité que [Shanon 1975] confère à la M&S (à savoir : *comprendre* ou *résoudre*) dans le contexte de la dualité structure-comportement que prône la théorie des systèmes.

Cette première partie de cycle de vie montre sept nœuds, à savoir :

(1) le système
(2) sa structure,
(3) son comportement,
(4) la structure du modèle théorique, i.e., les propriétés et règles,
(5) le comportement du modèle théorique, i.e., les résultats théoriques,
(6) la structure du modèle de simulation, i.e., son expression, et
(7) le comportement du modèle de simulation, i.e., ses trajectoires.

La seconde partie du cycle de vie sera révélée dans la dernière section de ce chapitre (séparation des préoccupations) et fera le focus sur l'ingénierie de ce modèle de simulation.

Nous inscrivons ces nœuds dans un continuum de huit espaces :

(1) l'espace des applications,
(2) celui des problèmes,
(3) celui de leurs objectifs,
(4) celui des exigences du réel,
(5) celui des hypothèses de modélisation,
(6) celui des expressions théoriques formelles,
(7) celui des contraintes, et
(8) celui des calculs (ce dernier étant l'endroit où sont exprimés les modèles de simulation).

Les nœuds sont reliés par des activités se situant dans le contexte d'un ou de plusieurs de ces espaces, et exprimant la double finalité mise en exergue par Shanon. Certaines activités se situent dans le contexte d'un seul espace, alors que d'autres sont transversales à plusieurs espaces. Notons bien que cette représentation par espaces est plus intuitive que formelle et qu'elle exprime surtout la nécessité, pour passer des applications à leur étude par simulation, d'examiner et de prendre en compte ce que chaque espace couvre.

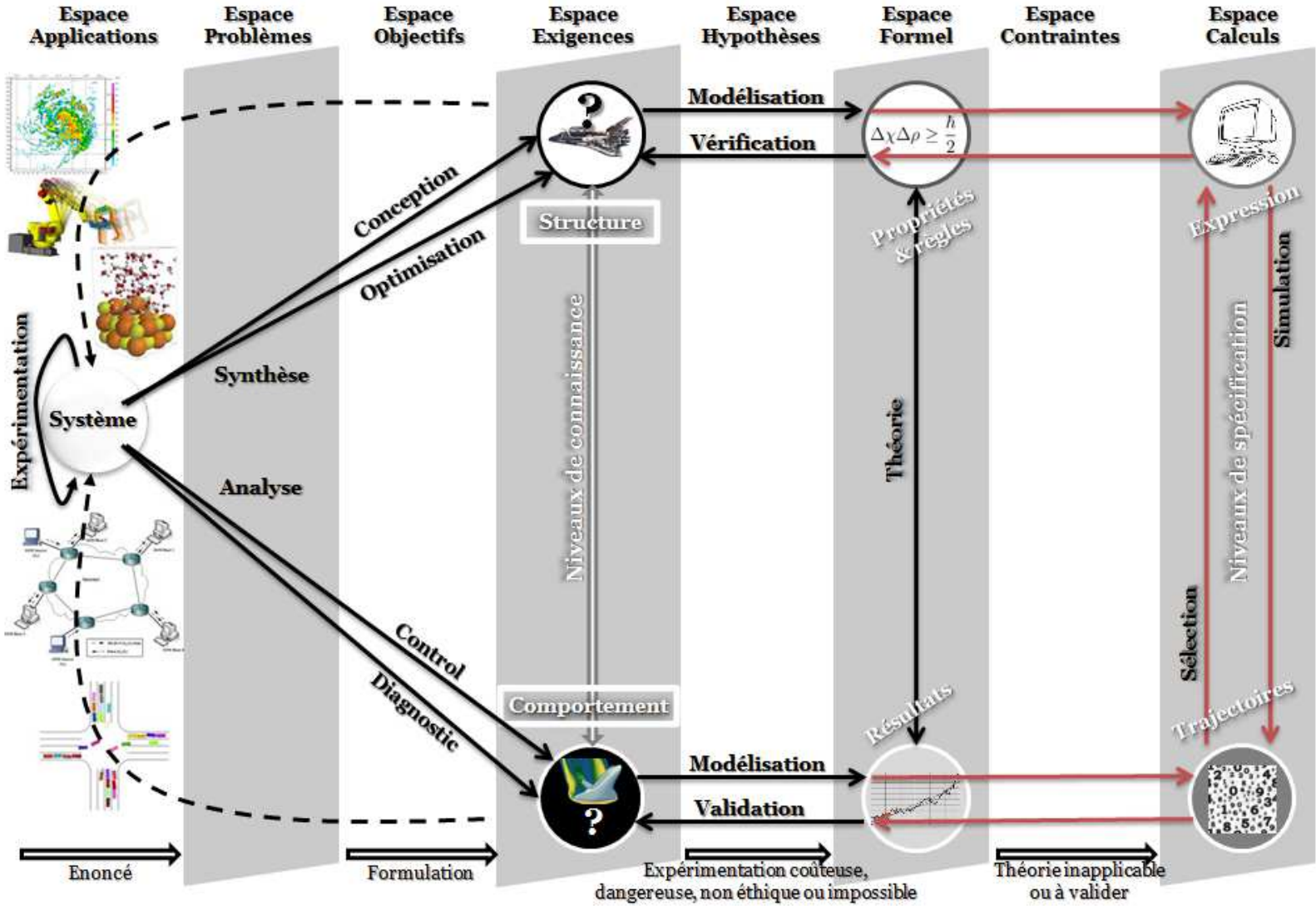


Figure 2. Cycle de vie de M&S (1ère partie)

1.1.1. Espace des applications

L'aide à la décision est au cœur d'un nombre important de grandes applications (transport, biomédecine, environnement, réseaux informatiques, télécommunication, robotique, nucléaire, etc.). Sont concernées, toutes les disciplines scientifiques attachées à l'étude de systèmes naturels ou artificiels porteurs de problèmes qui mettent en jeu plusieurs paramètres fortement corrélés dans le temps et dans l'espace. Ces problèmes sont qualifiés de complexes pour exprimer la grande variété et le grand nombre des relations internes, ainsi que l'imbrication de multiples liens de causalité. Un problème complexe inclut un (ou plusieurs) système(s) réel(s) et une ou plusieurs questions auxquelles l'étude tente de répondre. L'expression " système réel " est une vue de l'esprit, dans la mesure où le système en question peut ne pas exister. Le qualificatif " réel " est utilisé pour distinguer l'objet d'étude de sa représentation. L'expression " problème réel " est également utilisée dans le même esprit (le réel désignera donc de manière indifférente le problème réel ou le système réel).

Les réels de très grande complexité se retrouvent aujourd'hui dans les domaines issus des sciences de la terre et de la vie, des sciences sociales et humaines, des sciences de l'ingénieur, ou encore des sciences économiques et de gestion.

Parmi les pratiques d'aide à la décision par simulation, citons, entre autres :

- l'évaluation des performances (souvent vue comme discipline à part entière) qui permet d'extraire une connaissance quantitative du comportement des systèmes, et qui sert souvent les besoins de méthodes de prescription hybrides entre la simulation et les techniques itératives décisionnelles (recherche opérationnelle, optimisation combinatoire, intelligence artificielle...) ;
- le calcul intensif qui procède, via l'usage de super ordinateurs, à de larges séquences de simulation (on trouve dans cette catégorie des simulations dites de Monte Carlo, mais également bien sur la résolution de modèles numériques de grande taille, comme ceux de la mécanique des fluides, du génie civil, du génie climatique…) ;
- le temps réel qui, dans le cas de calculs très rapides, permet d'implanter un module décisionnel correctif sous contraintes et très interactif avec le système physique qu'il pilote ;

- la réalité virtuelle qui est à but d'entraînement et d'apprentissage par le biais d'une visualisation animée (ouvrant une large voie dans la médecine, la robotique, l'aéronautique, les applications militaires, la logistique d'urgence…) ;
- la vie artificielle qui procède d'une approche prospective génératrice de mondes nouveaux, et qui, au-delà du débat sur la légitimité des modèles manipulés, est à la base de l'exploration d'univers émergeants et des systèmes dits complexes.

De manière très traditionnelle, le traitement du réel peut donner lieu à des ***expérimentations*** sur le système correspondant. Dans bien de cas, ces expérimentations ne sont pas envisageables, à savoir :

- quand les coûts financiers engendrés sont importants (par exemple, lorsque le matériel d'expérimentation est cher ou lorsque les expériences conduisent à la destruction de composants onéreux), ou
- quand les durées de réalisation de ces expériences ne sont pas raisonnables (par exemple, lorsque les résultats à observer prennent des années avant d'être visibles), ou
- quand les risques sociaux engendrés sont importants (par exemple, lorsque l'environnement d'expérimentation est hostile à la vie, ou lorsque l'expérience conduit à des pertes de vie), ou
- quand ceci pourrait perturber le système dans son fonctionnement (par exemple, lorsque la présence des instruments d'expérimentation altèrent le comportement, voire la structure des entités du système), ou
- quand il est impossible d'accéder au système par des moyens d'expérimentation physique directe (par exemple, lorsque le système est à des années-lumière, ou qu'il n'existe pas et est à concevoir), ou
- quand l'éthique scientifique interdit une telle pratique (par exemple, lorsqu'il s'agit d'expérience sur la vie humaine ou animale à caractère polémique), ou encore
- quand les expériences sont non reproductibles (par exemple, lorsque des phénomènes irréversibles, telles la modification de structure ou de comportement voire la destruction du système, se produisent à l'issue de chaque expérience, interdisant toute possibilité de procéder à une nouvelle expérience dans des situations identiques).

L'alternative est alors l'étude par usage de modèle (analogique, théorique, ou de simulation). Les modèles analogiques sont des miniatures de systèmes réels (par exemple, une maquette de véhicule ou de bâtiment, une balle de ping-pong pour représenter un boulet de canon, une piscine pour représenter l'espace, etc.). Ce sont aussi des systèmes réels, et la seule différence avec les systèmes dont ils sont les miniatures se situe, dans le contexte de cet ouvrage, au niveau de l'échelle de travail. Nous sommes plutôt intéressés ici par les autres types de modèles, et en particulier par les modèles de simulation.

1.1.2. Espace des problèmes

L'amorce d'une étude (par simulation, ou même de manière générale) vient souvent de l'annonce d'un problème depuis un domaine d'application donné (ce que [Sadowski 1989] appelle élucidation). Elle implique deux phases, conduites par des échanges entre les experts du domaine d'étude et les concepteurs de solution, qui sont :

- l'***énoncé*** du problème et
- la ***formulation*** des objectifs.

L'énoncé du problème est la phase où l'objet et les finalités de l'étude sont précisés. Dans notre cycle de vie, il s'agit de se mouvoir d'un nœud du système (structure ou comportement) vers l'autre. Ceci correspond, de manière générique, à deux classes de problème, à savoir :

- ***analyse*** lorsqu'il est question d'étudier le comportement à partir de la connaissance de la structure, et
- ***synthèse*** lorsqu'il est question de trouver une structure capable de générer le comportement constaté ou espéré.

Sur la Figure 2, que ce qui est inconnu d'un nœud du système est représenté par un point d'interrogation, alors que ce qui est connu est représenté par une icône (carlingue de navette spatiale pour représenter le concept de structure, flux engendré par la navette en mouvement pour représenter le concept de comportement). Très souvent, dans chaque nœud, on ne dispose que de connaissances partielles (une petite part de connu, et une grande part d'inconnu).

1.1.3. Espace des objectifs

Dans la formulation des objectifs, le nœud de départ est celui pour lequel les connaissances sont suffisamment connues ; ces connaissances peuvent être :

- quantitatives (données numériques issues de mesures physiques et/ou d'observations faites sur le système), ou
- qualitatives (éléments structurels, contraintes et règles comportementales extraites des expertises du domaine d'application).

Dans un cas de synthèse (comportement vers structure), l'objectif se ramène à l'une des deux classes suivantes :

- ***conception***, i.e., proposition d'une ou de plusieurs alternatives de structure permettant de générer un comportement attendu, ou
- ***optimisation***, i.e., détermination de la configuration ou des paramètres de fonctionnement qui permettent d'atteindre au mieux le but visé ; la nature du problème amène souvent à se restreindre à un objectif d'amélioration (l'optimum pouvant être atteint sans qu'on puisse le prouver).

Dans un cas d'analyse (structure vers comportement), l'objectif se ramène à l'une des deux classes suivantes :

- ***diagnostic***, i.e., compréhension liens existant entre les événements qui prennent place au sein du système. Ceci inclut la capacité à prédire ce qui arriverait lorsque le système est placé dans une situation initiale donnée, ainsi qu'à en évaluer les performances selon des critères définis (pour un système conçu par l'homme, il s'agit souvent de savoir quelle serait sa réaction face à des situations ne correspondant pas un fonctionnement nominal connu ; pour un système naturel, il s'agit souvent d'un travail de prospective visant à mettre à jour des futurs inconnus [Simon 1969]) ;
- ***control***, i.e., pilotage du comportement d'un système par prescription de commandes (pouvant être correctives) ; ces prescriptions sont élaborées, soit en ligne, i.e., à mesure que le système évolue (pour le guider, ou alors à but d'entraînement de celui qui le pilote), soit hors ligne, i.e., avant la mise en action du système.

La Table 2 présente quelques exemples d'applications dont les problèmes et objectifs sont élucidés selon la terminologie adoptée en Figure 2.

Table 2. Exemples d'étude

Applications et questions	**Problèmes correspondants**	**Objectifs correspondants**
Trafic urbain : (1) Pourquoi, quand et où naissent des embouteillages dans un réseau de transport urbain dont l'architecture et les éléments de circulation sont connus ? Qu'arriverait-t-il si la vitesse autorisée est réduite de 10% ? (2) Que se passe-t-il avec un conducteur au volant d'un véhicule dans ce trafic, au fil de ses décisions de conduite ? (3) Quel est le réglage des feux de signalisation qui serait le plus bénéfique à la fluidité du trafic ? (4) Comment construire une portion nouvelle du trafic pour faire face à la situation dégradée récurrente ? (5) Combien de feux faut-il placer dans le réseau, où les positionner et comment les régler pour que le débit de véhicules sortant du réseau par jour soit maximal et le nombre d'accidents minimal ?	(1-3) Analyse (structure connue : véhicules, voies, règles de circulation, etc. ; comportement à identifier : formation de clusters très denses de véhicules dans des secteurs du réseau) (4-5) Synthèse (comportement attendu : plus grande fluidité du trafic ; structure à trouver : largeur et agencement des voies et éléments de régulation)	(1) Diagnostic (pourquoi = compréhension ; quand et où = prédiction ; quoi si = exploration de futurs possibles ; comment = évaluation des performances) (2) Control (deux perspectives envisageables : entraînement de conduite avisée pour le conducteur, ou guidage en ligne de conduite pour désengorger le trafic) (3) Control (guidage hors ligne pour des feux normaux, guidage en ligne pour des feux intelligents) (4) Conception (5) Optimisation

Météo : quel temps fera-t-il dans les prochains jours à partir de la situation météorologique du moment ?	Analyse (structure connue : pression, température, humidité, vents, lois physiques, etc. ; comportement à trouver : formation de nuages, pluie, neige, etc.)	Diagnostic (prédiction)
Nucléaire : quel est le niveau de résistance d'une installation située sur les côtes en cas de tsunami ?	Analyse (structure connue : architecture de la centrale, dispositifs de sécurité ; comportement à trouver : ruptures mécaniques, radioactivité, etc.)	Diagnostic (exploration de futurs possibles)
Diffusion de feu : (1) Quelle est la cartographie des zones brûlées d'une forêt en cas de départ de feu en été depuis un point situé en altitude ? (2) Quelle stratégie doit être adoptée par les pompiers pour lutter le plus efficacement contre une propagation de feu en cours ? (3) Comment aménager le territoire pour sécuriser la faune et les habitats contre les aléas des feux de forêt ?	(1-2) Analyse (structure connue : géographie du terrain, flore, météo, etc. ; comportement à identifier : dégâts, diffusion du front de feu, phénomènes émergents,) (3) Synthèse (comportement attendu : zones brûlées non menaçantes ; structure à trouver : position des barrières de feu et des espèces vivantes, définition des zones constructibles)	(1) Diagnostic (exploration de futurs possibles) (2) Control (guidage en ligne) (3) Conception

Système économique : quel est l'impact socio-économique d'une augmentation de la TVA de 2% ?	Analyse (structure connue : composantes socio- économiques et leurs liens ; comportement à trouver : variations du chômage, de la consommation...)	Diagnostic (prédiction)
Jeu : (1) Comment faire jouer aux échecs un algorithme contre un humain ? (2) Comment faire jouer aux échecs un humain contre un algorithme ?	Analyse (structure connue : plateau, pions et positions, règles du jeu ; comportement à trouver : prochains coups à jouer)	(1) Control (guidage en ligne) (2) Diagnostic (entraînement)
Système de production : (1) Quelle est la suite de mouvements qu'un robot de manutention, en charge du transport de produits vers différentes machines d'usinage, doit suivre pour permettre à l'usine de produire correctement ? (2) Comment corriger ces mouvements une fois implantés sur site, en cas de panne ou de rupture de stock ? (3) Comment disposer les machines et les zones de stockage pour une productivité maximale ?	(1-2) Analyse (structure connue : vitesse et capacité du robot, rapidité et capacité des machines, etc. ; comportement à trouver : suite de mouvements à effectuer) (3) Synthèse (comportements connu et attendu respectivement : mouvements du robot, et niveau satisfaisant de productivité ; structure à trouver : position des machines et des stocks)	(1) Control (guidage hors ligne) (2) Control (guidage en ligne) (3) Optimisation

En résumé, l'énoncé du problème et la formulation des objectifs nous amène à une transition de nœud à nœud, que l'on pourrait décrire ainsi :

- Analyse : transition de la structure vers le comportement
  - Diagnostic : nous savons comment le système est fait (sa structure) et nous désirons savoir comment il évoluera dans le temps (son comportement).
  - Control : nous avons la structure du système et nous désirons lui dicter son comportement face à son environnement (en réalité, nous ne pouvons que l'inciter vers des décisions de comportement espérées en lui imposant des stimuli d'entrée appropriés).
- Synthèse : transition du comportement vers la structure
  - Conception : nous voulons obtenir un comportement donné et nous cherchons à mettre au point une structure qui générerait ce comportement.
  - Optimisation : nous connaissons le comportement du système que nous cherchons à améliorer selon nos critères, et nous cherchons les modifications à apporter à sa structure pour y parvenir.

1.1.4. Espace des exigences

Dans la théorie générale des systèmes, la transition d'un des nœuds système (structure ou comportement) vers l'autre traduit une traversée de ***niveaux de connaissances*** [Klir 1969], que Georges Klir hiérarchise (du niveau le moins riche en connaissance au niveau le plus riche) de la manière suivante :

Niveau 0. « *Source* », i.e. la connaissance à propos des variables observées et mesurées du système.

Niveau 1. « *Données* », i.e. l'ensemble des données collectées sur les variables observées et mesurées.

Niveau 2. « *Génératif* », i.e. les mécanismes par lesquels ces données sont générées.

Niveau 3. « *Structure* », i.e. la composition de composants produisant le système génératif global.

Ainsi, plus nous sommes proches de la structure, plus nous possédons de connaissances sur le système (sommet de la hiérarchie). Cette connaissance est sous une forme condensée que la descente de la hiérarchie (analyse) permet de

décanter de manière à révéler le comportement du système. Par contre, la remontée de cette hiérarchie (synthèse par inférence) nécessite qu'à chaque niveau, des conditions d'unicité soient établies pour pouvoir passer au niveau supérieur (plusieurs niveaux de connaissance plus riches pouvant, par décantation, produire un même niveau de connaissance moins riche).

Pour établir une correspondance stricte entre cette hiérarchie et les concepts de Structure et de Comportement, il est possible de voir le comportement d'un système comme le Niveau 1 de la hiérarchie de connaissance (en acceptant que le Niveau 0 ne soit autre qu'une identification formelle des variables quantifiées au Niveau 1). Il est également possible de voir la Structure d'un système comme le Niveau 2 (dans les cas où cette description est disponible et opérationnelle) ou le Niveau 3 (dans les cas où la description directe et opérationnelle du Niveau 2 est plus difficile à obtenir, et qu'il est préférable d'adopter une démarche cartésienne consistant à « diviser pour régner ») de la hiérarchie de connaissance.

Par Espace des exigences, nous entendons l'ensemble des connaissances que nous impose le réel, et par dualité l'ensemble des inconnues de la situation (à ne pas confondre donc avec le concept d'exigence tel que le Génie logiciel l'introduit à travers l'analyse des besoins, même s'il est possible de voir une certaine proximité entre ces concepts). La notion de hiérarchie de connaissance capture bien ces notions de connaissance et d'inconnues.

1.1.5. Espaces des hypothèses, du formel et des contraintes

La démarche scientifique dite « traditionnelle » pour réaliser les transitions nœud système à nœud système repose sur deux piliers :

- d'une part, l'expérimentation directe sur le référent, comme évoqué dans l'Espace des applications, et d'autre part
- la théorie qui, lorsqu'elle est établie, permet une résolution analytique par des équations (propriétés et règles caractérisant la structure du système), et dont l'émergence nécessite l'établissement de modèles formels, sur lesquels des méthodes permettant de produire les résultats du comportement du système sont définies sous des hypothèses à préciser.

La simulation informatique est l'alternative moderne, celle qui est présentée comme le troisième pilier de l'activité scientifique. Cette pratique, labellisée M&S (pour Modélisation & Simulation), est adoptée lorsque les méthodes analytiques sont inapplicables (hypothèses théoriques non vérifiées) et que les expériences directes sur le réel s'avèrent non souhaitables (pour les raisons évoquées précédemment), ou encore lorsqu'une confrontation de résultats obtenus par la théorie ou l'expérience directe est souhaitée.

Deux phases sont alors prépondérantes :

- la *modélisation* (prolongement de l'effort nécessaire dans l'établissement d'une théorie) et
- la *simulation* (activité spécifique à l'Espace des calculs).

La ***modélisation*** est la construction d'une représentation du système, soit à partir de la connaissance déjà acquise de la structure de ce système, soit à partir d'une structure supposée pouvoir générer le comportement attendu. Ce modèle est une simplification du référent ciblé, sous des *hypothèses* et des *contraintes* précises (les dernières fixent les conditions sous lesquelles le modèle doit être mis en œuvre, tandis que les premières définissent les suppositions de simplification du réel). Sa spécification doit être sujette à ***vérification***, pour s'assurer que la représentation décrit bien le système référent et pas un autre, et ce dans le contexte de l'étude (hypothèses et contraintes) et pas dans un autre.

1.1.6. Espace des calculs

La **simulation** est la génération, à partir de la structure du modèle (expression), de son comportement (trajectoires), selon des plans d'expérience. La connaissance de ce comportement est « censée » nous renseigner sur celle équivalente du système, dans la mesure où ce comportement de modèle est nécessairement une interprétation de celui du système. La cohérence de cette interprétation (i.e., le fait que le modèle puisse se substituer au système dans la prise de décision) doit être sujette à ***validation***, i.e., évaluée comme légitime ou non. Cette opération peut parfois nécessiter de procéder à des ajustements des paramètres du modèle pour que les comportements respectifs du modèle et du système soient alignés (on parle alors de calibrage de modèle).

Par ailleurs, il est possible que l'inférence s'opère dans l'Espace des calculs, i.e. que des trajectoires (interprétées à partir du comportement du système réel) servent de critères de ***sélection*** d'un modèle dans une librairie de modèles selon des conditions et des méthodes à définir.

De manière assez naturelle, la transition d'un nœud modèle (expression ou trajectoires) vers l'autre peut s'interpréter comme une traversée de ***niveaux de spécification***. Les niveaux les plus élevés sont proches de l'***expression*** et les moins élevés des ***trajectoires***. La hiérarchie de niveaux de spécification n'est autre qu'un moyen de modéliser la hiérarchie de connaissances.

1.2. Problématiques d'ingénierie des modèles

L'activité de modélisation pose la problématique générale de **spécification**. En effet, il s'agit tout d'abord d'abstraction, i.e. de l'identification des réalités étudiées et des entités porteuses des connaissances établies ou supposées sur ces réalités. Comme elles prennent des formes variées, à la difficulté de leur identification s'ajoute celle de leur spécification. La modélisation est une activité très ancienne, très certainement antérieure à la simulation, et en tout cas beaucoup plus générale que celle envisagée en simulation. Ce que la modélisation a de particulier dans le cas de la M&S, c'est qu'elle est faite à but de simulation, ce qui impacte la démarche et limite l'espace des formalismes adéquats à la spécification. De surcroît, l'hétérogénéité des abstractions entraîne celle de la spécification (en termes de formalismes), la rendant ainsi plus difficile à appréhender sur le plan de la simulation. L'intégration de modèles hétérogènes est l'expression d'une double nécessité : (1) celle de combiner les modèles de simulation avec des modèles issus d'autres méthodes pour la résolution de problèmes complexes (aspect abordé en section 2.1), et (2) celle de décrire les modèles de simulation eux-mêmes à divers niveaux d'abstraction et donc au moyen de formalismes divers (aspect abordé en section 2.2). En section 2.3, nous nous intéressons à une forme originale d'intégration, celle qui consiste à construire des modèles, qui ne sont plus destinés à la simulation uniquement, mais également à d'autres types d'analyse (tels, l'analyse formelle et le prototypage rapide). Ce sont des modèles que nous qualifions de multi-analyse.

L'activité de simulation pose, elle, la problématique générale de ***synthèse*** (de code de simulation). Il s'agit de production d'automates capables de générer des traces à partir de spécifications de modèles et ce en partant de situations initiales, et pour des intervalles temporels contraints par des conditions finales spécifiques. Ces automates sont synthétisés conformément à des algorithmes, dont la mise au point tient à la question globale de synchronisation temporelle inhérente à toute simulation. Cette synchronisation devient plus ardue lorsque ces automates doivent s'exécuter sur des partitions de ressources de calcul, situation que la prédominance des architectures distribuées rend inévitable. En effet, l'architecture de ces partitions n'est pas sans influence sur l'interopérabilité des algorithmes répartis. Les difficultés de synthèse ne sont pas qu'algorithmiques, mais également technologiques. Au niveau algorithmique, le besoin scientifique est la maîtrise des stratégies de simulation et la connaissance de leurs performances (aspect abordé en section 3.1). Au niveau technologique se pose un besoin récurrent de génie logiciel : celui de la mise au point de connecteurs d'intégration de composants hétérogènes (aspect abordé en section 3.2). En section 3.3, nous nous penchons sur l'Ingénierie Dirigée par les Modèles au service du développement de cadriciel (ou framework) de synthèse assistée (à défaut d'être entièrement automatisée) de code de simulation.

A l'intersection de ces deux activités (modélisation et simulation), se trouve la problématique générale de la ***qualité***, i.e., cohérence et validité pour la modélisation, justesse et performance pour la simulation (c'est l'objet de la section 4.1). L'analyse formelle de propriétés (aussi bien de la structure des modèles conçus que des traces générées) est une approche à cette problématique. C'est cette direction que suggère la section 4.2.

La Figure 3 offre une vue synoptique des préoccupations de l'ouvrage.

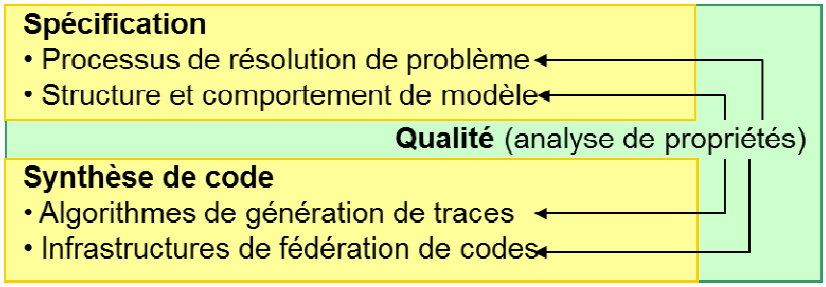


Figure 3. Problématiques de M&S

1.3. Séparation des préoccupations

Nous adhérons à l'idée que le principe de séparation des préoccupations a été le moteur des avancées majeures en M&S, et surtout, que c'est par sa généralisation que les frontières actuelles de complexité seront franchies, à l'instar de l'essor qu'ont connu les bases de données pour passer des données fragmentées dans des fichiers disparates et gérées par un programme monolithique aux processus métiers [Godart & Perrin 2009].

La séparation des préoccupations, connue également sous le nom de principe de *Dijkstra*, prône que les détails d'implémentation ne doivent pas encombrer l'entendement conceptuel et la représentation initiale d'un modèle. Elle trouve son origine historique dans les considérations d'ordre méthodologique mises en avant suite aux recherches entamées sur les langages de simulation (notamment, SIMSCRIPT, GPSS, GASP et SIMULA), inspirées des travaux précurseurs sur la notion de *world views* [Lackner 1964], [Kiviat 1967]. La compétition entre constructeurs de langage, dans les années 1970, a stimulé la nécessité de se rapprocher d'avantage des aspects conceptuels. Cette effervescence s'est d'abord traduite par l'ambition d'inclure au niveau des langages existants, des capacités d'expression de plus en plus étendues, avant de se traduire par une dichotomie entre les notions de *modèle* (ou *modèle abstrait*) et de *simulateur* (ou *modèle concret*).

Des contributions majeures à l'édification du principe de raffinements successifs (à partir d'un niveau d'abstraction élevé jusqu'à la production du programme de simulation) comme fondement méthodologique, ont été apportées par les travaux entrepris de manière parallèle, d'une part par les Norvégiens du groupe SIMULA dans le cadre du projet DELTA [Nygaard & Dahl 1978], et d'autre part à travers la méthode conique (*Conical Methodology* ou CM) [Nance 1981], [Overstreet & Nance 1986], [Nance 1987]. Cette dernière tire son nom du fait que le raffinement progressif d'une spécification de modèle est vu comme un cône dont la base (i.e., le niveau le plus détaillé) serait le programme exécutable. Le modèle global est la somme de toutes les spécifications successives et non uniquement la dernière. La démarche méthodologique proposée alterne des phases de description incrémentale descendantes et ascendantes, de manière à produire des représentations successives du modèle, à différents niveaux d'abstraction, qui peuvent être vérifiées et validées avant la phase

d'implémentation conduisant au programme exécutable. Un formalisme appelé CS (*Condition Specification*) permet de produire ces spécifications de manière uniforme [Overstreet et al.1994].

La dichotomie modèle/simulateur a pour avantage d'améliorer le processus de vérification, certaines erreurs de spécification pouvant alors être mises à jour au niveau conceptuel plutôt qu'au niveau d'implémentation où elles sont plus difficiles à détecter. En outre, différentes implémentations de simulateur peuvent être proposées pour une même spécification de modèle.

L'idée poursuivie ici est que ce principe correctement réitéré défait progressivement, au fil de la révélation de couches de préoccupation supplémentaires, l'entremêlement des concepts, réduisant ainsi la complexité globale et fragmentant alors les verrous que cette dernière engendre. Aussi, nous préconisons une démarche de structuration décrite en Figure 4 et qui distingue :

- Les processus de M&S, ensemble d'activités dont la logique est indépendante des modèles mis en œuvre et des solutions techniques mobilisées. Ce niveau se concentre sur les objectifs de l'étude entreprise, la manière dont les contraintes sont respectées et la démarche par laquelle le problème posé doit être résolu. Les processus de M&S décrivent des modèles de résolution.
- Les modèles des systèmes cibles, i.e. les descriptions de la structure et des règles comportementales des référents auxquels les études de M&S s'intéressent. Ce sont ces modèles cibles qui sont sollicités par les processus de résolution de la couche supérieure (un processus pouvant solliciter plusieurs modèles différents, et un modèle pouvant être sollicité par plusieurs processus différents).
- Les automates de simulation, composants logiciels qui réalisent les algorithmes de simulation. Un automate peut générer les traces de plusieurs modèles cibles différents et un modèle cible peut utiliser différents automates de génération de trace.
- Les connecteurs d'interopérabilité, infrastructures mettant en œuvre des protocoles de fédération d'automates dispersés sur des ressources de calcul hétérogènes. Un automate peut s'exécuter sur différents connecteurs, et un connecteur peut accepter différents automates.

Cette séparation permet de bien distinguer les divers niveaux d'hétérogénéité, ainsi que les problématiques qu'ils induisent, et qui constituent des verrous à la l'ingénierie des modèles de simulation :

- unification de méthodes de résolution hétérogènes (niveau Processus) ;
- intégration de modèles de simulation hétérogènes (niveau Cible) ;
- agrégation de ressources de calcul hétérogènes (niveau Automate) ;
- fédération des codes de simulation hétérogènes (niveau Connecteur).

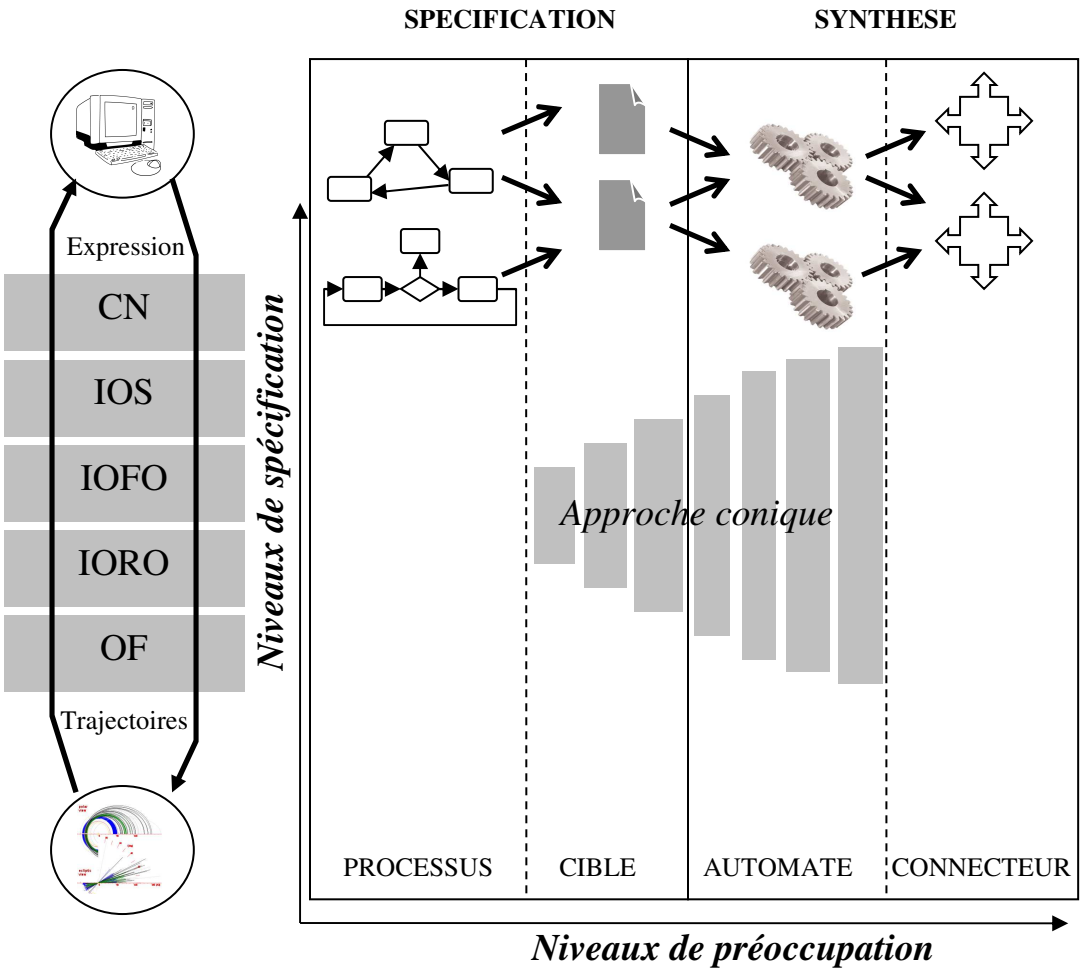


Figure 4. Séparation des préoccupations (2nde partie du cycle de vie)

Les problématiques de spécification évoquées précédemment couvrent les niveaux Processus et Cible, alors que les problématiques de synthèse couvrent, elles, les niveaux Automate et Connecteur. Les problématiques de qualité concernent l'ensemble de ces niveaux.

D'autre part, les niveaux de préoccupation ainsi dégagés sont orthogonaux aux niveaux de spécification qui traduisent la hiérarchie de connaissance dont le

sommet est la structure d'un système et la base en est le comportement. Pour bien comprendre cela, il faut comprendre que chaque niveau de préoccupation s'intéresse à un système dans son ensemble (on y retrouve donc les concepts de structure et de comportement), mais qu'il l'exprime de manière différente (par des concepts ou du code).

La Figure 4 montre comment l'approche conique se situe par rapport à cette orthogonalité. Elle montre également la hiérarchie à cinq niveaux que propose le paradigme DEVS [Zeigler 1976] pour formaliser les niveaux de spécification :

Niveau 0. Le niveau OF (*Observation Frame*) décrit le cadre d'observation du système. Il s'écrit <X, Y, T> où X désigne l'ensemble des entrées que le système peut recevoir (information observée), Y l'ensemble des sorties qu'il peut générer (information mesurée), et T la base de temps (en général l'ensemble des nombres réels positifs).

Niveau 1. Le niveau IORO (*Input Output Relation Observation*) décrit les relations observées entre les entrées et les sorties du système. Il s'écrit <X, Y, T, Ω, R> où X, Y et T sont conformes aux définitions du niveau 0, Ω désigne l'ensemble des segments d'entrée admissibles par le système (un segment étant une valeur constatée sur un intervalle de temps donné), et R désigne l'ensemble des couples (segment d'entrée, segment de sortie) observées.

Niveau 2. Le niveau IOFO (*Input Output Function Observation*) décrit les liens fonctionnels observés entre les entrées et les sorties du système. Il s'écrit <X, Y, T, Ω, F> où X, Y, T et Ω sont conformes aux définitions du niveau 1, et F désigne la fonction (ou l'ensemble de fonctions) exprimant les segments de sortie en fonction des segments d'entrée.

Niveau 3. Le niveau IOS (*Input Output System*) décrit les dépendances entre les entrées et les sorties du système sous la forme d'une machine à états. Il s'écrit <X, Y, S, δ_{int}, δ_{ext}, δ_{conf}, λ, ta> où X et Y sont conformes aux définitions du niveau 2. S désigne l'ensemble des états que peut prendre le système. Les transitions d'état sont dictées par une fonction de transition externe δ_{ext} (activée lorsqu'une entrée est reçue par le système, et qui fait passer le système dans un nouvel état), une fonction

d'avancement du temps ta (définissant la durée de vie maximale de chaque état), et une fonction de transition interne δ_{int} (activée lorsque la durée de vie maximale d'un état est atteint, et qui fait passer le système dans un nouvel état). En cas de simultanéité de réception d'entrée et d'expiration de durée de vie maximale de l'état courant, la fonction de transition confluente δ_{conf} (définie comme arbitrage entre les deux autres fonctions de transition) est activée.

Niveau 4. Le niveau CN (*Coupled Network*) décrit le système comme décomposable en un réseau de sous-systèmes interconnectés. Il s'écrit <X, Y, D, $\{M_d\}_{d \in D}$, EIC, EOC, IC> où X et Y sont conformes aux définitions du niveau 3. D désigne l'ensemble des noms (ou références) sous-systèmes, et chaque M_d décrit un des sous-systèmes au niveau 3 ou au niveau 4. Les interconnections entre sous-systèmes sont décrites par IC (*Internal Coupling*) sous forme d'une matrice de couplage dont chaque couplage indique comment la sortie d'un sous-système est connectée à l'entrée d'un autre sous-système. EIC et EOC sont aussi des matrices de couplage, mais EIC (*External Input Coupling*) décrit comment les entrées du système global sont transférés vers les entrées de ses sous-systèmes, tandis que EOC (*External Output Coupling*) décrit comment les soryies des sous-systèmes sont transférés vers les sorties du système global.

Bien qu'utilisée, le plus souvent, pour spécifier uniquement les modèles au niveau de préoccupation Cible, cette hiérarchie s'applique à chacun des autres niveaux de préoccupation. En pratique, ce sont les niveaux de spécification IOS et CN qui sont définis de manière conceptuelle (i.e., au niveau de préoccupation Cible) et implémentés (i.e., au niveau de préoccupation Automate). Dans l'absolu, il est par exemple possible de réaliser la séquence suivante :

1. concevoir une solution de résolution simulation-centrée (i.e., au niveau de préoccupation Processus) au niveau de spécification IOFO (par exemple, sous forme d'équations différentielles) ou au niveau de spécification IOS (par exemple, sous forme de Processus Métiers) ;

2. proposer une description des systèmes cibles impliqués (i.e., au niveau de préoccupation Cible) en se plaçant au niveau de spécification CN (par exemple, sous forme de modèles DEVS couplés) ;
3. implémenter les codes correspondants (niveau de préoccupation Automate) au niveau de spécification IOS (par exemple des simulateurs DEVS distants) ; et
4. relier ces codes par un composant (niveau de préoccupation Connecteur) conçu au niveau de spécification IOS.

Les chapitres 2 et 3 de cet ouvrage suivent la philosophie de cette séparation généralisée des préoccupations : spécification en chapitre 2, avec distinction entre processus de résolution et modèles de système, synthèse en chapitre 3, avec distinction entre automates de simulation et connecteurs d'interfaçage distant. Le chapitre 4 s'attache aux qualités théoriques et opérationnelles de ces différentes éléments.

Chapitre 2.
Spécification

En fait, l'étude par simulation des systèmes complexes recèle deux sortes de complexité : celle liée à la compréhension et à la maîtrise des règles comportementales internes de ces systèmes, et celle due à la nature même de l'étude, c'est-à-dire la difficulté à résoudre la question posée à l'origine de l'étude (problèmes NP difficiles, NP complets, NP durs). Cette double complexité conduit le plus souvent à fusionner des concepts et des processus méthodologiques d'inspirations pluridisciplinaires (i.e., des emprunts à l'Intelligence artificielle, à la Recherche opérationnelle…) dans un cadre commun.

Avec la séparation généralisée des préoccupations, nous évitons que ces deux niveaux de complexité s'intriquent au point de ne pas permettre leur traitement distinct. Au niveau de préoccupation Processus, nous mettons l'accent sur la spécification du modèle de résolution du problème posé. Au niveau de préoccupation Cible, il s'agit de spécifier le modèle du (ou des) système(s) impliqué(s) dans le problème posé.

2.1. Modèle de résolution

Un modèle de résolution simulation-centré décrit par essence une construction qui plonge un ou plusieurs modèles de simulation dans un dispositif de prise de décision (l'homme ou/et un dispositif informatisé de pilotage).

Les preneurs de décision peuvent interagir avec ce système selon trois axes :

(1) proposer des plans d'expériences sur lesquels le modèle de simulation est mobilisé, et réutiliser les résultats de simulation pour résoudre le problème posé (et d'autres problèmes de la même classe) ; dans

certains cas, en l'occurrence lorsque les phénomènes décrits incluent un aspect stochastique, chaque expérience est réitéré plusieurs fois, et l'analyse des résultats s'accompagne d'une quantification du degré de confiance qu'il leur est accordé (calcul d'intervalles de confiance) ;

(2) extraire, à des fins d'analyse autre que la simulation (e.g., analyse formelle), des connaissances à partir de l'information statique qu'est le modèle conceptuel ; ce dernier est donc une base de connaissance partagée entre modeleurs, concepteurs de systèmes physiques, experts du domaine et preneurs de décision ;

(3) envoyer des commandes au système physique (si ce dernier existe) ou de conception (s'il est à optimiser ou à concevoir) ; dans le premier cas, il s'agit d'un système de pilotage, qui doit de manière répétitive, récupérer l'état du système physique, initier au besoin une campagne de simulation, en analyser les résultats, et produire de nouvelles commandes pour le système physique ; dans le second cas, les commandes sont remplacées par des plans de conception et/ou de configuration. Par ailleurs, le système physique peut être remplacé par (ou augmenté de) l'humain, le pilotage pouvant conduire alors à de l'entraînement (interaction Homme-Machine, ou Humain dans la boucle). En inversant les rôles, la situation se ramène à de l'apprentissage (« *Machine Learning* »).

En pratique, les modèles de résolution par simulation sont peu formalisés. La formalisation est le plus souvent réservée au(x) modèle(s) du (des) systèmes mobilisé(s), mais très rarement à l'enchaînement des activités qui caractérise l'usage de ce(s) modèle(s) pour l'atteinte des objectifs de l'étude (sous les contraintes imposées). Pourtant, cette expression formelle, en plus de rendre non ambiguë la description du schéma de résolution simulation centrée, ouvrirait la voie à la manipulation symbolique de cette description (donc à la possibilité d'évaluer la qualité du processus construit) et à l'automatisation des mécanismes de synthèse de code à partir de cette description. Par ailleurs, les formalismes ne manquent pas pour capturer cette description, même si le choix du formalisme le plus approprié se pose (nous retrouverons cette question plus loin, avec la spécification du modèle du système).

Dans certains cas, les formalismes de description de processus (tels, diagramme d'activité, BPMN, organigramme…) sont utilisés pour décrire l'algorithme de résolution du problème, mais très souvent, sans y incorporer la description du (ou des) système(s) impliqué(s).

En réalité, la diversité des problèmes et des catégories auxquels ils appartiennent (Intelligence Artificielle, Recherche Opérationnelle…) se traduit par une diversité d'approches de résolution (algorithmes gloutons, méthodes linéaires et non linéaires, algorithmes génétiques, heuristiques...) qui proviennent d'expertises dans des champs disciplinaires distincts (et parfois assez éloignés) de la M&S, cette dernière offrant une expertise dans la spécification des systèmes dynamiques en vue de leur analyse par simulation.

Comment donc prendre en compte la diversité des problèmes, systèmes, et démarches de résolution dans un cadre unificateur, tout en préservant la nécessaire adaptabilité de cette dernière à des contextes spécifiques ? La littérature révèle deux sortes d'approches de définition de modèle de résolution par unification de méthodes hétérogènes :

- Les couplages simulation-centrés : il s'agit de schémas génériques de couplage fonctionnel entre la simulation et des outils issus, le plus souvent de la Recherche Opérationnelle [Pratt et al.1994], [Caux et al. 1995], [Artiba & Elmaghraby 1996], [McHaney 1999], ou de l'Intelligence Artificielle [O'Keefe 1986], [Ören 1989], [Merkuryeva & Merkuryev 1994].
- Les tissages simulation-centrés : dans ce cas, la simulation n'est plus simplement une boîte noire utilisée dans un schéma de résolution, mais fusionne avec cette connaissance dans une description homogène. Des exemples sont [Ferber & Drogoul 1992] pour la fusion entre simulation et approche multi agents, [Wild & Pignatiello 1994] pour la fusion entre simulation et backtracking, et [Turner & Baker 2008] pour l'analyse structurelle de fiabilité des systèmes dynamiques par simulation.

2.1.1. Couplage simulation-centré de méthodes

Les problèmes très complexes trouvent leur résolution dans le couplage de la simulation avec des méthodes résolutoires, que bien souvent la littérature expose

sous forme de connexion de boîtes, les unes représentant ces méthodes résolutoires et les autres les modèles de simulation. Ce schéma, presque simpliste, recèle de vraies difficultés dans sa mise en œuvre, en particulier dans la manière dont s'interfacent réellement ces boîtes. Pour bien comprendre cette difficulté et savoir comment l'aborder, il importe de comprendre d'abord les deux échelles de temps que cache le couplage simulation-centré : celui de la simulation et celui du couplage. La figure 5 décrit cette dualité et met en lumière les aspects qui relèvent du modèle et ceux qui relèvent de son cadre d'utilisation :

- Le long de l'axe temporel τ se déroule le processus de résolution simulation-centré. Ce temps est celui de l'horloge murale. Au cours de ce processus, un modèle est choisi, instance d'une classe. Dans le cas le plus général, la classe est un modèle paramétré et l'instance est le modèle issu du choix de valeurs spécifiques pour ces paramètres.
- Pour chaque instanciation de modèle, une série de simulations (réplications) est conduite. Chaque simulation nécessite une initialisation de l'état du modèle et la définition des conditions d'arrêt. La simulation s'exécute alors pour un temps simulé donné (T1, T2, T3…), qui a une durée dans le temps mural (i.e., le temps d'exécution de la simulation).
- A la fin d'une série, une nouvelle instance est produite, à partir de nouveaux paramètres et une séquence similaire d'expériences est menée.
- Entre chaque simulation, mais aussi entre deux instanciations, différents types de calcul peuvent être effectués : sauvegarde de résultats de simulation, cumul, traitements spécifiques… Les paramètres d'instanciation sont fournis par les méthodes résolutoires avec lesquelles le modèle de simulation est couplé.

Appelons M le modèle générique de simulation, paramétré par le vecteur $\alpha^{(n)}$ de dimension n. Aussi, $M(\alpha_i^{(n)})$ est une instance de ce modèle avec les valeurs des paramètres fixés à $\alpha_i^{(n)}$. Appelons EF le modèle générique représentant le dispositif nécessaire à l'expérimentation sur un modèle de simulation (cadre expérimental, concept introduit par [Zeigler, 1976]). Pour les mêmes raisons, EF peut être paramétré par un vecteur de dimension m-n : $\beta^{(m-n)}$ et $EF(\beta_j^{(m-n)})$ désignera une instance particulière de ce cadre d'expérimentation.

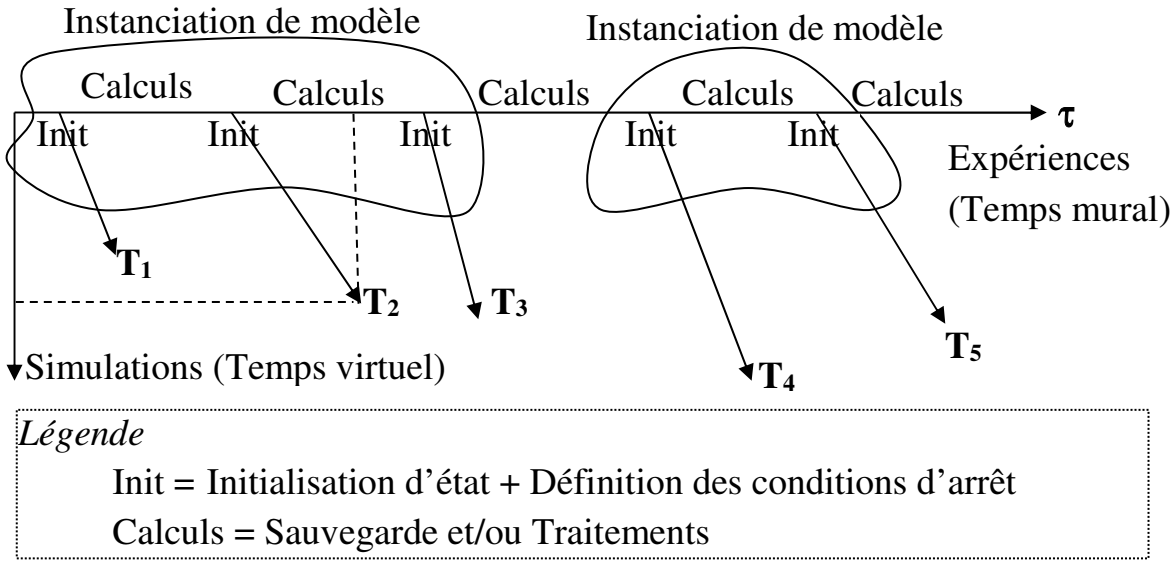


Figure 5. Mise en œuvre de couplage simulation-centré

Dans un couplage simulation-centré, comme le montre la Figure 6, l'ensemble des expérimentations possibles sur le modèle de simulation est pris comme une fonction F du domaine des paramètres possibles à celui des résultats, fonction qu'invoquent les méthodes résolutoires G dans leurs algorithmes et heuristiques. En d'autres termes, il s'agit de construire une suite $\{\gamma_i^{(m)}\}$ telle que :

$$\gamma_{i+1}^{(m)} = G(\gamma_i^{(m)}, F(\gamma_i^{(m)})), \text{ avec}$$

- $F(\gamma_i^{(m)}) = \pi_i^{(s)}$, où F désigne la fonction d'évaluation (réalisée par les simulations du modèle) et $\pi^{(s)}$ le vecteur de dimension s des résultats calculés à la suite des simulations.
- $\gamma_{i+1}^{(m)} = G(\gamma_i^{(m)}, \pi_i^{(s)})$, où G désigne la fonction de guidage, qui détermine les nouvelles valeurs des paramètres (à soumettre à de nouvelles expérimentations) en fonction des résultats des expérimentations précédentes.

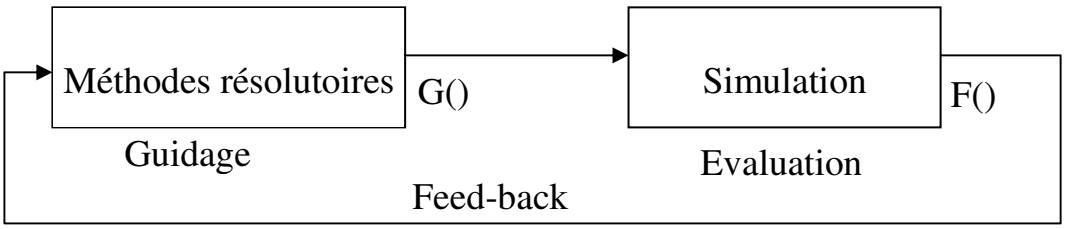


Figure 6. Couplage simulation-centré

La construction des fonctions F et G recèle des difficultés à la fois d'ordre théorique (synthèse des algorithmes de guidage), méthodologiques (réalisation

de F) et pratiques (implémentation logicielle de l'architecture de résolution simulation-centrée).

La fonction F est réalisée par le couplage entre M et EF, ce que nous écrivons :

$$F = M \oplus EF, \text{ avec}$$

- ✓ $F(\gamma_i^{(m)}) = M(\alpha_i^{(n)}) \oplus EF(\beta_i^{(m-n)})$, i.e., l'évaluation du vecteur de paramètres de dimension m, entraîne l'instanciation du modèle avec le sous vecteur de dimension n issu de ce vecteur initial, l'instanciation du dispositif d'expérimentation avec le sous vecteur de dimension m-n issu du même vecteur initial, et leur couplage dont l'exécution fournit le résultat de l'évaluation.
- ✓ $\gamma_i^{(m)} = \alpha_i^{(n)} \bullet \beta_i^{(m-n)}$, où $\bullet$ est l'opérateur de concaténation.

La fonction G est réalisée par les méthodes résolutoires mises en œuvre. Cette réalisation dépend du domaine d'étude et des bases théoriques sous-jacentes.

Les deux études qui suivent, et qui proviennent de situations réelles, sont des illustrations de couplage simulation-centré mettant en œuvre le principe décrit, mais mettant aussi en exergue le défaut de formalisation complète dès lors que des méthodes hétérogènes sont en jeu.

2.1.1.i. Ordonnancement dynamique simulation-centré

Cette étude concerne la mise au point d'une approche réactive d'optimisation et d'aide au pilotage de système de production par couplage simulation-centré. Bien que de nombreux résultats probants aient été établis en ordonnancement [Blazewicz et al. 1994], [Carlier & Chretienne 1988], [Mac Carthy & Liu 1993], les méthodes classiques résistent mal au passage à l'échelle industrielle en raison de la difficulté d'intégrer les contraintes temps réel inhérentes à la très grande majorité des systèmes de production. Le degré d'intégration de ces contraintes dépend du type de pilotage adopté :

- En pilotage périodique, des décisions sont périodiquement déterminées hors ligne avant mise en application en ligne (faible degré de réactivité).
- En pilotage réactif sans prévision, les décisions sont progressivement construites en ligne sans aucune visibilité sur le futur du système (fort

degré de réactivité).

- En pilotage réactif avec prévision, les décisions sont construites en ligne mais avec une visibilité sur le futur du système pour une certaine fenêtre de temps (degré médian de réactivité).

L'étude porte ici sur l'ordonnancement dynamique d'un Flowshop hybride à 5 étages (Figure 7). Ce dernier, dans lequel des produits subissent une série de traitements les faisant passer de l'état de matière première à celui de produit manufacturé, est constitué de plusieurs étages, chacun représentant un pool de machines parallèles spécialisées dans un type d'opération (les taux de production, pour une même opération, peuvent varier en raison de la variété des machines : certaines sont automatiques, d'autres manuelles, d'autres semi-automatiques). Une machine ne peut opérer que sur une variété donnée de produit à un moment donné ; pour qu'elle puisse traiter une autre variété, il est nécessaire qu'elle fasse un changement d'outil (ces changements ont un coût économique, humain et technique ; ils doivent donc être optimisés au possible). Les étages sont séparés par des stocks et l'approvisionnement de l'atelier en matières premières est toujours assuré.

Dans le cas étudié, il s'agit d'ateliers à grandes séries. Les caractéristiques de tels systèmes sont que peu de types de produits sont fabriqués mais chaque type de produit fait l'objet d'une très large production. Sur la Figure 7, il y a un problème d'ordonnancement en entrée sur la Machine *Ma* et un problème d'allocation de ressources sur les pools de machines réalisant respectivement les opérations *op2* et *op3*. Des automates assurent le transport des pièces et les machines sont approvisionnées en flux tiré. Ainsi, le problème d'ordonnancement de l'atelier repose entièrement sur le bon ordonnancement sur la Machine Ma et la bonne allocation de ressources sur les pools de machines parallèles. Un mauvais ordonnancement produit une famine sur certaines machines ou certains stocks et une surcharge sur d'autres. Ce problème est un flowshop hybride selon la classification de Conway [Conway et al. 1967], et spécifié de la manière suivante selon la notation étendue de [Rinnooykan 1976] : HF5 / ($\varnothing^{(1)}$, $Qk^{(2)}$, $Qn^{(3)}$, $Qk^{(4)}$, $Qm^{(5)}$) / … / Ec_{min}, C_{tools}, où

- HF5 : flowshop hybride à 5 étages.
- $\varnothing^{(1)}$: une seule machine compose le 1er étage.
- $Qk^{(2)}$: k machines parallèles composent le 2ème étage (ceci modélise le

stock Sa avec k variétés de produits).

- $Qn^{(3)}$: n machines parallèles composent le 3ème étage.
- $Qk^{(4)}$: k machines parallèles composent le 42ème étage (stock Sb).
- $Qm^{(5)}$) : m machines parallèles composent le 5ème étage.
- Ec_{min} : somme des écarts entre la production et le plan de production (sous-productions et sur-productions).
- C_{tools} : nombre de changements d'outil.
- Cette notation ne permet pas de spécifier les contraintes inter étages, ni celles liées aux changements d'outil (d'où les points de suspension).

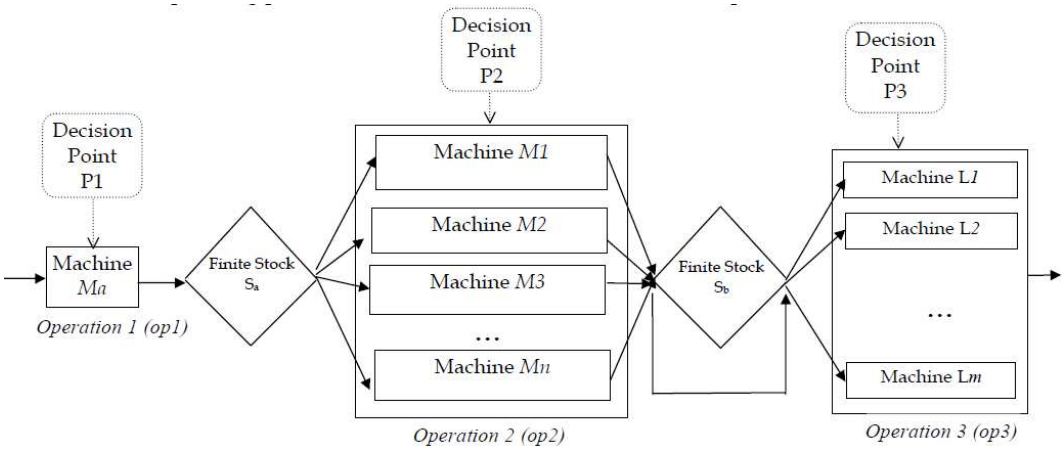


Figure 7. Flowshop hybride à 5 étages

Les méthodes mathématiques mobilisées pour traiter les problèmes d'optimisation dans un Flowshop hybride sont traditionnellement des techniques de programmation linéaire et non linéaire. Elles sont confrontées à l'explosion de la taille des variables et des contraintes (par exemple, 4 variétés de produit et 48 périodes de production conduisent à 3125 variables et 3918 contraintes). La simulation a beaucoup été utilisée pour le Flowshop hybride à 2 étages, mais peu pour un nombre plus élevé d'étages [Hunsucker & Shah 1992]. Par ailleurs, ces travaux ne prennent pas en compte les contraintes temps réel. Pourtant, la grande difficulté pour piloter la production dans ces systèmes provient de la nécessité de réactivité aux aléas. C'est pourquoi le plan de production doit être construit de manière progressive, chaque plan étant remis en cause par l'apparition d'un nouvel événement imprévu (ordonnancement opérationnel [Van Looveren et al. 1986]). C'est un cas particulier d'allocation de ressources multi périodes et multi produits [Proust & Grünenberger 1994].

Ici, la fonction objectif est la suivante :

Minimiser { coût de changement d'outil +
coût de sous-production par rapport au programme prévu +
coût de surproduction par rapport au programme prévu}.

Différentes techniques peuvent être utilisées pour spécifier ce type d'optimisation multi objectifs [Gravel et al. 1992] : sommes pondérées, dominance stochastique… Dans le cas présent, les critères (objectifs) sont hétérogènes (le critère de changement d'outil est de nature différente des deux autres) et hiérarchiques. Par ailleurs, certaines techniques de modélisation issues de l'Intelligence Artificielle (systèmes experts, Multi-Agents…) ont fait leur preuve dans la résolution de problèmes de pilotage temps réel [Kusiak & Chen 1988]. L'idée récurrente derrière ces techniques est de spécifier des points de décision et de les intégrer dans le système d'information des ateliers de production ciblés, afin de rendre la prise de décision réactive. Cette idée est mise en œuvre ici dans un couplage qui interface un module de simulation avec un module d'optimisation (ce dernier étant composé d'heuristiques et d'algorithmes stochastiques). L'architecture fonctionnelle de la solution mise en place (appelée « approche réactive ») est donnée en Figure 8. Il s'agit d'un couplage simulation-centré, intégré au système d'information (SI) et interagissant de manière dynamique avec le système physique, les preneurs de décision (ceux qui définissent les objectifs à atteindre) et les opérateurs humains (ceux qui sont en charge du plan de production journalière). La solution d'ordonnancement est construite de manière dynamique par boucle de rétroaction entre la simulation et l'algorithme du module décisionnel :

- La simulation met en œuvre un modèle (structure de l'atelier et règles de production) et un dispositif d'expérimentation, ce dernier étant paramétré par le plan initial de production (qui lui sert à calculer les écarts de production), le générateur d'événements imprévus (suite d'événements datés à envoyer au modèle) et le planning de la période (générateur de produits vers le modèle). Ces éléments définissent donc les composantes du vecteur $\gamma^{(m)}$. L'évaluation (la fonction F) fournit le vecteur d'état de l'atelier et les écarts de production constatés.
- Le module décisionnel (fonction G) est composé de trois algorithmes multi objectifs : GAAC pour *Greedy Algorithm of Autonomy Calculation*, GAAS pour *Greedy Algorithm of Stock Saturation* et, HSD pour

Hierarchical Stochastic Descent) [Belkhiter et al. 1997].

Les principes de ces algorithmes sont les suivants :

- L'algorithme GAAC se focalise sur la satisfaction du plan de production, en utilisant la notion d'autonomie d'une machine, calculé par $C_{ij} = (S_{ij}+P_{ij})/R_{ij}$, où S_{ij} est la quantité de produits de variété j disponibles pour la machine i au début de la période de production, P_{ij} est la quantité de produits de variété j produites par la machine i pendant cette période, et R_{ij} est la quantité de produits de variété j consommés par les machines en aval de la machine i pendant cette période. L'algorithme privilégie les machines très autonomes pour éviter la sous-production et les machines peu autonomes pour éviter la surproduction.
- L'algorithme GAAS se focalise sur la réduction du nombre de changements d'outil, en utilisant la notion de saturation de stock, calculé par $K_{ij} = C_{ij}/(S_{ij}+P_{ij}-R_{ij})$, où C_{ij} est la capacité du stock i en produits de variété j, et S_{ij}, P_{ij} et R_{ij} sont définis comme précédemment. Les stocks à forte saturation sont privilégiés pour réduire les changements d'outils.
- L'algorithme HSD est une adaptation de la méthode classique de descente [Cea 1968], prenant en compte les critères multiples et les hiérarchisant : d'abord la satisfaction du plan de production (minimisation de la somme des écarts de production), puis la réduction du nombre de changements d'outil.

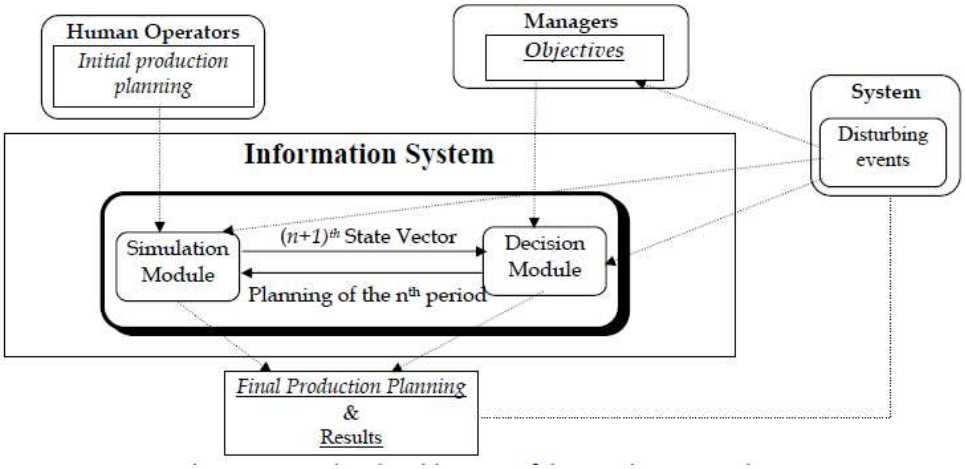


Figure 8. Architecture fonctionnelle de l'approche réactive

L'étude comparative des trois algorithmes mis au point a produit les résultats exhibés en Figure 9. Les tests ont porté sur plus de 120 plans de production (plusieurs milliers de produits par plan). Les deux critères calculés sont respectivement la variation (en %) de la production par rapport au plan de production prévu (1er graphique de la Figure 9) et le nombre de changement d'outils (2nd graphique de la Figure 9). Les résultats soulignent bien le compromis existant entre la satisfaction du plan de production (environ 25% moins bonne avec GAAS qu'avec GAAC) et la réduction du nombre de changements d'outil (10 fois plus importante avec GAAS que GAAC). L'algorithme HSD est symbolique de ce compromis. Les solutions définies durant ces travaux ont été validées par les industriels et intégrées au système d'aide au pilotage des ateliers de production concernés.

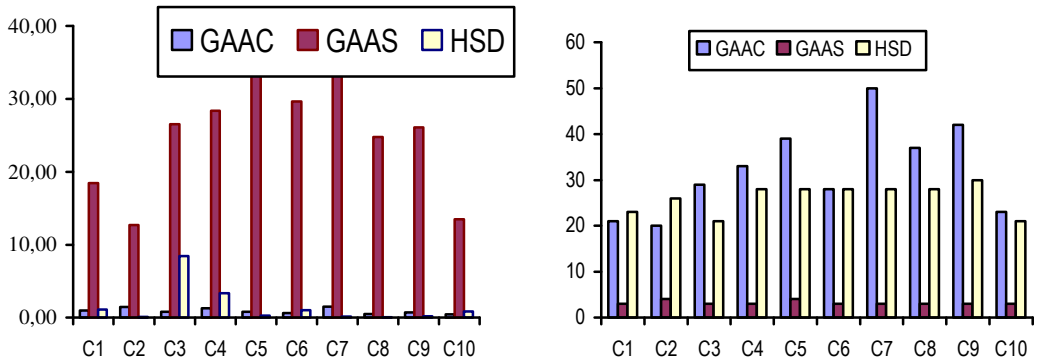


Figure 9. Résultats comparés des approches de résolution

2.1.1.ii. Couplage triple

L'étude des Ateliers de Traitement de Surfaces (ATS) constitue, par la variété de configurations et de contraintes qui s'y trouvent, un problème à part entière (dénommé HSP, pour Hoist Scheduling Problem) [Manier & Baptiste 1994]. Il existe un très grand nombre d'ATS dans l'industrie. Le principe consiste à faire subir à des produits, une succession de bains électrolytiques ou chimiques visant à réaliser des dépôts métalliques ou minéraux. Ces traitements sont réalisés dans des cuves juxtaposées selon diverses configurations (en I, U, H...). Un ou plusieurs robots de manutention, pilotés par automate, sont utilisés pour transférer les produits de cuve à cuve.

L'optimisation du fonctionnement d'un tel système peut poser des problèmes aux niveaux suivants :

- La planification des mouvements de robot, pour trouver la suite des mouvements à réaliser afin d'effectuer tous les transferts nécessaires au traitement de la charge de l'atelier. la complexité provient ici de la contrainte temporelle sur les temps de bain (bornes inférieure et supérieure). En outre, les durées de bain autorisées sont du même ordre de grandeur que les temps de déplacement des robots.
- L'ordonnancement amont, opportun lorsque les produits subissent des suites de bains différentes. Il s'agit de trouver un ordre optimal d'entrée en ligne des pièces.
- L'agencement des cuves. Le critère principal est la fréquence de visite des cuves, certaines pouvant même être utilisées plusieurs fois par un produit (cuve multifonction, par opposition à mono fonction). Parfois, une cuve peut accueillir plusieurs produits à la fois (cuve multi bac). L'agencement peut aussi dépendre de contraintes de précédence ou de proximité des bains.
- L'optimisation du nombre de supports de transport, dans le cas où ces derniers ne sortent jamais de la ligne. Deux options existent : rechercher un cycle optimal de mouvements de robot dans lequel tous les supports sont toujours pleins, ou définir une séquence de transferts de supports qui induit une gestion de supports vides.
- L'optimisation du nombre de robots, qui dépend de la gestion adoptée. Les zones d'intervention des robots peuvent se recouvrir sur plusieurs cuves (sans possibilité pour les robots de se croiser) ou sur une seule (cuve d'échange).

Dans l'étude présente, la juxtaposition des cuves est linéaire (configuration en I) et l'agencement des cuves est une contrainte fixe pré établie. Par ailleurs, toutes les cuves sont mono fonction et mono bac. Chaque support de transport est un fût contenant un lot de plusieurs dizaines de milliers de pièces identiques. Les lots se différencient par leur gamme, i.e. la séquence de bains à subir (les séquences pouvant bien sûr avoir des cuves communes mais pas dans le même ordre, ni avec les mêmes contraintes de temps de bain). Un seul robot assure la manutention et le nombre de support est fixe sur la ligne. La ligne fonctionne par cycle journalier et les campagnes de traitement (chacune définie par la liste

des gammes des lots à traiter) varient d'un jour sur l'autre, mais aussi peuvent subir une modification en cours de traitement. Deux objectifs priment donc : l'optimisation et le pilotage temps réel de cette ligne.

La littérature propose plusieurs études d'optimisation d'ateliers de traitement de surface [Phillips & Unger 1976], [Bracker & Chapman 1985], [Thesen & Lei 1986], [Shapiro & Nuttle 1988], [Lei & Wang 1989], [Thesen & Lei 1990], [Yih 1990], [Lei & Wang 1991], [Lacoste & Baptiste 1992], [Song & al. 1993], [Baptiste et al. 1994], [Chen & Chu 1994], [Hanen & Munier 1994], [Armstrong et al. 1995], [Caux et al. 1995], [Ge & Yih 1995], [Lamothe et al. 1995], [Varnier et al. 1995], [Lamothe et al. 1996], [Ng 1996], [Varnier 1996]. Les nécessaires hypothèses propices à l'application de ces méthodes ne sont pas réalisées dans le cas de la ligne étudiée. L'enjeu est de taille et concerne plusieurs millions de pièces à très forte valeur ajoutée. Il l'est encore pour plusieurs industries (aéronautique pour le traitement électrolytique des composants de moteur d'avion, industrie électronique pour les dépôts métalliques sur les circuits imprimés, bijouterie pour le plaquage en métal précieux…).

La solution proposée dans [Gourgand et al. 2003] est un couplage entre un modèle de simulation (pour constater l'effet des décisions de gestion sur le système), un modèle Multi-Agents (pour spécifier les règles de gestion décisionnelle de la production à l'intérieur de la ligne) et des algorithmes stochastiques (pour optimiser la production en entrée du système). La figure 10 présente l'architecture de ce couplage. La simulation rassemble un modèle de simulation et un dispositif d'expérimentation. Le modèle de simulation spécifie la structuration statique d'un atelier de type général, tandis que la partie décisionnelle est un agrégat d'agents autonomes évoluant de manière asynchrone, chacun situé au niveau d'une entité physique pour laquelle des décisions sont requises (Figure 11). La solution proposée se caractérise ainsi :

- Un atelier possède une ou plusieurs lignes. Les robots de manutention appartiennent à une seule ligne, une ligne pouvant avoir plusieurs robots. Les cuves appartiennent à une ligne, et chaque ligne a une desserte (zone d'échange), i.e., une cuve accessible par plusieurs robots et/ou transporteurs. Les dessertes gèrent leurs zones d'échange.
- Un ou plusieurs éléments de flux visitent les lignes. Selon le type d'atelier,

un élément de flux correspond, soit à l'ensemble produit + support de transport, soit au seul produit (dans ce cas, il n'y a pas de support de transport sur la ligne). Les éléments de flux n'appartiennent pas à une ligne mais à l'atelier, car ils passent d'une ligne à l'autre en fonction de la gamme des produits associés.

- Un ou plusieurs transporteurs relient les lignes.
- Un ou plusieurs opérateurs gèrent les chargements et déchargements.
- Chaque agent met en œuvre des règles locales, soit en agissant sur une ressource par des actionneurs, soit en communiquant avec d'autres agents.
- Un agent superviseur de plus haut niveau gère l'atelier.
- Un agent superviseur existe par ligne ; il transmet les conflits qu'il ne peut gérer à son superviseur, ce dernier disposant d'informations plus globales.
- Lorsqu'un agent ne dispose pas des informations nécessaires à une prise de décision, il avertit son agent superviseur et ce processus se poursuit jusqu'à l'agent de niveau hiérarchique suffisant. La décision prise redescend la hiérarchie par l'intermédiaire des tableaux noirs, structures de données à accès contrôlé et à caractère modulaire.
- Les agents, associés aux transporteurs et aux robots, prennent des décisions locales selon les informations lues dans les tableaux noirs.
- Un agent gère par desserte les ressources qui y sont liées.
- Un agent associé à chaque cuve évalue, à partir des informations locales, un indice représentatif de l'urgence pour l'accès à un moyen de transport.

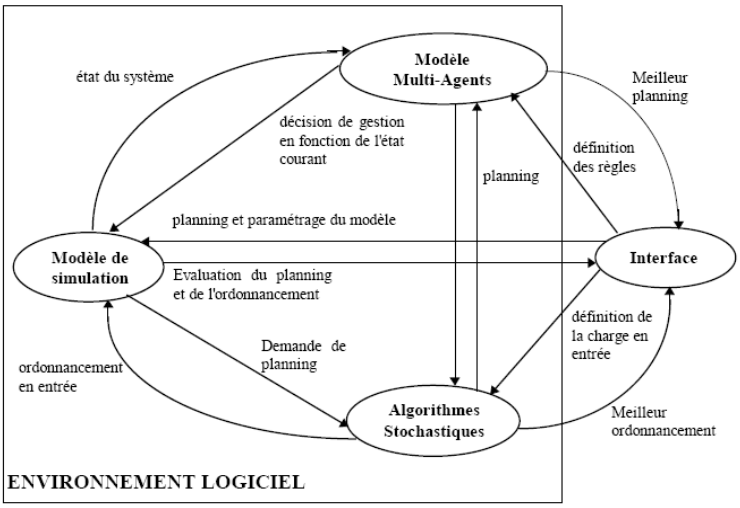


Figure 10. Couplage Triple

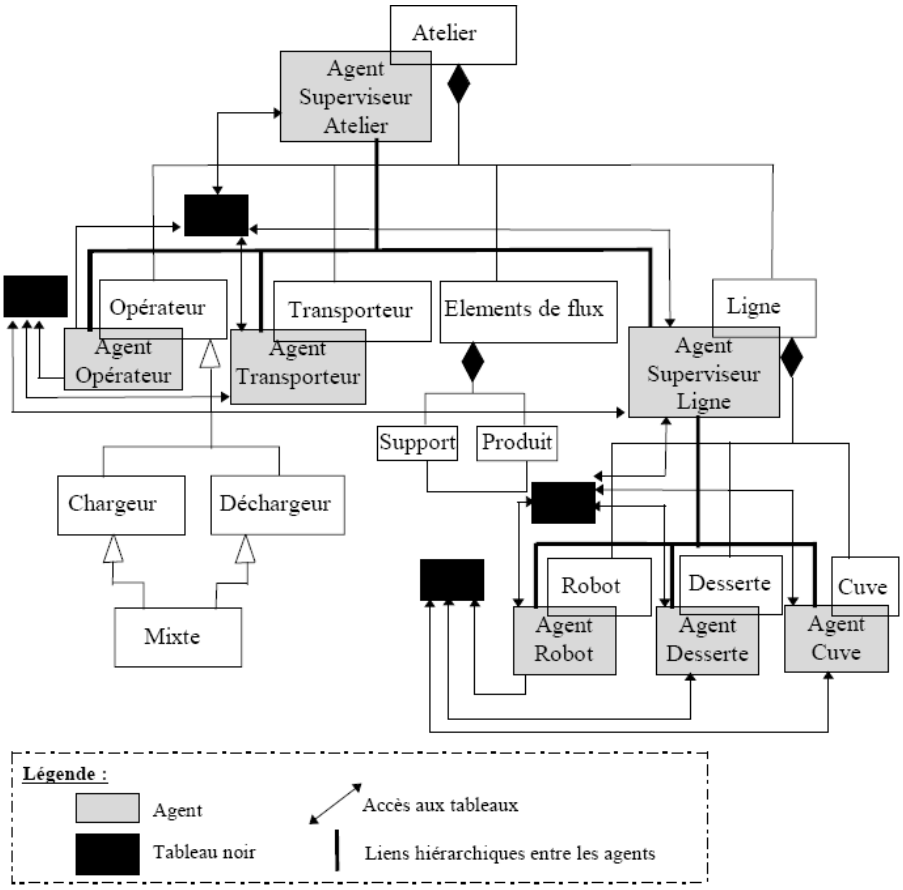


Figure 11. Organisation spatiale des modèles mobilisés

Une implantation sur site a été réalisée sous la forme d'un module d'aide à la décision (Figure 12) développé en langage Pascal (le module de supervision est implanté en LabView et celui de pilotage en C). Dans un contexte prédictif, la mise en œuvre du Couplage Triple est faite hors-ligne afin d'obtenir un ordonnancement amont et un planning des mouvements de robot satisfaisants. Dans un contexte réactif, l'interface est relayée par le système physique dont les requêtes parviennent à l'environnement logiciel par fenêtres de temps variables. Le module implanté se sollicite de trois manières :

(1) pour réaliser une planification à la demande du responsable humain,
(2) pour coller à la précédente planification dont le traitement n'est pas fini,
(3) pour recalculer dans un délai très court une nouvelle planification, suite à un aléa survenu sur la ligne (panne, variation constatée dans le temps réel de déplacement du robot…).

Des couplages réduits à l'utilisation de deux modules sont aussi possibles :

- Le couplage *modèle de simulation + modèle Multi-Agents* permet

d'améliorer le fonctionnement d'un ATS. L'objectif est alors de compléter un SI existant.

- Le couplage *modèle de simulation + algorithmes stochastiques* permet de résoudre des problèmes d'optimisation des ATS pour lesquels le SI ne peut pas évoluer. Il peut s'agir d'un ATS pour lequel l'objectif d'optimisation se réduit à l'ordonnancement amont.
- Le couplage *algorithmes stochastiques + modèle Multi-Agents* convient aux problèmes d'optimisation ne nécessitant pas d'évaluation. Les règles décisionnelles sont des algorithmes ou des formules explicites opérant sur des modèles mathématiques. Il reste alors à structurer le modèle Multi-Agents, en l'absence d'organisation spatiale physique prédéfinie.

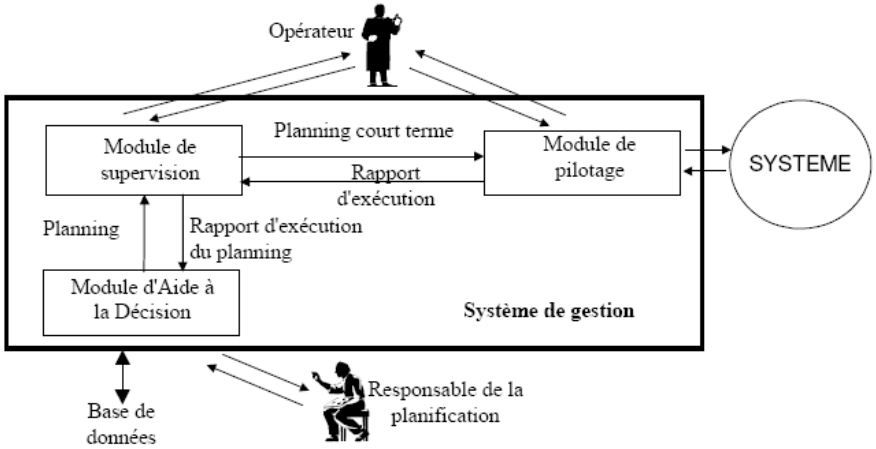


Figure 12. Implémentation du Couplage triple

2.1.2. Tissage simulation-centré de méthodes

Les deux cas précédemment étudiés illustrent parfaitement le possible glissement entre couplage fonctionnel (caractéristique du couplage simulation-centré classique) et couplage structurel. L'éclatement de la partie résolutoire en modules fonctionnels dont certains ont une très forte interactivité avec la simulation, place ces composants à la frontière entre modèle de simulation et méthodes résolutoires. Dans le cas du couplage triple, le modèle Multi-Agents peut apparaître comme un composant des méthodes résolutoires ou comme un composant du modèle de simulation. Ceci se décide, entre autres, en fonction du niveau d'interactivité de ce modèle avec les autres entités de la simulation. Si

des mécanismes de back-tracking sont intégrés aux décisions prises par les agents et que le reste des entités de simulation est conçu pour faire et défaire les actions liées à ces décisions et annulations, alors tous (modèle Multi-Agents et modèle de simulation) peuvent se retrouver sous la même gestion de temps simulé (et constituer donc un modèle global de simulation). Ceci a évidemment une influence sur la définition des fonctions F (réorganisation de M et EF) et G et sur celle du vecteur de paramètres $\gamma^{(m)}$ décrit précédemment.

En somme, la philosophie du tissage simulation-centré est que la solution du problème provient du comportement d'un système dynamique global, dont la structure est à spécifier. Cette solution s'apparente souvent à l'activité d'un ou de plusieurs processus, d'où une tendance à l'usage de formalismes à processus (organigramme, diagramme d'activité, BPMN, etc.) pour décrire le procédé de résolution. Néanmoins, le tissage n'est vraiment effectif que lorsque les considérations de type processus (en l'occurrence les activités et la concurrence) sont intimement liés aux considérations habituellement prises en compte dans la spécification de modèle de système (en l'occurrence les états et les événements).

2.1.3. Etats/Evénements versus Activités/Concurrence

Les perspectives de modélisation sont définies dans [Zeigler 1976] selon deux critères fondamentaux, à savoir : le temps et les changements d'état du système. Selon ces critères, trois grandes perspectives de modélisation pour simulation des systèmes dynamiques sont possibles :

- Le temps est (i.e., évolue en) continu, et les changements d'état (du système) sont continus. Une telle spécification est dite DESS (pour Differential Equation System Specification). Les formalismes qui s'y prêtent le mieux sont les équations différentielles (partielles, ordinaires, du premier ou second ordre, etc.).
- Le temps est discret (il fait des sauts par incréments fixes), et les changements d'état sont discrets (car ne pouvant se faire qu'à ces instants discrets). Une telle spécification est dite DTSS (pour Discrete Time System Specification). Parmi les formalismes qui s'y prêtent bien, citons les automates cellulaires, les équations aux différences, les L-System, etc. (pour les divers formalismes mentionnés, voir [Fishwick 1995] pour une

présentation très pédagogique).

- Le temps est continu, mais les changements d'état sont discrets (les intervalles de temps entre deux changements d'état consécutifs sont de longueurs variables). Une telle spécification est dite DEVS (pour Discrete Event System Specification). De multiples formalismes s'y prêtent bien, tels les Réseaux de Pétri, les automates à états finis, les réseaux de files d'attente, etc.

La passerelle de DESS vers DTSS est possible par discrétisation du temps et approximation du comportement du système selon cette base temporelle. Celle de DTSS vers DEVS semble plus naturelle, du fait que la première perspective peut être facilement interprétée comme un cas particulier de la seconde. Celle de DESS directement vers DEVS est possible à travers la technique de quantisation [Zeigler et al. 2000]. Ces possibilités de passerelle rendent légitimes le choix de la perspective DEVS comme cadre universel de modélisation pour simulation.

Si la classification en DESS, DTSS, et DEVS se veut être un spectre complet des spécifications possibles pour la simulation des systèmes dynamiques, leur dénominateur commun réside dans le fait qu'elles adoptent toutes une perspective encore plus fondamentale. Selon cette dernière, un système se caractérise à tout instant par un état, qui subit un changement suite à des événements qui sont soit internes, soit externes (les événements externes traduisant l'interaction du système avec son environnement). Ce paradigme Etats/Evénements constitue un ***Modèle de Calcul*** (MoC, pour Model of Computation [Buck et al. 1994]) commun à toutes les spécifications de la théorie de la M&S [Zeigler 1976].

D'autres formes de description sont possibles, ne reposant pas sur ce MoC, comme montré en Figure 13. Des formalismes comme CSP (Communicating Sequential Processes), PN (Process Network), SDF (Synchronous Data Flow), etc. reposent plutôt sur les notions d'activités et de concurrence. D'autres formalismes bien connus de nature similaire sont les diagrammes d'activités et les diagrammes de séquence d'UML, ou encore BPMN (Business Process Modeling Notation). Selon le MoC Activités/Concurrence, un système exécute à tout instant une activité (incluant la réception d'événements externes et l'émission d'événements vers l'extérieur), les changements d'activité étant

dictés par les règles de concurrence avec les autres systèmes (accès à des ressources partagées et communications synchrones/asynchrones avec l'environnement).

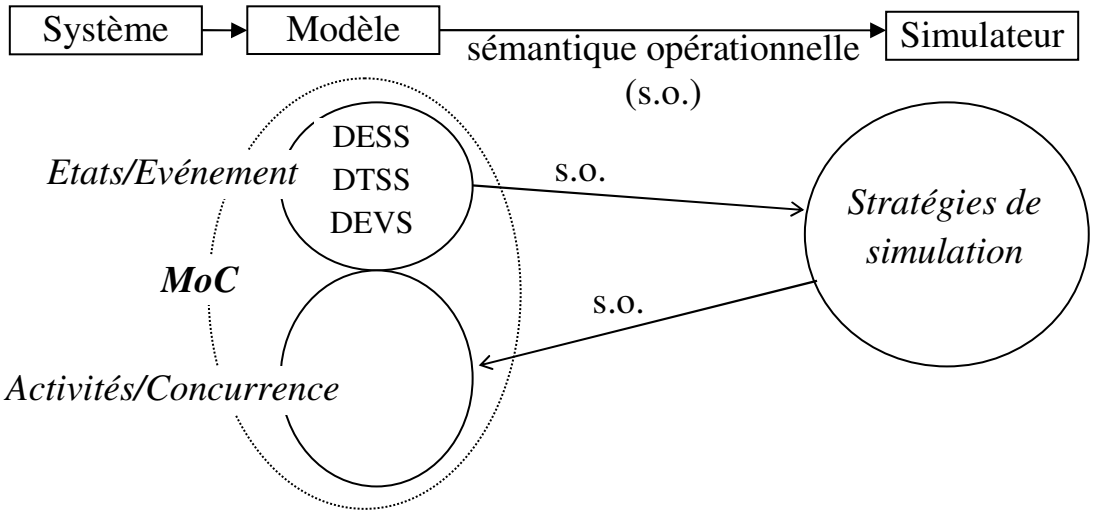


Figure 13. Etats/Evénements versus Activités/Concurrence

Notre point de vue est que le MoC Activités/Concurrence est dual du MoC Etats/Evénements. Mieux, nous constatons que de manière classique, en phase de modélisation, c'est le MoC Etats/Evénements (MoC EE) qui est privilégié, et que la sémantique opérationnelle proposée pour les modèles spécifiés s'appuie systématiquement sur un MoC Activités/Concurrence (MoC AC). En témoignent toutes les stratégies de simulation qui existent (toutes présentées en chapitre 3, à savoir : événements-centrés, activités-centrés, processus-centrés, processus parallèles optimistes et pessimistes). Toutes s'expriment en terme de flux de contrôle : dans les cas séquentiels, un unique processus exécute toutes les activités qui réalisent la simulation ; dans les cas purement parallèles, plusieurs processus concurrents sont en charge des opérations de gestion de la simulation (y compris les activités de synchronisation entre eux). Notons que dans ce dernier cas, les processus sont bien concurrents même si l'approche est purement parallèle, car malgré la présence de plusieurs processeurs exécutant en parallèle la simulation, chaque processeur est probablement en charge de l'exécution concurrente d'activités locales (par exemple, la réception de message, de manière concurrente à la consommation d'événements internes).

Une illustration de ce point de vue est donnée en Figure 14, avec la spécification en MoC AC de la stratégie (dite aussi protocole) de simulation DEVS. Le

modèle global (CM1) y est une construction modulaire hiérarchique en MoC EE (AM pour les composants dits atomiques, CM pour les composants dits couplés), alors que la sémantique opérationnelle est exprimée en MoC AC (Racine, Coordinateur et Simulateur sont des processus concurrents, chacun en charge de générer le comportement d'un des composants du modèle global, les simulateurs se chargeant des composants atomiques, les coordinateurs des composants couplés, et la Racine de la gestion du temps virtuel global). La Figure 14 ne montre que les activités des simulateurs et de la racine, mais celles des coordinateurs sont définies de manière similaire ([Zeigler et al. 2000]).

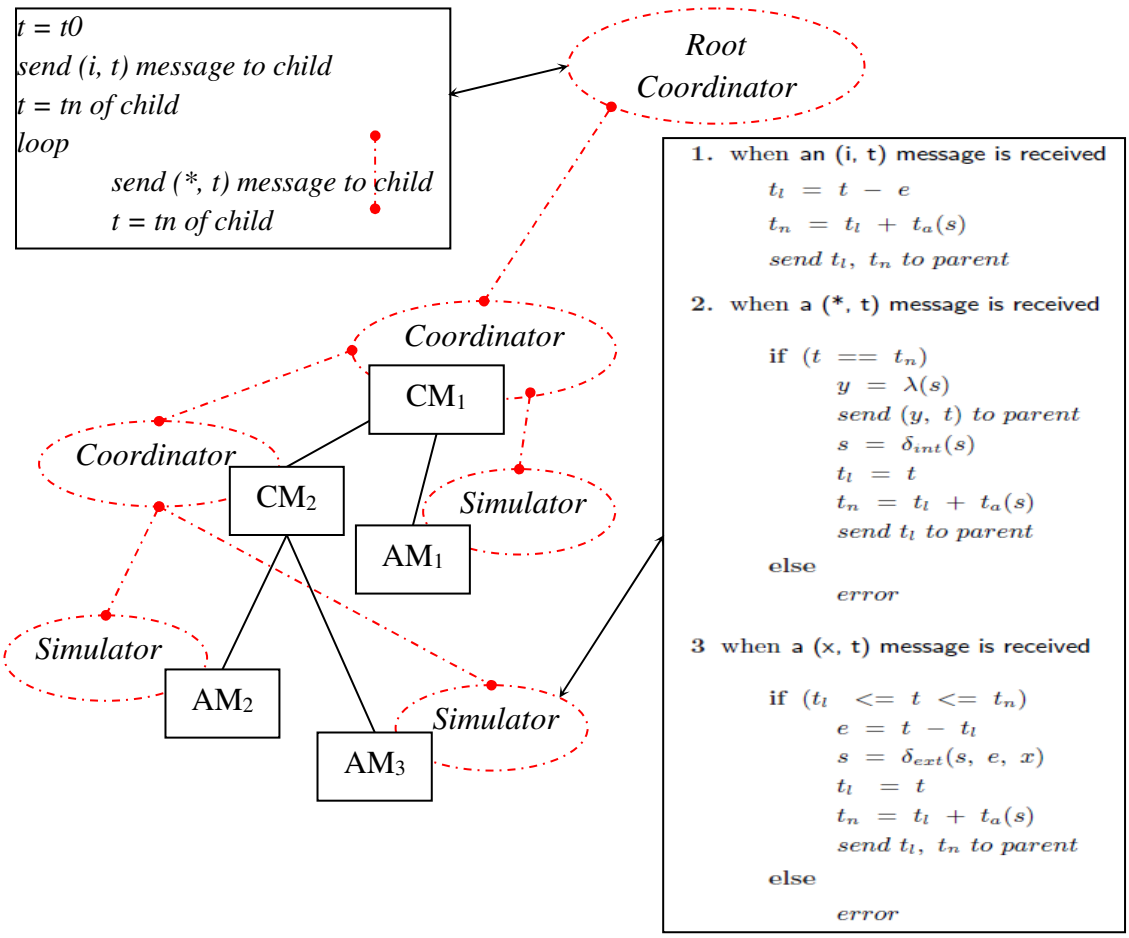


Figure 14. Sémantique opérationnelle en MoC AC de DEVS

Une formalisation à caractère universel du MoC AC (à l'image de celle du MoC EE proposée par DEVS) étendrait les possibilités de modélisation pour simulation aux processus de résolution, pourvu qu'une sémantique opérationnelle soit disponible (elle-même exprimée en MoC AC). En d'autres termes, une telle spécification serait simulable, donc correspondrait à l'esprit du tissage de méthodes, où le modèle de résolution est un système dynamique traité

sous l'angle de la théorie des systèmes au même titre que les modèles des systèmes mobilisés dans cette résolution.

2.1.3.i. Formalisation du MoC AC

Selon la théorie de la M&S ([Zeigler 1976]), emblématique de la MoC EE, un système est un automate à états (ensemble d'états S, fini ou infini), dont l'interface avec son environnement se définit par un ensemble d'entrées (X) et un ensemble de sortie (Y). L'état du système évolue le long d'une base de temps ($T \subseteq R^+$), comme le montre la Figure 15.

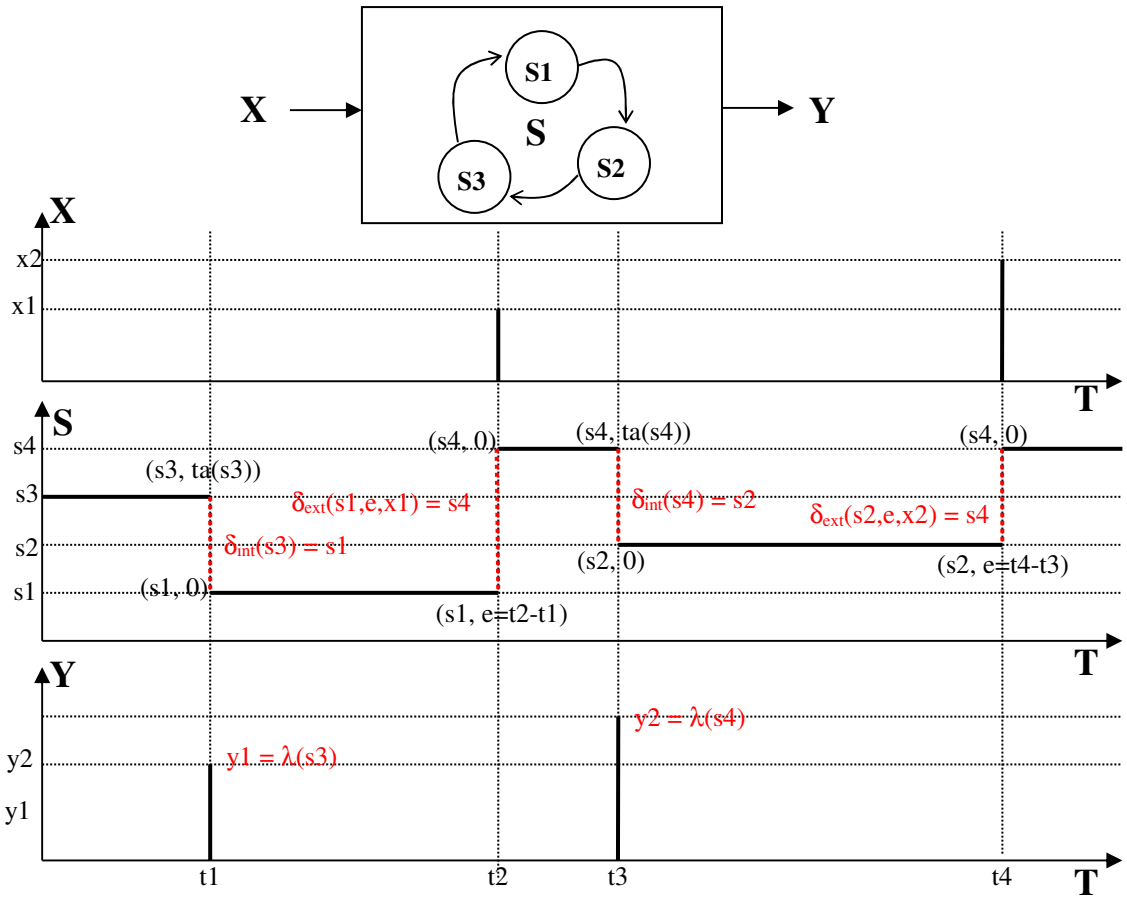


Figure 15. Comportement en MoC EE de système

A tout instant, le système est dans un état donné (auquel on associe un état total (s,e), s étant l'état en question et e le temps écoulé dans cet état), dont il ne sort qu'après y avoir passé un temps qui correspond, soit à sa durée de vie initialement prévue (définie par une fonction ta, dite d'avancement du temps),

soit à l'avènement d'un signal provenant de l'environnement du système (réception d'une entrée). Le système change alors d'état en effectuant une transition externe (dans le cas d'un stimulus externe) ou interne (dans le cas où la durée de vie définie par la fonction ta sur l'état courant est écoulée). Ces deux types de transitions sont définis par deux fonctions, respectivement appelées fonction de transition externe (δ_{ext}) et fonction de transition interne (δ_{int}). Les éventuelles réactions du système à son environnement sont traduites par une fonction de sortie (λ) qui n'est activée qu'en cas de transition interne. La Figure 15 montre l'évolution simultanée de l'état, des entrées et des sorties du système, en mettant en évidence les lois qui gouvernent son comportement (fonctions de transition, d'avancement du temps et de sortie).

La vision duale présentée par la Figure 16, décrit un système comme une entité qui est à tout instant en cours d'exécution d'une activité non interruptible (la dualité vient de l'opposition activité versus état, l'ensemble des activités est noté A). Dans cette perspective, les interactions avec l'environnement (échanges via l'interface I du système) sont elles aussi des activités ou des parties d'activité (envoi d'événements en sortie, récupération d'événements en entrée). Le système possède des propriétés (ensemble P) qui sont satisfaites ou pas au fil de la vie du système (de manière structurée, des variables ou des clauses logiques sont utilisables pour matérialiser ces propriétés ; si ce sont des variables, elles sont similaires, voire identiques dans certains cas, aux variables d'état utilisées dans le MoC EE). Ces propriétés sont modifiées par les activités conduites.

Lorsqu'une activité prend fin, le système procède à une transition d'activité selon une loi de transition qui peut être soit sur synchronisation, soit sur délai, soit sur condition :

- Quand le système, dans son activité courante, a besoin d'une synchronisation avec une ressource externe, il émet une demande via son interface, en direction de cette ressource (par exemple, y1 à t1 et y2 à t3). A l'opposé, il est capable de prendre en compte toute demande extérieure de synchronisation à la fin d'une activité non interruptible (x1 à t2 et x2 à t4). Dans ce dernier cas, la loi de transition sur synchronisation est définie pour cette activité (δ_{synch}). Il y a transition sur synchronisation si une sollicitation émise par l'environnement du système est constatée par ce dernier. Cette sollicitation se traduit par la mise à jour par l'extérieur de

variables correspondant aux entrées du système. Les règles de concurrence implicites font que ces variables sont des ressources partagées protégées par des mécanismes d'exclusion mutuelle.

- En l'absence de demande externe dans un délai défini sur l'activité sujette à transition sur synchronisation, une transition sur délai (δ_{del}) s'opère.
- Si la transition sur synchronisation n'est pas définie pour une activité, alors il y a, à la fin de cette activité, transition sur condition (δ_{cond}). Cette condition est évaluée en fonction de la propriété courante du système.
- Enfin, une activité en cours d'exécution peut être sujette à anomalie. Dans ce cas, il y a transition sur exception (δ_{exc}). L'arrêt du système est une activité terminale.

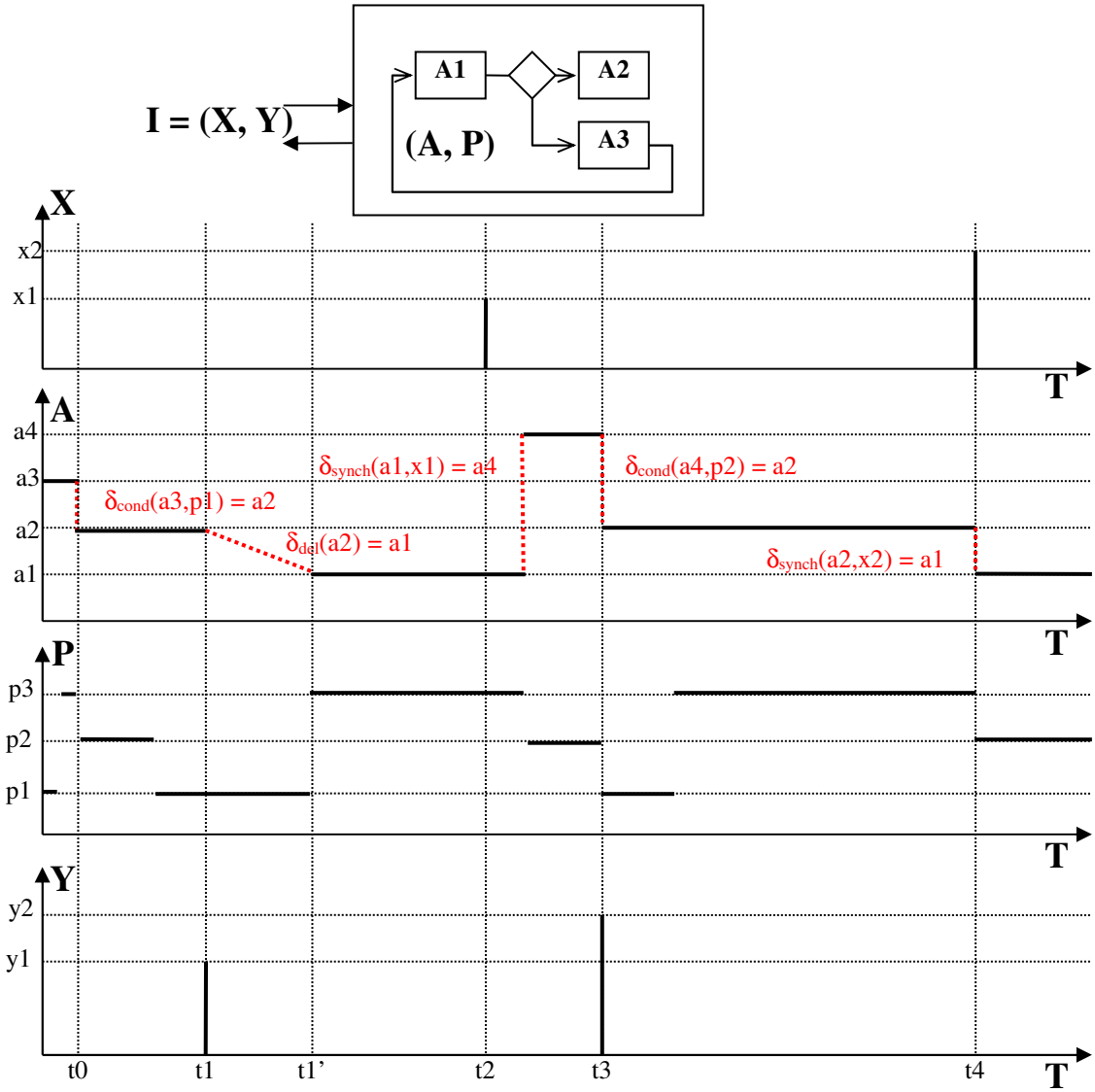


Figure 16. Comportement en MoC AC de système

La formalisation proposée pour cette vision duale est un modèle générique de réseau de processus communiquant par messages :

$$R = \{T_k\}_{k \in Card(R)}$$

Chaque processus T_k est une structure :

$$T_k = <I, A, P, \delta_{synch}, \delta_{del}, \delta_{cond}, d, B, \delta_{exc}>, \text{ où}$$

- I est l'interface de communication du processus.
- A est l'ensemble des activités insécables du processus.
- P est l'ensemble des propriétés possibles du processus.
- $\delta_{synch} : A \times I \rightarrow A$ est la fonction de transition d'activité sur synchronisation.
- $\delta_{del} : A \rightarrow A$ est la fonction de transition d'activité sur délai.
- $\delta_{cond} : A \times P \rightarrow A$ est la fonction de transition d'activité sur condition.
- $d : A \rightarrow R^{+\infty}$ est la fonction de délai accordé avant transition sur délai.
- B est l'ensemble des anomalies possibles.
- $\delta_{exc} : A \times B \rightarrow A$ est la fonction de transition d'activité sur exception.

Il faut noter que dans le MoC AC, le caractère hiérarchique ou non du réseau est dans la description des activités des processus. Dans une structure hiérarchique, les processus sollicités par tout processus ne le sont par aucun autre processus.

2.1.3.ii. Protocole de simulation des modèles en DCM AC

Ce protocole générique permet de produire de manière opérationnelle le comportement d'un modèle formalisé selon le DCM AC. Il repose sur une arborescence à un seul niveau au-dessous de la racine de simulation (Figure 17). Cette dernière (*Synchroniseur*) est un nœud de gestion du temps simulé, qui à chaque cycle, envoie une requête de synchronisation à tous les nœuds du réseau (*Get tLoc*) lui permettant de récupérer la date locale courante de chacun d'eux. Elle calcule ensuite la date courante globale simulée *tGlo* (minimum des dates locales récupérées, ou mise en œuvre d'algorithmes de calcul plus complexes), puis envoie une requête de synchronisation à tous les nœuds du réseau (*Set tGlo*), lui permettant de leur communiquer cette date calculée. La simulation progresse ainsi jusqu'à réalisation des conditions d'arrêt.

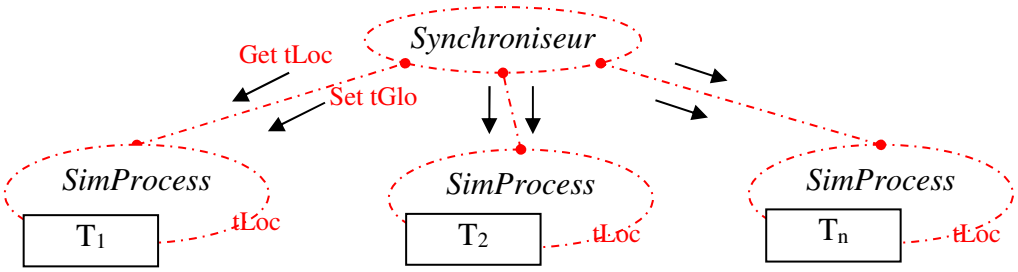


Figure 17. Sémantique opérationnelle du DCM AC

Les nœuds (*SimProcess*) sont, chacun, en charge de reproduire le comportement d'un processus T_k décrit au niveau conceptuel (Figure 18).

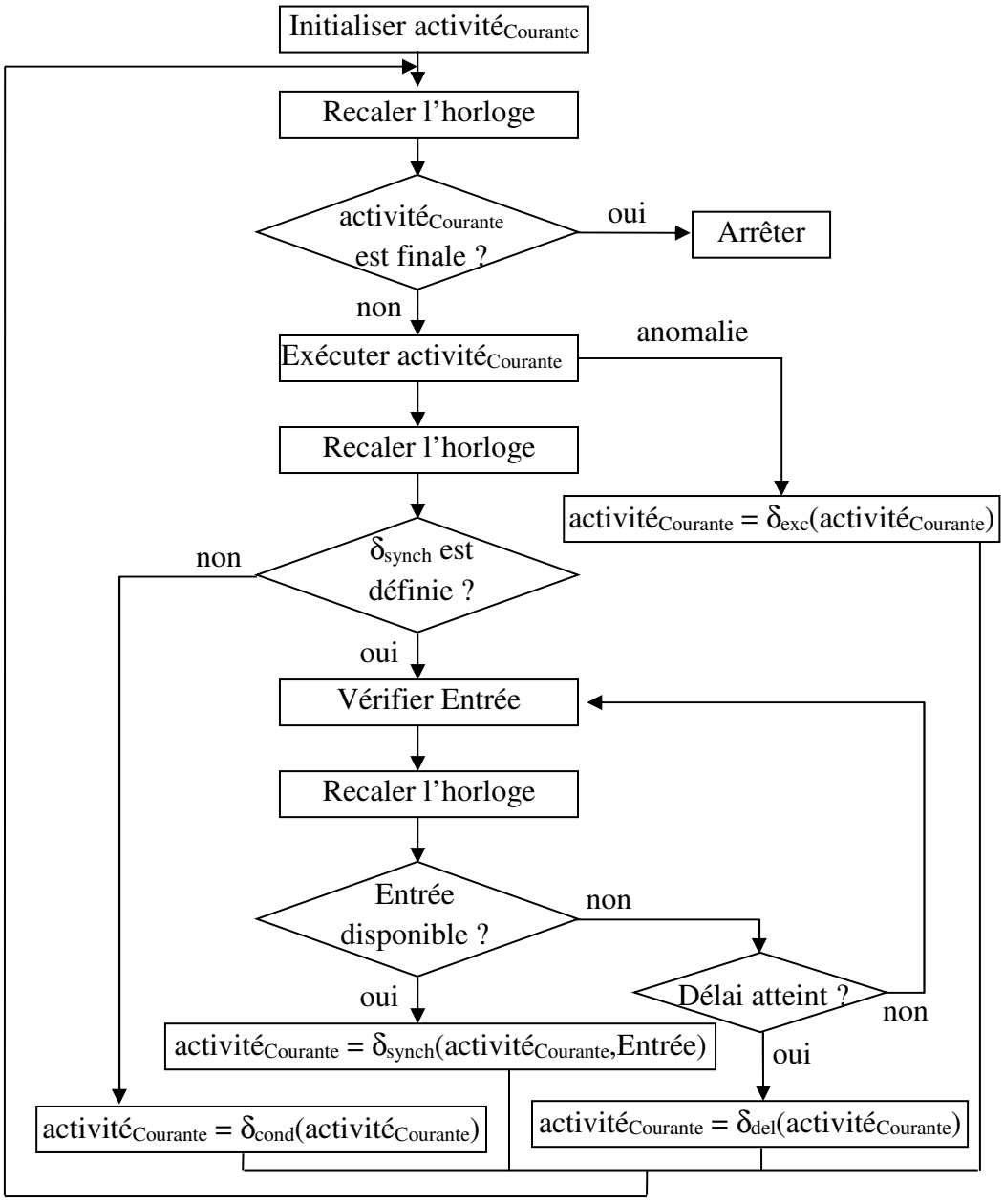


Figure 18. Comportement d'un processus de simulation concurrent

Tout processus de simulation gère sa date locale *tLoc* (si nécessaire, via un processus fils) et effectue, à la fin de l'activité insécable qu'il exécute, une transition sur synchronisation lui permettant de communiquer, soit avec la racine de simulation (pour transmission de la date locale et réception de la date courante de simulation), soit avec tout autre processus l'ayant sollicité pour une synchronisation (demande de ressource/service).

La structure à plat de ce protocole de simulation est indépendante de la structure hiérarchique ou non hiérarchique sous-jacente au modèle conceptuel. Force est également de constater que la description de ce schéma par la formalisation en MoC ACM proposée est triviale (activités, transitions d'activité, propriétés…).

2.2. Modèle de système

Une modélisation s'appuie sur le triptyque « Abstraction, Spécification, Association ». Il s'agit d'abord d'interpréter, via une élimination de détails, un objet concret par une abstraction. Puis, il faut produire une description de cette abstraction dans un formalisme adéquat (le produit étant alors une structure). Comment décider du formalisme le plus adéquat ? Ce qui fait défaut est une relation systématique entre phénoménologie et formalismes, i.e., une grille de lecture établissant une correspondance systématique entre un espace de phénomènes du réel et un espace de formalismes. La Table 3 est un (minuscule) fragment possible d'une telle grille.

Table 3. Phénoménologie et relation aux formalismes (fragment)

Espace de phénomènes	*Espace de formalismes*					
	Machine à états	Diagramme d'activité	Réseau de Pétri	Equations différentielles	Automates cellulaires	…
Diffusion				•	•	
Concurrence			•			
Synchronisation		•	•			
Transitions	•				•	
…						

La notion d'espace de formalismes est un concept qui transparaît derrière plusieurs travaux de recherche en prise avec la question de spécification d'abstractions hétérogènes. Elle permet de définir une approche méthodologique (voire théorique) qui permette de répondre à cette question de manière formelle ou semi formelle. C'est la structuration de cet espace de formalismes qui diffère dans ces différents travaux.

Quelques perspectives majeures sont réunies dans l'espace de formalisme proposé en Figure 19, où ne sont montrés que certains formalismes connus. L'approche de multi modélisation [Fishwick 1995] sous-entend une partition de l'espace des formalismes entre modèles déclaratifs, fonctionnels, à contrainte, et spatiaux, qui en terme de phénoménologie, mettent l'accent respectivement sur les concepts de dynamique (ou transition d'états ou d'événements), flux, équilibre et communication. Il est fréquent que la construction de ces modèles passent d'abord par celle d'un ou de plusieurs modèles conceptuels porteurs des concepts fondamentaux, et affinés ensuite. Ces modèles conceptuels servent à analyser le domaine ou le problème, mais rarement à la simulation.

L'approche multi-paradigme [Vangheluwe et al. 2002] souligne de manière explicite le fait qu'un formalisme se caractérise par son domaine syntaxique et une fonction de correspondance qui plonge par méta modélisation ce domaine dans le domaine syntaxique d'un autre formalisme (ce dernier est alors le domaine sémantique du premier). En d'autres termes, la sémantique d'un formalisme est donnée par un autre formalisme (DEVS, par son caractère universel, peut servir de domaine sémantique à une large classe de formalismes). Il est également possible que cette sémantique soit donnée directement par les stratégies de simulation définies pour ce formalisme (i.e., pour simuler les modèles exprimés dans ce formalisme ; on parle alors de sémantique opérationnelle).

Une spécification pour simulation se situe au niveau de connaissance « Structure » de la hiérarchie de Klir [Klir 1969] et les stratégies de simulation permettent d'en obtenir une connaissance au niveau « Données ». La Figure 19 est très loin d'être exhaustive dans la liste des formalismes présentés, à savoir : UML (Unified Modeling Language), FSA (Finite State Automata, ou machine à états finis), FEA (Finite Event Automata, ou machine à événements finis), Petri Net (Réseau de Pétri), QN (Queueing Network, ou réseau de files d'attente),

Flow Graph (graphe de flux), Kinetic graph (graphe kinétique), FBD (Functional Bolck Diagram, ou diagramme fonctionnel), Electrical Network (réseau électrique), DE (Differential Equations, ou équations différentielles), ODE (Ordinary Differential Equations, ou équations différentielles ordinaires), Bond Graph (graphe de contraintes), MAS (Multi Agent System, ou syst-me multi-agent), CA (Cellular Automata, ou automates cellulaires), et LS (Lyndenmeyer System, ou système de Lyndenmeyer), et PDE (Partial Differential Equations, ou équations différentielles partielles). La position de ces formalismes dans l'espace n'a d'autre sens, que leur classification dans la partition de Fishwick.

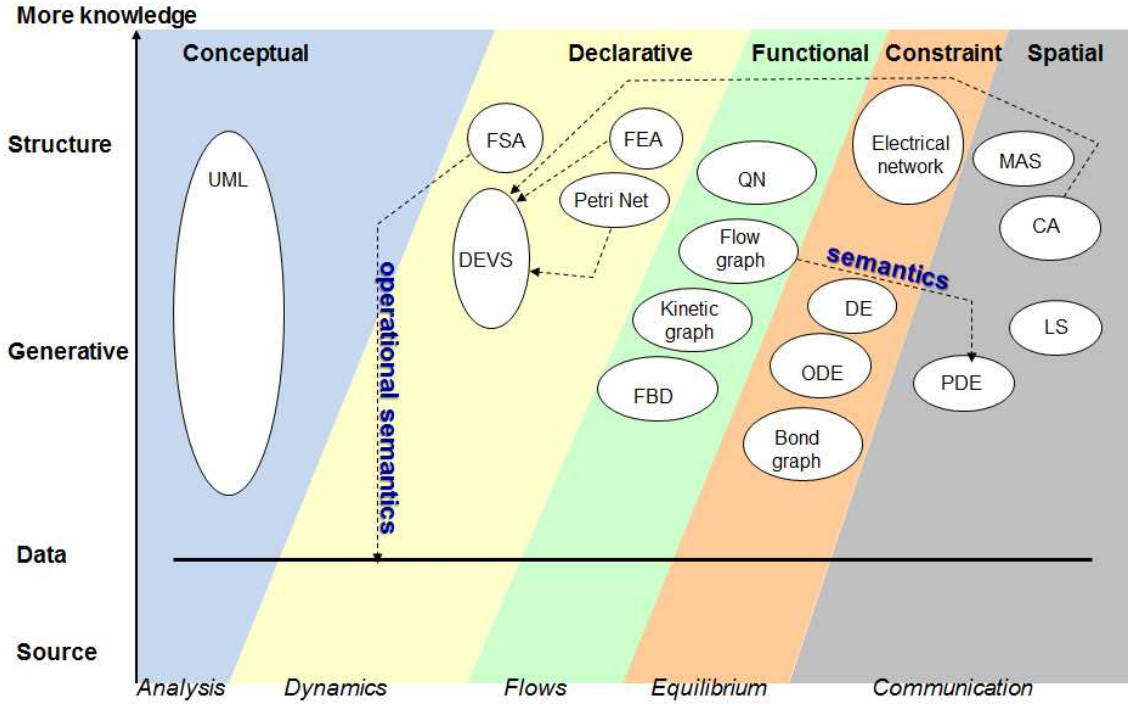


Figure 19. Espace des formalismes

Ce qui, généralement, advient en modélisation, en écho à l'approche de décomposition cartésienne (i.e., une abstraction se décompose en abstractions plus détaillées, qui à leur tour se décomposent… jusqu'à un niveau jugé le plus détaillé, ne nécessitant plus aucune décomposition), et qu'illustre la Figure 20, c'est un emboîtement d'abstractions, chacune s'exprimant dans un formalisme différent (le même formalisme pouvant évidemment apparaître plusieurs fois). En termes de hiérarchie de spécification, le système est décrit au niveau CN (un automate cellulaire, dans le cas de la Figure 20) et chacun de ses composants est décrit au même niveau ou au niveau IOS (une machine à états finis, dans le cas de la Figure 20), voire même au niveau IOFO (une équation différentielle ordinaire, dans le cas du premier sous-système de la Figure 20). On parle alors

de *raffinement* lorsqu'on va du système vers ses sous-systèmes et d'*abstraction* dans le sens contraire. Lorsque plusieurs niveaux d'abstractions sont en jeu, ce glissement progressif se heurte à une difficulté majeure : la cohérence globale de l'ensemble des spécifications hétérogènes.

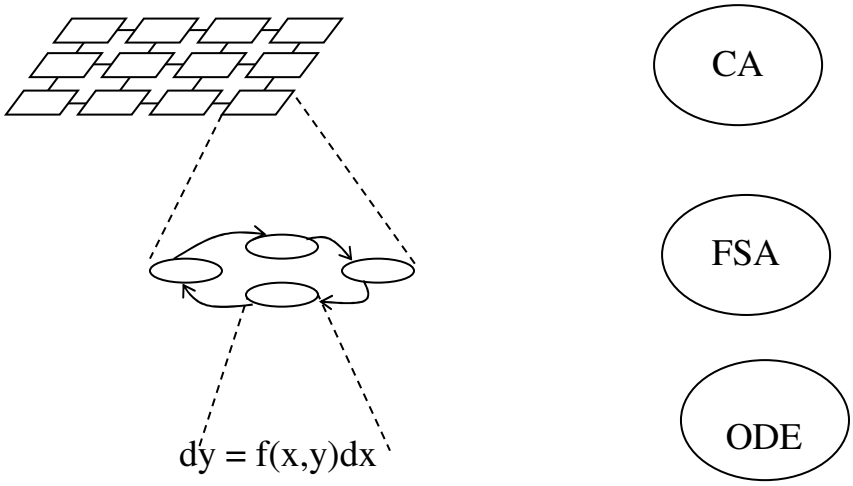


Figure 20. Spécification en poupées russes

En fait, il apparait un paradoxe de double besoin, à savoir la nécessité de multi formalisme d'une part, et le besoin d'holisme d'autre part. En effet :

- D'une part, en dehors des cas les plus simples, un modèle inclut plusieurs niveaux d'abstractions dont la spécification nécessite l'adoption de plusieurs formalismes, un seul étant très rarement adéquat pour tous les niveaux, voire pouvant masquer des caractéristiques qu'un autre formalisme mettrait facilement en évidence.
- D'autre part, un modèle multiple est difficile à maîtriser dans son expression globale et peu adapté à l'analyse de propriétés, contrairement à un modèle exprimé dans un unique formalisme. Structure et comportement comportent, chacun, des aspects statiques, dynamiques et fonctionnels, qu'il est peu aisé de distinguer clairement, car entremêlés dans une spécification globale qui mélange plusieurs formalismes. Une difficulté supplémentaire est l'alignement sémantique entre différentes parties du modèle global, exprimés dans différents formalismes, i.e., l'assurance que les propriétés exprimées par ces parties ne se contredisent pas, ou ne soient pas compatibles.

Il se pose donc au concepteur un problème d'intégration de formalismes : comment concilier la puissance d'expression que confère le multi formalisme et la maîtrise sémantique que confère l'adoption d'un formalisme unique ?

Cette question est abordée dans la littérature sous trois angles possibles :

- La fusion de spécifications hétérogènes, qui consiste à réaliser une somme des formalismes d'expression des spécifications concernées et à consacrer cette dernière comme un nouveau formalisme dont la syntaxe et la sémantique sont maîtrisées.
- L'interfaçage de spécifications hétérogènes, qui consiste à utiliser un formalisme tiers comme une glue permettant de combiner les spécifications concernées.
- La réécriture de spécification, qui consiste à choisir un formalisme cible comme dénominateur commun, dans lequel les spécifications hétérogènes sont traduites. Cette cible peut être, soit celui des formalismes d'expression de ces spécifications qui possède le plus grand pouvoir d'expression, soit un tout autre formalisme remplissant cette condition, voire un nouveau formalisme construit dans ce but.

2.2.1. Fusion de spécifications hétérogènes

L'arrivée d'UML [Rumbaugh et al. 1999] puis de SysML [Weilkiens 2008] a consacré la composition de spécifications. UML (Unified Modeling Language) juxtapose des formalismes initialement développés de manière séparée par ses concepteurs, et propose une composition de spécifications faite de neuf diagrammes qui permettent d'exprimer quatre vues : la structure, le comportement, l'implémentation et l'environnement. La structure et l'implémentation sont réputées traduire les aspects statiques du modèle, tandis que le comportement et l'environnement sont réputées traduire ses aspects dynamiques et fonctionnels. Alors que UML est dédié à l'ingénierie logicielle, SysML est son pendant pour l'ingénierie des systèmes. Ce dernier est donc plus général certes, mais ne contient pas le premier, dont il reprend certains diagrammes auxquels il rajoute d'autres qui tiennent compte d'aspects non présents dans les systèmes logiciels mais présents dans les systèmes mécaniques, électriques… Et il laisse tomber certains diagrammes d'UML jugés trop spécifiques aux systèmes logiciels. Les approches d'unification de

processus M&S par l'orienté objet, précédemment signalées, s'appuient sur ce formalisme intégré (terme plus conforme que « formalisme unifié ») pour fournir un mode d'expression à leurs modèles.

La littérature foisonne d'approches de fusion de spécifications hétérogènes (dont un exemple notable est le formalisme µSZ [Bussow et al. 1997]) qui reposent sur une juxtaposition de formalismes jugés adéquats pour spécifier les aspects statique, dynamique et fonctionnel :

- spécification des aspects statiques avec, par exemple, un outil d'analyse (diagramme de classes UML, spécification SES/MB [Zeigler 1984]…) ou un modèle spatial (organisation multi agents, schéma en L-System, disposition cellulaire…) ;
- spécification des aspects dynamiques avec, par exemple, un modèle déclaratif (réseau de Pétri, automate à états…) ;
- spécification des aspects fonctionnels (qui détermine souvent l'outil d'implémentation) avec, par exemple, un modèle fonctionnel (diagramme de bloc, réseau de files d'attente…).

Très souvent, des schémas de fusion de formalismes, d'outils et d'environnements sont sources d'unification des paradigmes sous-jacents, comme par exemple [Buck et al. 1994] pour l'environnement Ptolemy, ou [Mokkarala et al. 1985] pour la vérification temporelle des modèles de simulation par approche fonctionnelle, ou encore [Raina & Muthukumar 2009] qui unifie divers environnements de simulation de circuits intégrés.

2.2.2. Interfaçage de spécifications hétérogènes

Une manière assez pratique d'aborder la composition de spécifications hétérogènes est de les interfacer entre elles. Une démarche générique d'un tel interfaçage est proposée par [Fishwick 1995]. Cette approche apporte tout d'abord un intéressant éclairage à la question du multi formalisme, en proposant une classification des abstractions et en indiquant, pour chacune des classes, une famille de formalismes adéquats :

- les modèles conceptuels, décrivent au plus haut niveau d'abstraction, la connaissance générale sur le système étudié. Leur objet est de mettre en évidence les entités du système (ou de la classe de systèmes) considéré(e),

ainsi que les relations qui les lient. Ils constituent donc une base de connaissance pour les abstractions ultérieures. UML ou le langage naturel sont des exemples de formalisme adéquat pour ce niveau.

- Les modèles déclaratifs proposent une description de système sous forme de séquences d'événements qui font passer le système d'un état à un autre. Ce niveau d'abstraction est adéquat pour représenter la dynamique des objets du modèle conceptuel. Les automates séquentiels (pour les objets passifs) ou les réseaux de Pétri (pour les objets actifs) sont des exemples de formalisme adéquats pour ce niveau.
- Les modèles fonctionnels proposent une description de système sous forme d'enchaînement de fonctions (les sorties des unes constituant les entrées des autres), disposés tels que le traitement final effectué par le système se retrouve en aval de cet enchaînement. Un tel niveau d'abstraction spécifie la circulation des flux au sein du système. Les réseaux de file d'attente ou les diagrammes de blocs fonctionnels sont des exemples de formalisme adéquat pour ce niveau.
- Les modèles de contraintes expriment les lois et contraintes du système. Les équations différentielles ou les équations aux différences sont des exemples de formalisme adéquat pour ce niveau.
- Les modèles spatiaux s'attachent à décrire les règles (au sens de la décision) de fonctionnement du système. L'idée maîtresse, à ce niveau d'abstraction, est de représenter les décisions globales comme le fruit des interactions de multiples décisions locales. Les automates cellulaires ou les systèmes multi agents sont des exemples de formalisme adéquat pour ce niveau.

Le modèle final issu de cette spécification, appelé *multi modèle*, est une hiérarchie de modèles à interfacer entre eux afin de garantir le passage cohérent d'un niveau d'abstraction à un autre (Figure 21). Fishwick propose d'interfacer les différents modèles au moyen de l'approche objet : chaque modèle est encapsulé dans un objet, et les modèles communiquent entre eux par envois de messages. La sémantique opérationnelle de ce modèle multiple est obtenue par l'implémentation orientée objet et l'intégration des mécanismes de gestion du temps (stratégies de simulation séquentielle, stratégies de simulation parallèle et distribuée, mise en œuvre d'outils ou d'environnements de simulation, comme SimPack…).

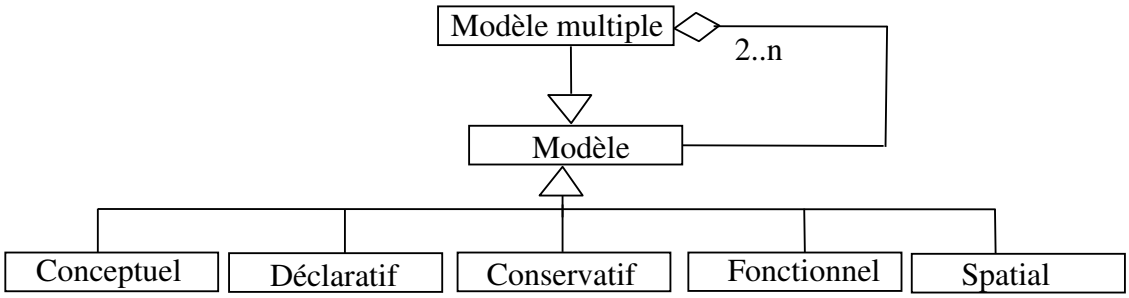


Figure 21. Meta modèle UML d'un modèle multiple

L'objet, comme concept, sert alors de glue inter paradigmes dans une perspective de M&S. Un exemple très représentatif est [Borshchev & Filippov 2004] qui unifie, par l'orienté objet, la dynamique des systèmes [Forester 1961] et le multi agents dans un cadre de simulation.

Certaines démarches d'intégration de spécifications hétérogènes sont à la frontière de l'interfaçage et et de la fusion, et peuvent :

- s'apparenter à l'une ou l'autre forme, selon la perspective dans laquelle on les place, comme [Friesen et al.1998] qui intègre UMLh et ZimOO, le second étant utilisé pour décrire le comportement des entités mises en évidence par le premier ;
- ou même laisser la possibilité d'aller dans l'une ou l'autre direction, comme [Paige 1997] qui propose une démarche générique pour intégrer des modèles formels hétérogènes. Les approches hybrides, qui sont des intégrations de modèles à événements discrets et de modèles continus, sont de même nature : par exemple, DEVS&DESS [Zeigler et al. 2000] est une fusion entre modèles discrets (DEVS) et continus (DESS), alors que la même démarche adoptée dans [Rabelo et al. 2005] ne donne pas lieu à un formalisme à syntaxe et sémantique unifiées.

2.2.3. Réécriture de spécifications hétérogènes

Dès lors que la complexité croissante des systèmes et problèmes rend le multi formalisme inévitable, ce que nous révèle la littérature en la matière est un double élan à la fois « progressiste » et « conservateur » :

- Progressiste, car l'intégration de formalismes incite à définir, par approches novatrices, de nouveaux formalismes plus puissants que ceux existants.

- Conservateur, car la définition perpétuelle de nouveaux formalismes n'est pas l'unique alternative. Il est également avantageux de tenter de tirer tout le profit des avancées réalisées avec les formalismes existants, à savoir, les résultats théoriques connus (par exemple sur les réseaux de Pétri, les files d'attente…) pour l'analysabilité, les outils et environnements évolués (comme QNAP, SIMSCRIPT, SIMAN, SLAM, SimPack…) pour la simulation (voir par exemple [Smith 2000] pour un panorama), ou les méthodologies dérivées pour une combinaison de bénéfices (comme [Kreutzer 1986], [MacDougall 1987]).

Ceci pousse à la réécriture de spécification. Cette démarche consiste à choisir un formalisme cible comme dénominateur commun, dans lequel les spécifications hétérogènes sont traduites. Cette cible peut être, soit celui parmi les formalismes d'expression de ces spécifications hétérogènes qui possède le plus grand pouvoir d'expression, soit un tout autre formalisme remplissant cette condition, voire un nouveau formalisme construit dans ce but.

2.2.3.i. Transformation de formalisme

Un moyen efficace de réécrire des spécifications passe par la méta-modélisation et la transformation de formalisme. En effet, si le méta modèle de chaque formalisme impliqué est capturé sous la forme d'un graphe, la transformation de formalisme se ramène à une grammaire de transformation de graphe, et partant tout modèle exprimé dans un des formalismes peut être transformé en un modèle exprimé dans l'autre formalisme. Dans un contexte de multi formalisme, plusieurs grammaires de transformation de graphe vont être élaborées pour faire converger les multiples spécifications faites dans les divers formalismes vers une unique spécification dans le formalisme final. La grande difficulté de cette approche réside dans la gestion de la cohérence et des redondances qui ne manquent pas d'apparaître lors de la fusion de plusieurs niveaux d'abstraction. En effet, une même portion de réalité peut être présente dans plusieurs spécifications (qui correspondent à différentes vues) du même système, et a contrario, le risque existe de décrire dans des vues différentes des réalités contradictoires ou non complémentaires.

Les travaux les plus notoires dans cette voie en M&S sont ceux de Vangheluwe et du groupe qui travaille sur l'approche multi paradigme [Vangheluwe et al. 2002], [Vangheluwe & De Lara 2002]. D'autres perspectives de métamodélisation pour la M&S sont présentées dans [Merkuryeva & Merkuryev 1999] sous forme d'état de l'art.

La transformation de modèles est une question globale en Génie Logiciel. Une approche standard est MDA (Model Driven Architecture) [OMG 2003], initiative de l'OMG (Object Management Group). Initialement conçue, pour définir une démarche de synthèse automatique de code à partir d'un méta modèle (transformation de modèle à code), afin de permettre l'obtention de produits spécifiques à n'importe quelle plateforme (PSM, pour Platform Specific Model) à partir d'une même spécification conceptuelle (PIM, pour Platform Independent Model), cette approche s'applique également à la transformation de modèle à modèle. Plusieurs techniques autres que les grammaires de transformation de graphe existent dans ce contexte (voir [Czarnecki & Helsen 2003] pour une classification de ces méthodes).

2.2.3.ii. DEVS

Le formalisme par excellence pour servir de dénominateur commun (i.e., le formalisme cible pour la réécriture de spécification) est DEVS [Vangheluwe 2000]. DEVS permet de représenter un système de deux manières.

La première manière consiste à concevoir un modèle atomique, qui décrit le système comme une entité possédant un comportement autonome, via des changements d'état internes et un processus de gestion du temps, une réactivité par rapport à l'environnement, et un mécanisme de génération d'évènements (i.e., de sorties). Formellement, on définit un modèle DEVS atomique par :

$$M = <X, Y, S, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta>, \text{ où}$$

- X est l'ensemble des entrées. Généralement, on définit X comme un ensemble de données, chacune indiquant la donnée reçue par le système et le port sur lequel la donnée a été reçue. $X = \{(p, v) / p \in Iports, v \in dom(p)\}$ où Iports désigne l'ensemble des références des ports d'entrée.

- Y est l'ensemble des sorties. On le définit de façon similaire à X. $Y = \{(p, v) / p \in Oports, v \in dom(p)\}$ où Oports désigne l'ensemble des références des ports de sortie.
- S est l'ensemble des états possibles du système.
- ta est la fonction d'avancement du temps. Elle est utilisée pour définir la "durée de vie" d'un état, i.e. la durée durant laquelle le système reste dans un état donné avant de faire une transition interne. ta: $S \rightarrow R^+$.
- δ_{int} est la fonction de transition interne. Elle définit la façon dont le système change d'état de manière autonome. δ_{int}: $S \rightarrow S$.
- δ_{ext} est la fonction de transition externe. Elle définit la façon dont le système change d'état lorsqu'il est stimulé par un évènement exogène. δ_{ext}: $Q \times X^b \rightarrow S$, où $Q = \{(s, e) / s \in S, 0 \leq e < ta(s)\}$ et où X^b est l'ensemble des sacs d'événements sur X (un sac de X est une liste de valeurs de X qui autorise la répétition de valeur dans la liste).
- δ_{conf} est la fonction de conflit, qui définit le comportement du modèle lors d'une collision entre une transition externe et une transition interne (i.e., lorsque le système reçoit un événement exogène au moment même où la durée de vie de son état courant expire). δ_{conf}: $S \times X^b \rightarrow S$.
- λ est la fonction de sortie. Invoquée après chaque transition interne, elle définit les évènements générés par le système, en fonction de son état courant. λ: $S \rightarrow Y^b$.

DEVS fournit d'autre part la possibilité de décrire un système de façon hiérarchique, en concevant un modèle couplé. Ce dernier agrège des composants, qui peuvent être, soit des modèles atomiques, soit eux-mêmes des modèles couplés. Ces composants sont reliés par leurs ports de sortie et d'entrée ; ainsi, les évènements générés par un modèle deviennent des stimuli pour d'autres. Formellement, un modèle DEVS couplé est une structure :

$$M = \langle X, Y, D, \{M_d / d \in D\}, EIC, EOC, IC \rangle, \text{ où}$$

- X est l'ensemble des ports et des valeurs d'entrée.
- Y est l'ensemble des ports et des valeurs de sortie.
- D est l'ensemble des références (e.g., les noms) des composants.
- Pour chaque $d \in D$, M_d est un modèle DEVS (couplé ou atomique).
- EIC (External Input Coupling) définit le couplage entre les ports d'entrée du modèle couplé et les ports d'entrée de ses composants.

- EOC (External Output Coupling) définit le couplage entre les ports de sortie des composants et les ports de sortie du modèle couplé.
- IC (Internal Coupling) définit le couplage entre composants du modèle couplé (sorties vers entrées).

Cette structure hiérarchique permet de modéliser un système de façon analytique (i.e., descendante) ou synthétique (i.e., ascendante). Dans le premier cas, on commence par représenter le système avec un modèle couplé, puis on décompose chaque composant jusqu'à définir la base de la hiérarchie, i.e. les modèles atomiques. Dans l'autre cas, on conçoit ou on réutilise des modèles atomiques que l'on couple jusqu'à obtenir un modèle représentant de façon satisfaisante le système.

Enfin, comme vu précédemment, la sémantique opérationnelle de ces modèles (atomiques et couplés) est définie par un protocole consistant en une hiérarchie d'automates abstraits (appelés Coordinateurs et Simulateurs), sous le contrôle d'un automate gestionnaire du temps simulé (appelé Coordinateur Racine) [Zeigler 1976]. Les Simulateurs ont en charge de générer le comportement des modèles atomiques et les Coordinateurs celui des modèles couplés.

Pour autant, DEVS manque de communicabilité. En effet, il n'existe pas de syntaxe concrète graphique complète spécifiquement défini pour lui, ce qui rend la spécification peu accessible aux experts des domaines d'application non spécialistes du formalisme. Par ailleurs, le lecteur intéressé par plus de détails sur DEVS rencontrera dans la littérature deux versions fondamentales. La première remonte aux origines de ce paradigme, et est maintenant désignée sous le nom de Classic DEVS (ou CDEVS). Dans cette version, le modèle atomique n'a pas de fonction de conflit car le caractère purement séquentiel sous-jacent interdit la possibilité qu'un modèle atomique qui serait arrivé au terme de la durée de vie de son état courant puisse recevoir exactement à ce moment simulé une entrée provenant d'un autre modèle. Cette interdiction est garantie par l'introduction d'une fonction appelée Select dans le modèle couplé, qui permet au modélisateur de définir des priorités entre ses composants. Par ailleurs, dans cette version, la notion de sac d'événements n'existe pas, un modèle atomique ne pouvant pas recevoir plus d'une entrée à la fois, à tout instant simulé. Les algorithmes définis au niveau du protocole de simulation pour cette version diffèrent de ceux définis pour la seconde version apparue une décennie plus tard.

Cette dernière est reconnue dans la littérature sous le label PDEVS (ou Parallel DEVS) [Chow 1996]. Bien des auteurs, comme nous, font en réalité référence à elle, lorsqu'ils parlent de DEVS.

En outre, des extensions à DEVS ont été définies pour traiter des sous-classes particulières de systèmes (il est possible de voir là, une manière d'accroître la communicabilité du formalisme dans un domaine donné, en se rapprochant de la sous-classe de systèmes spécifiques à ce domaine). Citons, entre autres :

- DEVS symbolique (le temps prend des valeurs symboliques et non des valeurs réelles et il devient ainsi possible de faire de la prospective sur l'ensemble des trajectoires probables d'un modèle à partir d'une situation donnée) [Zeigler & Chi 1992].
- DEVS flou (la logique floue est utilisée pour décrire les transitions du modèle et exprimer ainsi l'incertitude caractéristique des phénomènes considérés) [Kwon et al. 1996].
- DEVS temps-réel (les modèles doivent s'exécuter en respectant des contraintes temporelles de l'horloge murale) [Zeigler & Kim 1993].
- DSDEVS (pour décrire les systèmes à structure variable) [Barros 1997].
- DEVS&DESS [Zeigler et al. 2000] est une extension qui réalise l'intégration flux continus/discrets.

2.2.4. Modèle multi-analyse

Il est maintenant bien admis, qu'un « bon » modèle est un modèle multi-perspective, i.e., une macédoine d'abstractions qui expriment différentes vues de la réalité (statique, dynamique et fonctionnelle). Que cette macédoine soit spécifiée par plusieurs formalismes ou convertie dans un seul, ce modèle est conçu pour une seule forme d'analyse, qui peut être la simulation, l'analyse formelle, l'émulation, etc.

En même temps, pour répondre aux diverses questions qui peuvent se poser autour d'un système, plusieurs formes d'analyse s'avèrent nécessaires. Les réponses apportées par simulation ne couvrent qu'une partie de ces questions, comme celles apportées par d'autres formes d'analyse. En d'autres termes, une seule forme d'analyse ne suffit pas pour faire une étude complète d'un réel complexe. Aussi, assiste-t-on traditionnellement, à la conception de plusieurs

modèles, chacun destiné à une forme spécifique d'analyse. Dans certains cas, chaque modèle est construit en repartant du système sous étude. Dans d'autres cas, un premier modèle conçu pour un type d'analyse donné, est ensuite réécrit pour servir de modèle à d'autres formes d'analyse. Ce dernier cas est plus difficile à réaliser lorsque les concepts exprimés dans le premier modèle ne couvrent pas (ou ne permettent pas de retrouver) les concepts requis pour les formes d'analyse ultérieures.

Ne peut-on concevoir un modèle qui puisse systématiquement servir à plusieurs types d'analyse à la fois ? Si oui, dans quel(s) formalisme(s) pourrait s'exprimer un tel modèle ? C'est pour répondre à ces interrogations, que nous introduisons la notion de modèle multi-analyse (nous pourrions aussi utiliser le terme de modèle polymorphe, dans le sens où le même modèle prend des formes différentes au regard des différentes formes d'analyse envisagées). Ici, comme suggéré par la Figure 22, l'accent est mis sur les trois formes d'analyse suivantes :

- La simulation (envisagée sous l'angle de la théorie des systèmes, i.e., la génération de comportement à l'aide de la structure ; en d'autres termes, la génération des trajectoires d'entrée, de sortie et des états d'un système, à partir de la connaissance de son architecture et de ses mécanismes internes). Cette forme d'analyse permet de révéler les propriétés dynamiques (ou transitoires) d'un système.
- L'analyse formelle (qui fait usage des méthodes formelles, elles-mêmes issues de la Logique). Cette forme d'analyse révèle les propriétés statiques (i.e., stationnaires, ou encore asymptotiques) d'un système.
- L'émulation (qui signifie que le modèle exprimé permet un prototypage du système, i.e., une version numérique qui fonctionne grandeur nature). Cette forme d'analyse permet de révéler les propriétés dynamiques d'un système que ne peut révéler la simulation, notamment l'interaction avec des acteurs non modélisées (tels, l'homme, la nature, etc.).

Nous envisageons la modélisation multi-analyse en termes de définition de langage unificateur. L'avantage dans ce cas est de conserver une même sémantique tout au long du cycle de développement d'un modèle, réduisant ainsi les incohérences sémantiques entre vues différentes d'un même système, et facilitant de fait le passage au code.

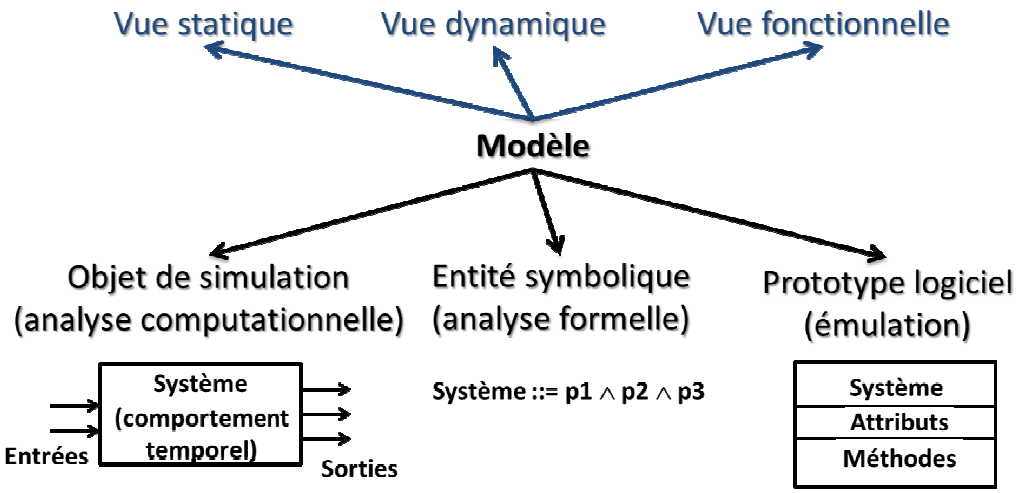


Figure 22. Approche multi-perspective versus approche multi-analyse

Nous introduisons ici le langage HiLLS (High Level Language for Systems Specification), un formalisme multi-analyse pour modéliser de manière graphique les systèmes complexes (y compris ceux représentatifs de processus de résolution), qui conjugue les qualités, pourtant difficiles à concilier, de communicabilité et d'universalité. D'une part, un formalisme graphique a le potentiel d'être aisément compréhensible de tous, ce qui permet aux acteurs venant de divers horizons scientifiques d'interagir plus facilement. Toutefois, une grande communicabilité a tendance à accroître la spécialisation du langage, d'où une baisse de son universalité. D'autre part, la recherche d'universalité (i.e., d'un haut pouvoir d'expression permettant la prise en compte de la plus large classe possible de systèmes) entraîne un haut niveau d'abstraction qui souvent, nuit à la communicabilité.

2.2.4.i. Ingénierie de langage

Comme décrit par la Figure 23, un langage possède une syntaxe abstraite (qui définit le vocabulaire et la grammaire du langage), une syntaxe concrète (qui définit les représentations données aux éléments du vocabulaire), et une sémantique (qui définit la signification des mots du langage). Dans le cas présent, les mots sont des modèles et la syntaxe abstraite est un méta-modèle décrivant tous les concepts pouvant apparaître dans ces modèles, ainsi que les liens qui les relient. Il est possible d'associer plusieurs syntaxes concrètes à une même syntaxe abstraite (ce qui donne plusieurs manières de s'exprimer visuellement dans ce langage). Toute syntaxe concrète peut être textuelle ou

graphique (dans ce dernier cas, on parle de langage de modélisation, alors que dans le premier cas, on parle de langage de spécification). La sémantique est définie par une fonction qui met en correspondance les éléments de la syntaxe abstraite, avec les éléments d'un autre langage (qui devient alors le domaine sémantique du langage en cours de définition). Dans le cas d'une approche multi-analyse, l'idée est de définir plusieurs fonctions de mise en correspondance entre la syntaxe abstraite d'un langage et différents domaines sémantiques qui permettent d'envisager différentes formes d'analyse.

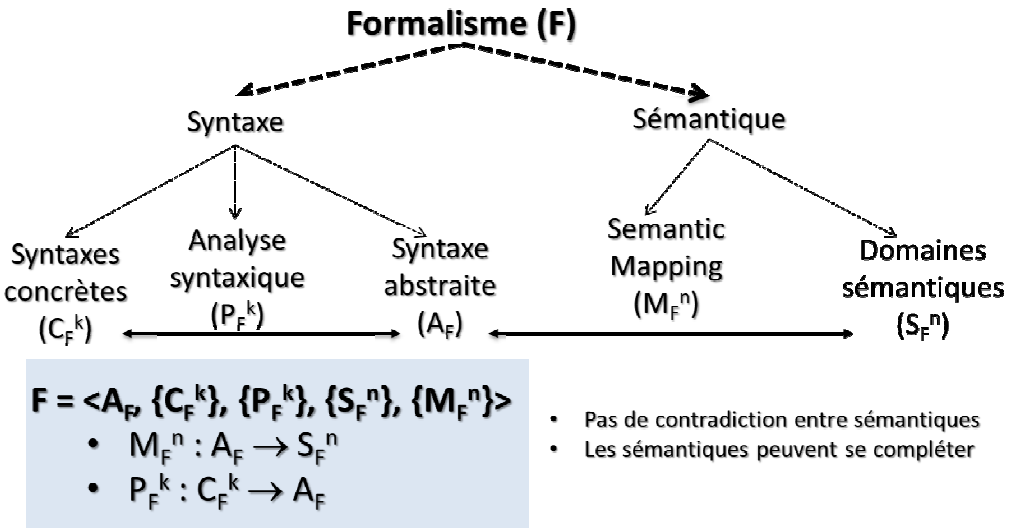


Figure 23. Langage multi-sémantique

Le langage multi-analyse HiLLS repose sur :

- Une syntaxe concrète graphique (unique), qui combine la vision de la théorie des systèmes (structure et comportement) à celle du Génie logiciel (aspects statiques, dynamiques et fonctionnels).
- Une sémantique opérationnelle donnée en DEVS, ce dernier opérant comme une machine de Turing de la simulation, et une sémantique opérationnelle destinée à l'émulation du système décrit, permettant donc son prototypage rapide (la sémantique pour simulation utilise un temps virtuel, alors que celle pour émulation utilise l'horloge murale).
- Une sémantique logique qui plonge dans les méthodes formelles, et qui permet de manipuler symboliquement les modèles décrits, comme des entités algébriques.

2.2.4.ii. Syntaxe concrète de HiLLS

Nous faisons, dans cet ouvrage, le choix d'une présentation intuitive de HiLLS. Certains aspects techniques sont présentés en détail dans [Aliyu et al. 2016], [Maïga et al. 2015], [Aliyu et al. 2015] et [Aliyu & Traoré 2015]. Un modèle HiLLS, comme présenté par la Figure 24, est une entité (HSystem) dotée d'un nom, et d'éventuels paramètres (par1, …, parn), i.e., des valeurs constantes utilisées dans la spécification. Il décrit un système capable de recevoir sur des ports d'entrée des données en provenance de son environnement. Ces ports sont décrits par leurs structures de données (portDecl1, …, portDecln). De manière similaire, des ports de sortie permettent au système d'envoyer des données vers son environnement. Les attributs internes du système (vu alors comme une entité au sens de l'orienté Objet, i.e., dotée d'attributs et de méthodes) sont capturés par des déclarations de variables, dont les propriétés sont exprimées sous forme de prédicats logiques. Les déclarations de variables et de leurs prédicats associés sont faites dans des structures appelées schémas (autant de schémas que nécessaire peuvent être définis). Les méthodes du système sont capturées par des schémas également (e.g., pour initialiser le système, effectuer des opérations de calcul, etc.). Les mécanismes permettant de générer le comportement du système sont décrits dans un diagramme dit de transition de configurations.

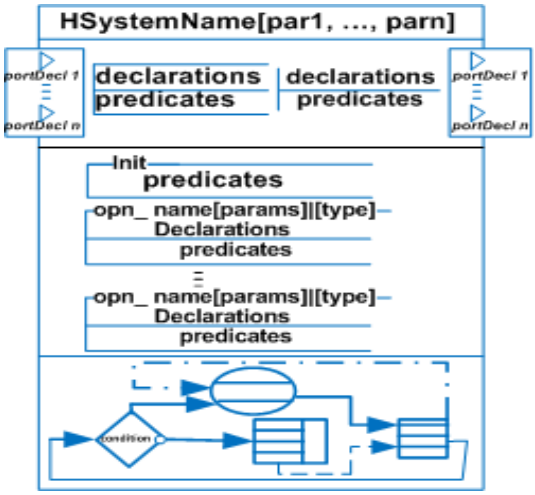


Figure 24. Architecture d'un modèle HiLLS

Une configuration correspond à une situation (terme plus général que « état ») dans laquelle le système peut se trouver. Cette situation, comme décrite par la Figure 25, est caractérisée par un label (i.e., une référence unique), des propriétés (i.e., les conditions que doivent vérifier les variables du système pour être dans cette situation), une durée de vie (sojournTime) et les activités (e.g.,

faire des appels à ses opérations internes) que le système effectue lorsqu'il se trouve dans cette situation. Dans certains cas, une configuration peut être composite, i.e., regrouper plusieurs autres configurations en son sein. Ses caractéristiques servent alors de facteurs communs à ses sous-configurations. La Figure 25 distingue également deux cas particuliers de configuration : celle de durée de vie infinie (dite configuration passive), représentée par un rectangle dont la façade droite est une ligne double, et celle de durée de vie nulle (dite configuration instantanée) représentée par un cercle (la situation initiale d'un système est une configuration instantanée, qui peut se représenter par un point noir si activité n'est entreprise et aucune propriété n'est à définir).

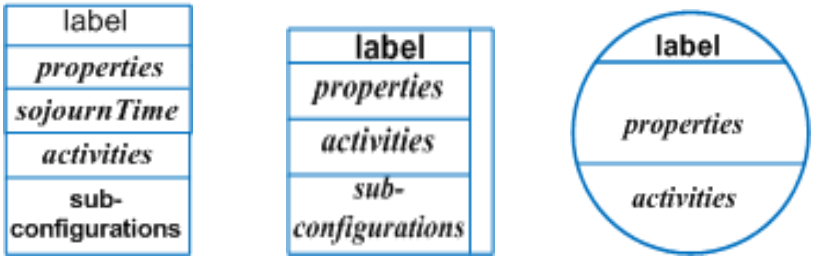


Figure 25. Configurations de modèle HiLLS

Deux configurations qui se succèdent sont reliées par une transition. Cette dernière intervient suite à un événement et sous condition (ou garde). Il est possible de dégager cinq types (non disjoints) de transition :

- La transition interne due à la fin de la durée de vie d'une configuration et qui provoque l'envoi d'une valeur sur la sortie du système (conformément à l'esprit du paradigme DEVS). Une telle transition part toujours de la façade droite d'une configuration vers la façade gauche d'une autre, comme le décrit la Figure 26. Par conséquent, aucune transition ne peut partir de la façade droite d'une configuration passive, alors que pour une configuration instantanée (y compris une configuration initiale), la notion de façade n'existe pas. L'envoi de valeur en sortie (outputEvents) est spécifié par un schéma ; dans ce cas, le nom de chaque port de sortie concerné est suivi d'un point d'exclamation, puis de la valeur envoyée (ceci est connu en théorie du control comme signifiant l'envoi d'une valeur sur une variable de sortie). Tout autre calcul nécessaire lors de cette transition (computations) est également décrit par un schéma.
- La transition externe due à l'arrivée d'une valeur d'entrée. Une telle transition part d'un flanc (haut ou bas) d'une configuration vers la façade

gauche d'une autre. Cette transition est soumise à plusieurs gardes ([trigger-1, …, trigger-n]). La principale garde est la réception de valeurs d'entrée, spécifiée pour chaque port d'entrée concerné, par le nom du port suivi d'un point d'interrogation, puis de la valeur reçue (comme en théorie du control. Une autre garde particulière est celle qui indique le temps que le système a déjà passé dans la configuration au moment de réception d'une entrée, et dont la valeur peut décider du sort de la transition (variable appelée « elapsed time » et nommée e dans DEVS). Tout autre calcul nécessaire lors de cette transition est également décrit.

- La transition confluente, qui, comme l'indique la Figure 28, est une transition à la limite entre un flanc (haut ou bas) et la façade droite d'une configuration. Elle correspond à la situation où le système reçoit une entrée au moment même de la fin de vie de sa configuration courante. Concernant une configuration instantanée, toute transition autre qu'interne est forcément une transition confluente.

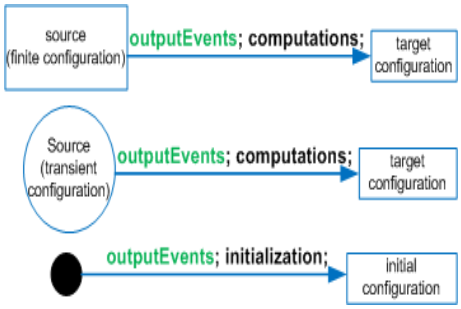


Figure 26. Transitions internes de configuration

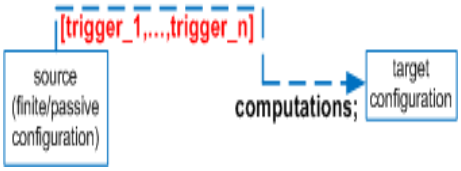


Figure 27. Transition externe de configuration

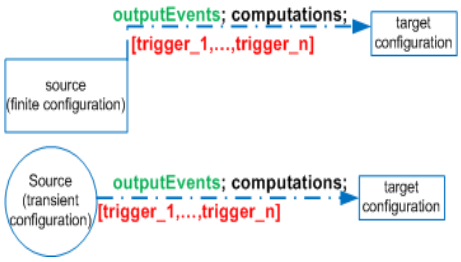


Figure 28. Transitions confluentes de configuration

Plusieurs transitions peuvent partager un nœud de décision, i.e., un point de test sur une condition dont le résultat détermine la transition effective. La figure 29 exhibe les trois cas de nœud de décision, qui correspondent respectivement aux transitions interne, externe et confluente.

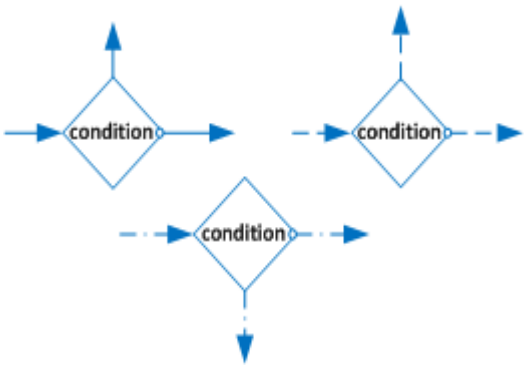


Figure 29. Nœud de décision

Un aspect intéressant avec les configurations et les transitions de configuration est que les trajectoires du système sont exprimées avec les mêmes concepts et symboles. Une trajectoire de système est un diagramme de transition de configuration qui est linéaire, et dans lequel les configurations et transitions décrites sont les situations réelles que le système a connu (valeurs réellement prises par les variables, durées réelles des configurations vecues, entrées réellement reçues et sorties réellement envoyées, etc.).

Par ailleurs, il est possible de construire de manière modulaire et hiérarchique des modèles HiLLS de plus en plus larges. Ceci est décrit par la partie gauche de la Figure 30, où un système est composé de m à n sous-systèmes. Les relations qu'entretiennent ces sous-systèmes correspondent à la configuration du système (et donc sont exprimées par les propriété de cette configuration). Un système à structure variable va donc avoir plusieurs configurations possibles et son diagramme de transition de configuration va décrire les règles de changement de structure du système. A l'opposé, un système « classique » n'aura dans son diagramme de transition de configuration qu'une transition initiale suivie d'une unique configuration passive décrivant les interconnections entre ses sous-systèmes. Par ailleurs, il est possible de dériver d'un système, un autre système qui pourrait spécialiser certaines caractéristiques de son système père (comme décrit dans la partie droite de la Figure 30). Les concepts de la Figure 30 proviennent du Génie logiciel (en l'occurrence, l'orienté objet), tandis que les

concepts précédents proviennent de la théorie des systèmes (tout en conjuguant les perspectives Etats/Evenements et Activités/Concurrence).

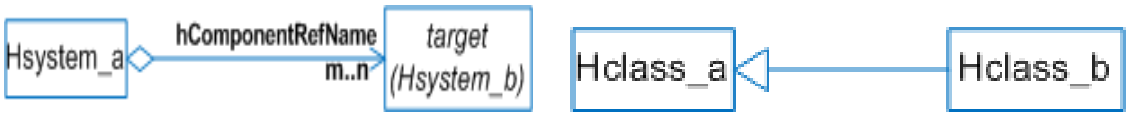


Figure 30. Composition et spécialisation de modèles HiLLS

2.2.4.iii. Exemple de modélisation HiLLS

Nous présentons ici les spécifications HiLLS d'un système (désigné par BVS, pour Beverage Veding System) constitué par l'interaction entre un utilisateur (désigné par U) et un distributeur automatique de boissons (désigné par BVM, pour Beverage Vending Machine).

Le BVM est configuré pour distribuer du cacao, du café, du jus d'orange et du jus de pomme aux prix respectifs de 1 €, 0,80 €, 1,20 € et 1,30 €. Il possède une interface de paiement et un clavier pour faire le choix de la boisson ou annuler la transaction en cours. Les boutons 1, 2, 3 et 4 de ce clavier sont utilisés pour choisir respectivement le cacao, le café, le jus d'orange et le jus de pomme, tandis que le bouton 5 annule une transaction en cours. L'interface de paiement n'accepte que les pièces suivantes en euros: 0,10 €, 0,20 €, 0,50 €, 1 € et 2 €. Toute autre pièce de monnaie est jugée «invalide» et rejetée immédiatement. Le distributeur dispose d'un receptacle pour délivrer les coupes de boissons et d'un autre pour le rendu de monnaie (solde pour les transactions terminées, remboursement pour les transactions annulées ou les pièces rejetées).

Une transaction commence avec le BVM par le choix d'une boisson, requête à laquelle il répond par une invitation à créditer le solde courant, et ce jusqu'à l'atteinte ou le dépassement du prix de la boisson sélectionnée. Un intervalle maximum de 2 minutes est autorisé entre les insertions de pièces. Si ce délai n'est pas respecté, la transaction est automatiquement annulée et les pièces insérées sont rendues par le BVM. C'est ce qui est fait aussi, lorsqu'une demande d'annulation est reçue avant la fin de la phase de créditement. Par contre, au terme d'une phase de créditement, aucune annulation n'est plus possible, et une phase de distribution commence. La boisson demandée est alors

délivrée au bout de 1 minute, les pièces insérées livrées au coffre du BVM et la monnaie retournée en cas de solde positif.

La Figure 31 propose une spécification par HiLLS du BVM. Deux ports d'entrée et deux ports de sortie sont définis. Le port *inC* est destiné à la réception des paiements, le port *code* au choix de la boisson, le port *outC* à la restitution de monnaie, et le port *cup* à la livraison de boisson. Les variables du système sont déclarées dans le schéma des attibuts : *credit* (pour le solde en cours), *price* (pour le montant à payer), et *current* (pour le montant de la dernière monnaie reçue). Les contraintes sur ces variables sont précisées dans le schéma. Les variables plus complexes apparaissent dans les relations entre le système et ses composants (y compris ceux qui ne sont pas des systèmes, mais de simples ressources). C'est le cas de *badC* (dernière monnaie invalide reçue), de *escrow* (monnaies déjà reçues pendant la transaction), et de *vault* (coffre-fort du distributeur), qui sont tous, des collections de 0, 1 ou plusieurs pièces de monnaie (d'où les différentes cardinalités indiquées sur les branches des relations de composition avec la ressource *Coin*). Les opérations que le système est capable d'effectuer sont décrites par des schémas de méthode, avec précision du type d'information requis pour chaque opération (paramètres du schéma) et du type d'information retourné par l'opération : *valid* (pour vérifier la validité d'une monnaie), *playWelcome* (pour afficher le message d'accueil à l'utilisateur), *setPrice* (pour déterminer le montant à payer en fonction de la boisson choisie), *playDispenseMessage* (pour afficher un message informant de l'état du processus de livraison), et *playCoinRequest* (pour inviter l'utilisateur à augmenter son solde). Un pseudo-code est utilisé ici pour décrire ces opérations, mais l'idée est de s'en tenir à la logique des prédicats autant que faire se peut.

Pour la partie dynamique (les parties statique et fonctionnelle ayant été exprimées à travers les ports, attributs et méthodes), le BVM présente un diagramme à 7 configurations, dont une configuration initiale. Les 6 autres sont les suivantes : *idle* (attente de requête pour une boisson), *charge* (créditement de solde en cours), *reject* (invalidation de monnaie reçue), *return* (retour de monnaie), *dispense* (distribution de boisson en cours) et *cancel* (annulation de requête). Chaque configuration possède des propriétés spécifiées par un prédicat sur les variables du système et une durée de vie (qui vaut, respectivement $+\infty$, 2, 0, 0, 1 et 0).

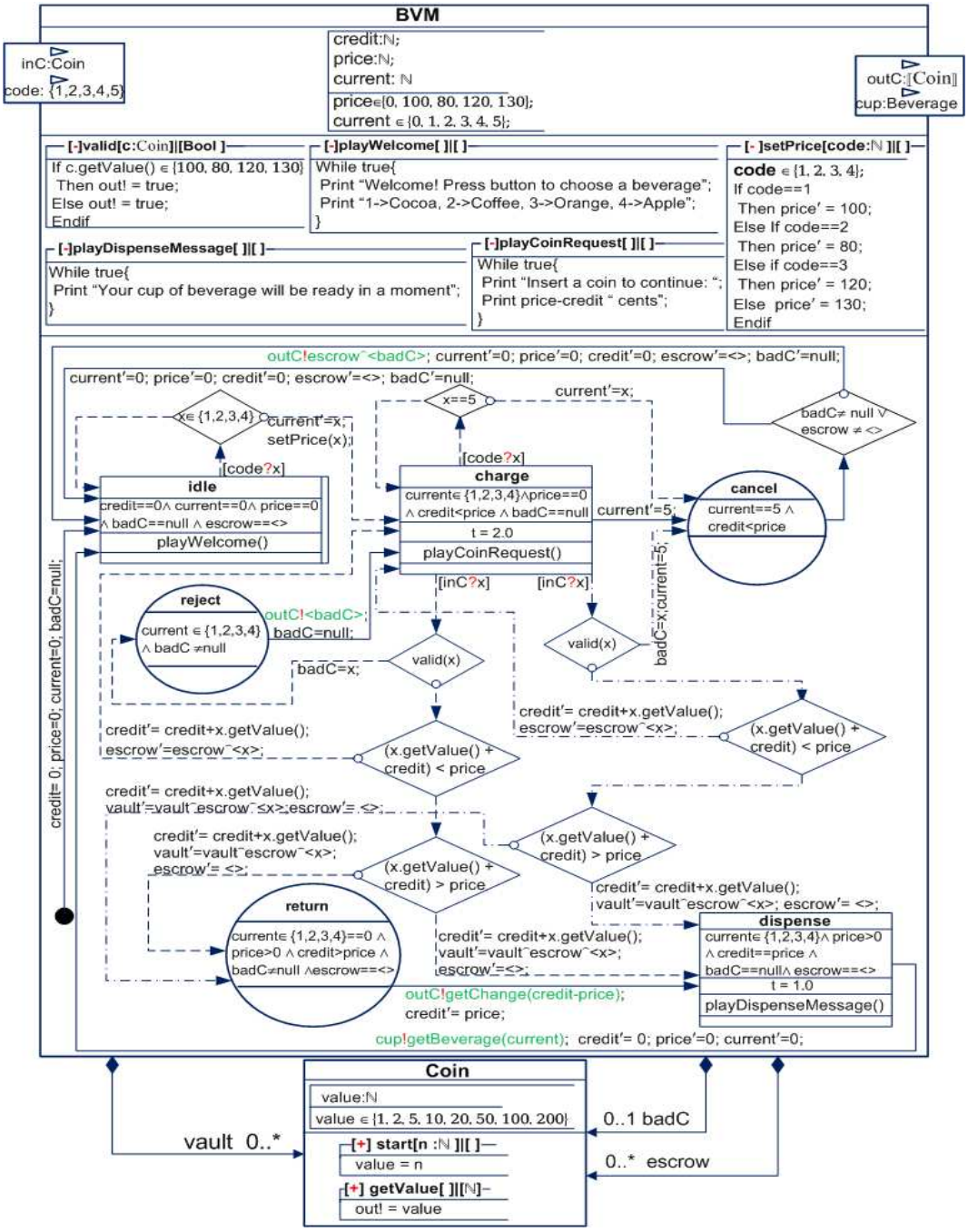


Figure 31. Modèle HiLLS d'un distributeur de boisson

La situation initiale met le système en *idle*, configuration dans laquelle il reste (en jouant son message d'accueil) jusqu'à ce qu'il reçoive un code en entrée (1,2,3, ou 4). Ceci déclenche une transition externe vers la configuration *charge*, tandis que le code reçu est utilisé pour calculer le prix à payer. En fait, le système passe en *idle* chaque fois qu'il n'y a plus de solde, plus de montant à payer, plus de monnaie valide ou invalide reçue, et plus de monnaie à rendre (credit == 0 ∧ current == 0 ∧ price == 0 ∧ badC == null ∧ escrow == <>). En *charge*, le système affiche le montant restant à payer. La transaction peut alors

se trouver en annulation, soit parce qu'aucune autre pièce n'a été reçue au bout de 2 unités de temps, soit parce qu'un ordre d'annulation est arrivé (*code* reçoit 5). En cas de réception de pièce de monnaie, cette dernière est vérifiée, puis soit rejetée (*badC*), soit mise au compte du solde de l'utilisateur (*escrow*). Si cette arrivée fait passer le solde au niveau ou au dessus du prix à payer, alors les pièces au compte du solde utilisateur sont versées dans le coffre-fort (*vault*), le solde restant est restitué (*return*), et la boisson demandée est délivrée (*dispense*). Sinon, le système revient en *charge*. Les opérations *getChange* et *getBeverage* sont des chémas dont la description n'apparait pas dans la Figure 31, faute d'espace. Par ailleurs, il faut noter la convention adoptée pour mettre à jour une variable ; elle fait usage du symbole *prime* (') pour signifier qu'il s'agit de la nouvelle valeur de la variable concernée (ainsi x' = x+1 veut dire que la valeur de x est incrémentée de 1).

La spécification HiLLS de l'utilisateur est donnée par la Figure 32. L'utilisateur peut recevoir des coupes de boisson, mais aussi des pièces de monnaie. Il peut aussi envoyer des pièces de monnaie, tout comme des requêtes pour boisson ou annulation. En situation *away*, qui dure une période aléatoire entre 1 et 10 minutes, il n'a pas de besoin particulier. Puis, de manière aléatoire, il développe l'envie d'une boisson, et passe alors en *ordering* pour émettre sa requête. Dans la phase de paiement (*inserting*), il lui faut 15 secondes pour choisir une pièce de monnaie. Une fois, le paiement effectué (*waiting*), il s'accorde un délai de 90 secondes pour recevoir sa boisson, sinon il annule la transaction (*cancelling*).

Le système entier (BVS) est décrit par la Figure 33. C'est un système fermé, donc ses interfaces d'entrée et de sortie sont vides. Il n'a pas de variables d'état mais le schéma d'état spécifie des contraintes sur les couplages entre les ports de ses composants. Par exemple, le prédicat *bvm.code == user.request* spécifie que la valeur du message sur le port *code* du BVM est toujours égale à celle du port *request* de l'utilisateur. Ces couplages sont établis (*connect*) au moment où le système s'initialise dans son unique configuration *working*.

Une des forces de HiLLS est qu'il devient très intuitif de décrire les systèmes à structure variable par des transitions de configuration, où chaque configuration décrit l'architecture interne du système sous forme de couplage de composants, tandis que les transitions permettent de décrire la variabilité de structure.

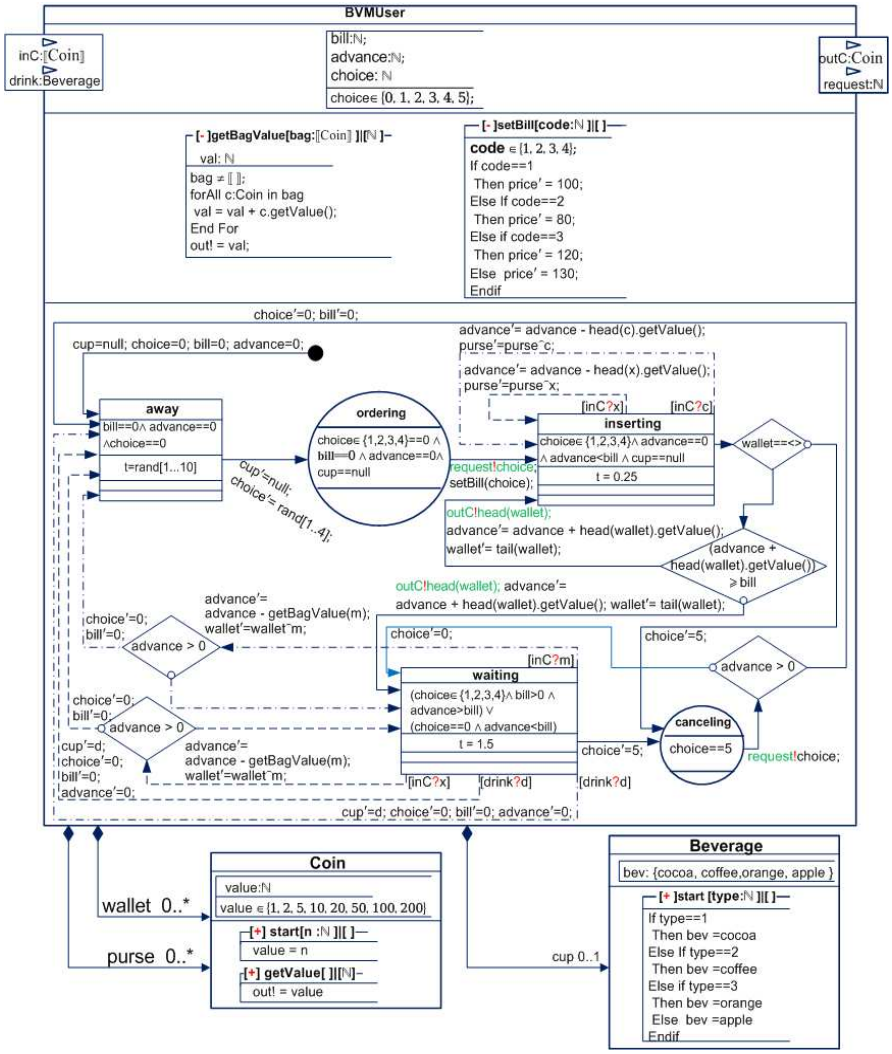


Figure 32. Modèle HiLLS d'un utilisateur

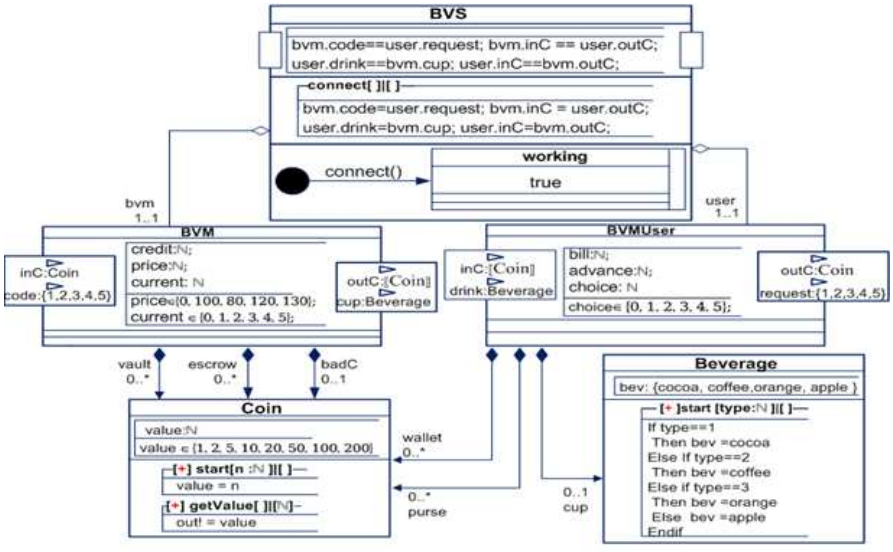


Figure 33. Modèle HiLLS du système de distribution de boisson

2.2.4.iv. Syntaxe abstraite de HiLLS

Cette manière intuitive d'introduire HiLLS ne doit pas faire perdre de vue le fait que ce langage est formellement décrit en termes de syntaxes et de sémantiques. La syntaxe abstraite de HiLLS est présentée en Figure 34.

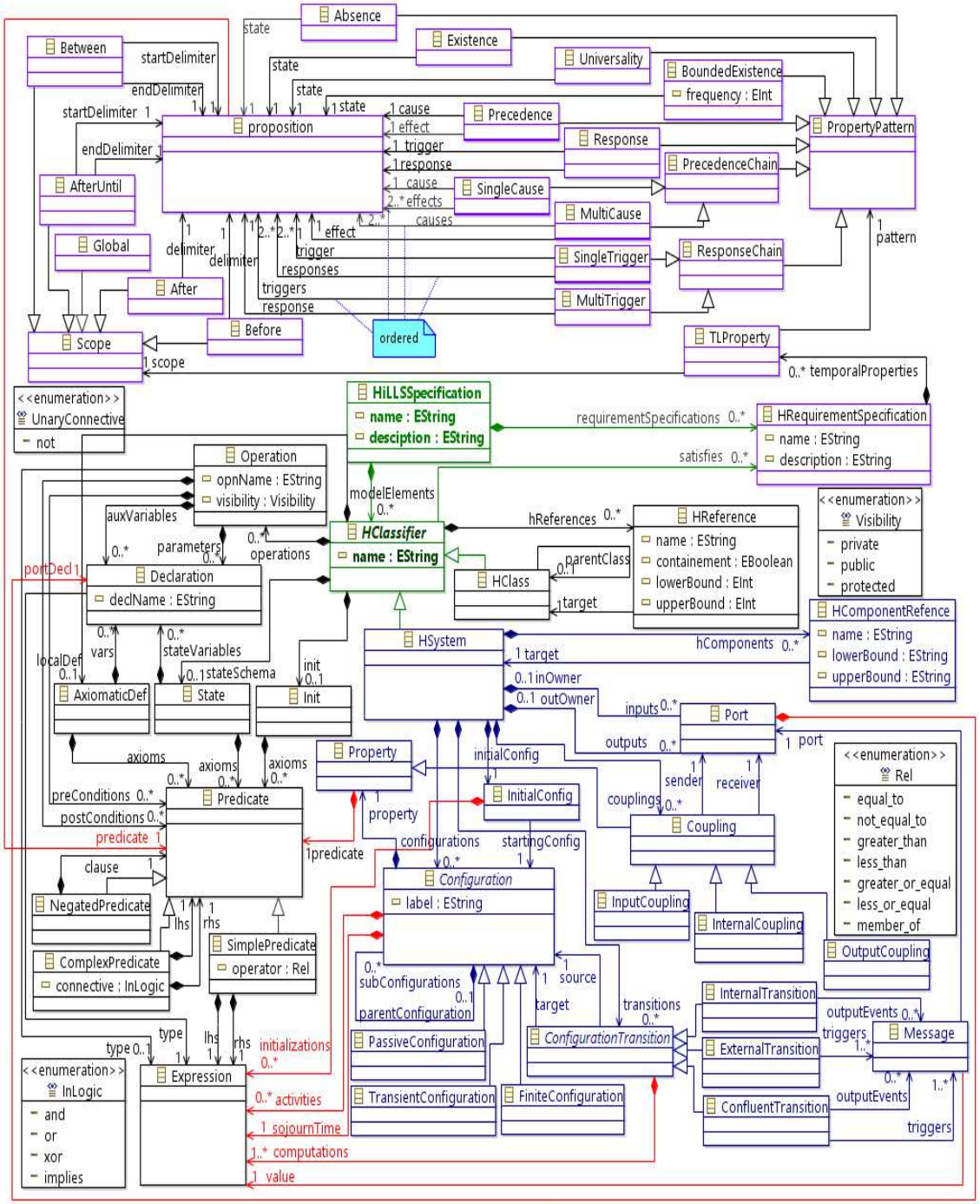


Figure 34. Syntaxe abstraite de HiLLS

Bien que cet ouvrage ne s'intéresse pas à la manière technique dont la syntaxe abstraite a été construite, il est bon de préciser qu'il s'agit d'un tissage des méta-modèles représentatifs des syntaxes abstraites des formalismes DEVS, Z et UML. C'est pourquoi, HiLLS hérite à la fois des concepts issus respectivement de la théorie des systèmes, de l'analyse formelle, et de l'orienté Objet.

La partie en bleue de la Figure 34 correspond aux concepts issus de DEVS, celle en vert provient de UML, celle en noir de Z et celle en pourpre de la Logique des prédicats.

Les sémantiques respectives de HiLLS sont développées (également de manière intuitive) dans les chapitres à venir.

Chapitre 3.
Synthèse de code

Il s'agit, à ce stade, de réaliser la partie basse du triptyque : « Abstraction, Spécification, Association », i.e. de substituer à une structure le comportement qu'il est censé produire, via la sémantique opérationnelle des formalismes utilisés en phase de spécification. Cette dernière s'exprime au moyen des stratégies de simulation mises en œuvre dans le contexte de ces formalismes.

3.1. Automate de simulation

La validité d'un automate de simulation repose sur son respect de deux propriétés élémentaires qui sont la propriété de vivacité et la propriété de sûreté :

- La propriété de vivacité est la certitude que le temps simulé progresse du début à la fin de la simulation, i.e. qu'elle ne bloque pas sur une valeur, ni ne régresse de manière irréversible. Cette propriété est garantie si le principe suivant, dit de déterminisme, est respecté : « le futur peut à tout instant être déterminé, au moins partiellement, et ce en fonction du présent et du passé ».
- La propriété de sûreté est l'exactitude de déroulement des événements. En d'autres termes, il faut que tous les événements qui doivent avoir lieu aient effectivement lieu, et que seuls ces événements aient lieu. Si l'ordre d'exécution des événements n'est pas respecté, certains événements qui doivent avoir lieu pourront s'en trouver annulés tandis que d'autres ne devant pas avoir lieu, vont apparaître. Le non-respect de l'ordre d'exécution des événements déroge à la contrainte de causalité (i.e., la cause précède l'effet). En somme, le futur influence le passé et on parle alors d'erreur de causalité.

Les stratégies de simulation mises en œuvre pour garantir ces deux propriétés se répartissent, dans la littérature, en deux grandes classes :

- les stratégies centralisées (ou « World views »), qui proposent une

génération séquentielle de comportement, et

- les stratégies réparties (simulation parallèle et distribuée), qui proposent, elles, la génération parallèle des fragments de comportement.

3.1.1. Stratégies centralisées

Les « world views » sont historiquement les premières stratégies de M&S (voir [Leroudier 1980] pour une introduction, [Overstreet & Nance 1986] pour une présentation détaillée, et [Zeigler 1976] pour une formalisation). Elles se répartissent en quatre classes :

- la stratégie événement-centrée (« event scheduling »),
- la stratégie activité-centrée (« activity scanning »),
- la stratégie processus-centrée (« process interaction »), et
- les stratégies hybrides.

3.1.1.i. Stratégie événements-centrée

Ici, tous les types d'événements susceptibles d'apparaître sont identifiés et un noyau de synchronisation constitué de variables d'état et de sous-programmes qui décrivent les changements d'état (i.e., les modifications des valeurs des variables d'état) associés aux événements est construit. Un échéancier permet de stocker les événements dont l'apparition est prévue (chaque événement ainsi prévu est estampillé par sa date supposée de réalisation). A chaque étape de la simulation, l'événement prévu d'estampille minimale est enlevé de l'échéancier pour être réalisé : ceci se traduit par la mise à jour des variables d'état, ainsi que la prévision de nouveaux événements et/ou l'annulation d'événements présents dans l'échéancier. Les actions liées à l'avènement d'un événement sont de durée nulle dans le temps simulé. L'ordre d'exécution des événements simultanés est fixé par l'implémentation, notamment par la manière dont sont insérés ces événements dans l'échéancier. Une mise en œuvre de cette technique est décrite dans [Sheppard et al. 1984]. Cette approche est utilisée dans les logiciels de simulation de première génération (QNAP2, GPSS, SIMSCRIPT…).

3.1.1.ii. Stratégie activités-centrée

Ici, ce sont les activités possibles du système qui sont identifiées et décrites. Chaque activité correspond à une séquence de traitements à faire sur les variables d'état lorsqu'une certaine condition est vérifiée. La simulation avance par incrément fixe, et à chaque pas, toutes les activités sont scrutées afin de trouver et d'exécuter celles dont la condition associée est vérifiée. Cette stratégie est attractive par son côté modulaire (de nouvelles activités peuvent facilement être intégrées, des anciennes peuvent être enlevées), mais elle est pénalisée par sa lenteur (toutes les activités sont scrutées à chaque pas de temps). Elle est adéquate pour simuler les modèles DTSS (i.e., à temps discret), tels les automates cellulaires, les éléments finis, etc.

3.1.1.iii. Stratégie processus-centrée

Dans la stratégie processus-centrée (qui peut être vue comme une vue combinée des deux précédentes techniques), les entités du système réel sont décrites par des processus (composants qui exécutent des activités, tentent d'accéder à des ressources, se mettent en veille lorsqu'une ressource demandée n'est pas disponible, et se réveillent à l'occurrence d'événements spécifiques pour poursuivre leurs activités). Une liste de processus suspendus est entretenue, et seul un processus est actif à la fois. Ce dernier est chargé, en fin d'activité (ou au moment de passer en veille), de réveiller un des autres processus en sommeil. Certains langages de simulation, tel que Simula, favorisent la mise en œuvre de cette stratégie. Des implémentations sont proposées dans [Bruno 1984], [Powers & Nute 1984] et [Sheppard et al. 1984]. Les logiciels de simulation GPSS, SIMAN, SLAM II et SIMSCRIPT l'utilisent.

3.1.1.iv. Stratégies centralisées hybrides

Des hybridations de stratégies centralisées ont été proposées, pour tirer profit de leurs bénéfices conjugués. C'est notamment le cas de l'approche triphasée [Tocher 1963], qui combine les techniques événement-centrée et activité-centrée. C'est aussi le cas du formalisme DEVS [Zeigler 1976] (qui propose,

dans sa sémantique opérationnelle, une sorte de synthèse des trois précédentes classes de stratégie).

3.1.2. Stratégies réparties

La finesse croissante du niveau de résolution dans la modélisation des systèmes complexes, a naturellement conduit à la question du parallélisme sous au moins l'une des deux formes suivantes :

- Les modèles deviennent coûteux en ressources mémoire et en temps, et une voie pour réduire ces coûts consiste à utiliser plusieurs processeurs et mémoires.
- L'idée intuitive que plus un modèle est réaliste, plus les résultats qu'ils génèrent sont proches de ceux du système réel modélisé, incite à rendre explicite dans le modèle, le parallélisme inhérent au système réel. L'analyse des systèmes complexes révèle très souvent que les problèmes sont naturellement posés de manière distribuée.

Selon [Righter & Walrand 1989], cinq stratégies sont possibles pour paralléliser une simulation :

- Paralléliser un programme de simulation séquentielle, en détectant automatiquement les parties de code pouvant être exécutées de concert ; une version de cette option a été proposée sous forme de décomposition hiérarchique consistant à exécuter de manière concurrente plusieurs sous événements d'un même événement [Concepcion 1989], [Zhang & Zeigler 1989]. L'un des premiers constats, dans ce domaine, fut que les techniques de vectorisation, issues du calcul parallèle, ne représentaient pas une voie prometteuse en simulation, en raison de la nature irrégulière et interdépendante des données d'un programme de simulation.
- Distribuer les expériences, en exécutant simultanément plusieurs simulations séquentielles dont les paramètres diffèrent. Les bénéfices de cette option apparaissent dans le cas de simulations fortement stochastiques nécessitant de longues sessions de simulation pour réduire la variance, ou lorsque le panel des paramètres de simulation est très large. Les inconvénients résultent principalement de deux facteurs : (1) chaque simulation doit pouvoir être supportée dans son intégralité par une ressource matérielle (processeur et mémoire), et il y a besoin d'autant de ressources matérielles que de

simulations ; (2) les résultats d'une session de simulation ne peuvent pas être utilisées pour guider le choix des paramètres d'entrée d'une autre session.

- Distribuer les fonctions du simulateur, en confiant à chaque processeur, une tâche de simulation (génération de nombres aléatoires, traitement des événements, prise de mesures et traitements statistiques associés, etc.) ; cette option a été développée par [Krishnamurthi et al. 1985] et par [Comfort 1984] ; le gain de temps qui en résulte est très limité.
- Distribuer les événements, en maintenant un échéancier global (de préférence dans une mémoire partagée) et en faisant traiter chaque événement par le premier processeur libre. C'est le champ de la simulation parallèle. Le multiprocesseur à mémoire partagée (MISD) est l'architecture matérielle hôte optimisée pour une telle technique.
- Distribuer le modèle en processus qui simulent, chacun, une partie du système, et entre lesquels, des mécanismes de synchronisation doivent exister. C'est le champ de la simulation distribuée.

Ces deux derniers champs constituent un même domaine, en raison de la similarité des questions qui s'y posent et des méthodes mises en œuvre : celui reconnu sous le label « Simulation Parallèle et Distribuée » (PDS pour « Parallel and Distributed Simulation », ou encore « Simulation répartie » en France).

3.1.2.i. Architecture générique

Construire une partition de codes de simulation en répartissant à dessein un grand code initial impose de concevoir un schéma d'exécution (ensemble de processus logiques de calcul en interaction) qui s'y prête. L'état global du système est représenté par une partition d'états locaux, chacun n'étant accessible que par un seul processus logique et chaque processus ne pouvant accéder qu'à un seul état local.

Il est possible de construire une partition orientée entité (chaque partie de la partition est contrôlé par un et un seul processus logique), ou orientée espace (plusieurs parties de la partition sont regroupées sous le contrôle d'un processus logique), ou les deux [Monsef 1997], [Fishwick 1995]. Il apparaît que l'optimisation de cette partition est fonction des performances du matériel, de

l'approche de gestion du temps simulé, et de la taille du modèle réparti [Nandy & Loucks 1993], [Schöf 1995], [Solcany et al. 1995], [Langlois & Phipps 1997].

Par ailleurs, l'exclusion de données et de variables globales peut générer de sérieux problèmes de programmation, selon l'application étudiée. Dans les applications pour lesquelles les aspects spatiaux sont très importants, certaines informations, notamment celles qui sont relatives à l'environnement, doivent être partagées par plusieurs processus. Deux approches peuvent être utilisées :

- Inclure la (ou les) variable(s) globale(s) dans un processus. Cette émulation de mémoire partagée, par un processus, présente le grave inconvénient de constituer un nœud d'étranglement pour tous les autres processus qui doivent lui adresser une requête de lecture ou de mise à jour de la (ou des) variable(s) globale(s) qu'il gère. De plus, comme les requêtes peuvent provenir de processus opérant à des dates de simulation différentes, la gestion chronologique de ces requêtes est un problème supplémentaire de synchronisation qui s'ajoute à celui existant déjà entre les processus logiques ordinaires.
- Dupliquer la (ou les) variable(s) globale(s) dans chaque processus. Cette solution, adoptée dans le cadre de la Simulation Distribuée Interactive (DIS, pour Distributed Interactive Simulation), i.e., incluant l'homme dans la boucle de simulation, nécessite la mise en place d'un protocole particulier pour assurer la cohérence entre toutes les copies de cette (ou ces) variable(s) globale(s) en cas de modification de valeur.

L'architecture générique de référence finalement retenue est un réseau de N processus logiques, généralement notés LP_0, LP_1, …, LP_{N-1}, chacun chargé de simuler une partie du système réel (Figure 35). Ces processus communiquent par échange de messages (porteurs d'événements), pour exprimer les interactions entre parties simulées du système. Chaque processus LP_i possède une horloge locale $CLOCK_i$ qui indique la date courante locale de simulation de ce processus. Cette date indique la date d'exécution du dernier événement traité par le processus (en dehors de la toute première date qui est la date d'initialisation du processus). Le réseau comporte un ensemble de liens logiques de processus à processus. Chaque lien logique matérialise la possibilité d'envoi de message d'un processus vers un autre, du canal de sortie de l'expéditeur au canal d'entrée du récepteur. Les messages reçus sur un canal d'entrée sont stockés dans une file d'attente d'entrée gérée en FIFO (First In First Out, dit encore PAPS pour

Premier Arrivé Premier servi). Il est imposé à chaque lien que les envois de message respectent un ordre croissant sur les estampilles des messages envoyés. Ceci garantit le fait qu'un message reçu sur un canal d'entrée est une borne inférieure de tous les messages qui pourront parvenir ultérieurement sur ce canal. Chaque file d'entrée est dotée d'une horloge qui indique le temps de simulation du canal correspondant. Ce temps, appelé temps canal, est l'estampille du premier message de la file si cette dernière n'est pas vide. Sinon, ce temps canal contient l'estampille du dernier message ayant été reçu par (et retiré de) la file. Le processus choisit toujours le canal dont le temps canal est minimal. Si la file d'entrée correspondante n'est pas vide, le premier message de cette file est enlevé de la file et consommé par le processus. Ce protocole garantit le respect de la contrainte de causalité locale.

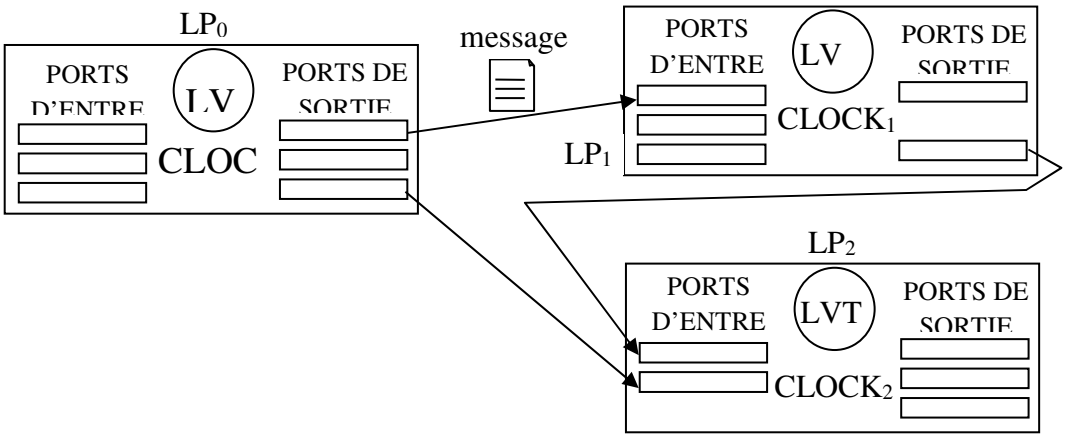


Figure 35. Architecture générique de simulation répartie

Dans ce modèle standard, si chaque processus respecte la contrainte de causalité locale, i.e. traite ses événements selon l'ordre chronologique croissant de leurs dates d'occurrence, alors le principe de causalité est garanti pour le modèle entier. Le respect de la contrainte de causalité locale est une condition suffisante de respect du principe de causalité globale, mais pas nécessaire, en raison de l'inexistence de liens de causalité entre certains événements se produisant à des dates différentes au sein d'un même processus. En effet, deux événements indépendants peuvent être exécutés dans un ordre chronologique non forcément croissant sans qu'une erreur de causalité apparaisse. Certes, pour des événements sans lien de causalité, l'ordre de traitement n'est pas important, cependant, le graphe de dépendance entre événements n'est pas connu à priori et sa nature dynamique et fortement complexe constitue la base de la problématique de la simulation répartie.

Dans quelle mesure deux événements peuvent-ils être exécutés de manière concurrente ? Les stratégies pour répondre à cette question se répartissent en deux classes :

- les stratégies pessimistes dites conservatives, qui n'autorise pas d'erreur de causalité [Misra 1986], et
- les stratégies optimistes qui les autorisent mais qui sont capables de les détecter et de les corriger [Jefferson 1985]. Cette correction utilise un principe de retour au passé de la simulation.

Des survols panoramiques de variantes pour ces classes se trouvent dans [Fujimoto 2000], [Ingels & Raynal 1990]. L'intérêt visé est d'optimiser les performances de la simulation (vitesse d'exécution accrue par parallélisation, ou taille de modèle accrue par agrégation de processeurs, ou les deux). Le verrou ici est souvent le passage systématique au code pour les techniques réparties (dans une certaine mesure, l'hétérogénéité des paradigmes de programmation et celle des langages support participent également à cette difficulté).

3.1.2.ii. Stratégies pessimistes

Dans le cas pessimiste, un LP ne traite un événement dont la date d'occurrence (i.e., l'estampille) est t, que lorsqu'il est sûr de ne pas recevoir des autres LPs de message d'estampille $t' < t$, ce afin de respecter la contrainte de causalité locale. Cette surveillance impose à un LP de s'assurer qu'aucun traitement d'événement ne sera fait tant qu'au moins une de ces files est vide. Ce LP se bloque donc, si une de ses files de réception de message est vide, car il ignore alors la date du prochain message qui arriverait dans cette file et ne peut donc garantir que ce prochain message sera d'estampille supérieure ou inférieure à celui des autres messages en attente dans ses autres files. Le problème de cette approche est la possibilité d'une situation d'inter-blocage des LPs lorsque ces derniers sont reliés en boucle et s'influencent en chaîne fermée. Si l'un des LPs se bloque, il entraîne le blocage de tous les autres. La Figure 36 illustre cette situation d'inter-blocage. Par exemple, LP_1 après réception d'un message de LP_0, émet un message soit vers LP_2 soit vers LP_3 en fonction d'une certaine règle. Si au cours de la simulation aucun message n'est envoyé vers LP_2, la file de réception de LP_4 nourrie par LP_2 sera en famine et finira par se vider, et LP_4 ne pourra plus faire avancer son temps de simulation. Il sera donc bloqué indéfiniment en

attendant l'arrivée d'un message sur la file venant de LP_2, bien qu'il ait des messages en attente en provenance de LP_3. Ainsi LP_0 à son tour sera aussi bloqué en attendant l'arrivée de message provenant de LP_4. Ceci finira par bloquer LP_1, qui à son tour affamera LP_3 et LP_2. Il faut noter que cette situation d'inter-blocage ne correspond pas à un inter-blocage du système réel, mais résulte de la stratégie mise en œuvre pour garantir le principe de causalité.

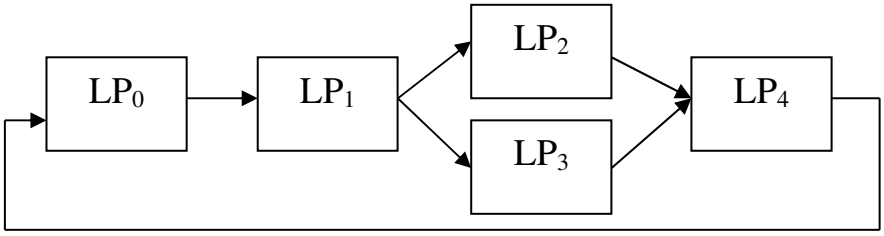


Figure 36. *Situation pouvant conduire à un inter-blocage*

Pour faire face à cette situation d'inter-blocage, la littérature propose deux classes de solutions : (1) la prévention de l'inter blocage, ou (2) la détection et la guérison de l'inter blocage. La plus emblématique des solutions de la première classe est celle dite à messages nuls [Chandy & Misra 1979], [Bryant 1977]. Celle des solutions de la seconde classe est celle dite du plus petit LVT [Chandy & Misra 1981].

Approche à messages nuls

La solution avec message nul a été proposée dans [Chandy & Misra 1979], [Bryant 1977]. L'idée de base de cette solution est de faire émettre par les processus, en plus des messages normaux de la simulation, des messages de contrôle appelés messages nuls. Un message nul n'a d'autre contenu que son estampille (date d'occurrence). Lorsqu'un LP transmet un message daté sur une de ses sorties, il transmet également un message nul de même date sur toutes ses autres sorties. Le traitement de ces messages par les LPs récepteurs n'implique aucun autre calcul que celui permettant de faire évoluer le LVT. En effet, un LP inspecte chacune de ses files de réception et prélève le message d'estampille minimal dans l'une d'elles (les messages étant rangés par estampille croissante dans les files, seul le message en tête de chaque file est inspecté). Si ce message n'est pas nul, il engendre un traitement conforme aux règles de la simulation, y compris l'avancée du LVT à la date indiquée par l'estampille de ce message ; si ce message est nul, la seule conséquence est l'avancée du LVT à la valeur indiquée par l'estampille du message nul. Sur l'exemple précédent (Figure 36),

LP_1 émet un message nul vers LP_2 (respectivement LP_3) à chaque fois qu'il émet un message vers LP_3 (respectivement LP_2). Ainsi LP_4 recevra régulièrement des messages sur chacune de ses files d'entrée et pourra donc faire progresser son temps de simulation. Cette solution ne suffit pas à prévenir l'inter-blocage dans tous les cas. Prenons l'exemple de la Figure 37 où une boucle de rétroaction directe existe entre LP_3 et LP_2. Au temps 5, LP_3 émet le message m1 vers LP_4, il émet donc également un message nul de même date vers LP_2. Au temps 10, LP_1 émet le message m2 vers LP_2. Les deux files d'entrée de LP_2 contenant, chacune un message, LP_2 peut donc prélever (NULL, 5) et faire avancer son LVT à 5. Mais la consommation du message nul ne provoque l'émission d'aucun message vers LP_3, et le système se bloque. Pour résoudre ce problème il faut connaitre une information supplémentaire appelée *Lookahead*. Dans la situation de la Figure 37 par exemple, LP_2 sait qu'il ne recevra aucun message ayant une date strictement inférieure à 5. Il peut en déduire qu'il n'enverra pas de message avant la date $5+\varepsilon$ ($\varepsilon > 0$), où ε représente la durée minimale de traitement d'un événement par LP_2. Pour LP_2, cette valeur ε constitue une certaine visibilité sur le futur qui est appelée son Lookahead. Ainsi, de manière générale, dans le cas où un LP n'a aucune sortie à faire, il envoie un message nul daté avec son Lookahead, i.e., avec la date au plus tôt du prochain envoi de message normal. Le Lookahead est fortement dépendant de la nature du modèle à simuler.

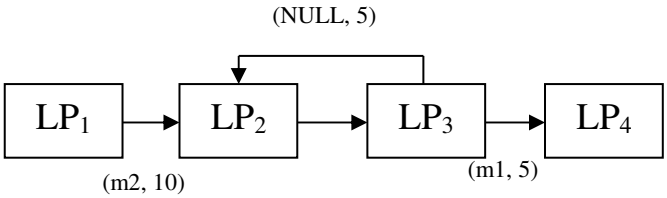


Figure 37. Inter blocage dans l'approche à messages nuls

Approche du plus petit LVT

Cette stratégie a été introduite dans [Chandy & Misra 1981]. Dans ce cas la simulation consiste à répéter la séquence suivante : (1) Simuler jusqu'à l'inter-blocage ; (2) Détecter l'inter-blocage ; et (3) Guérir l'inter-blocage en relançant l'exécution d'un ou plusieurs LPs. Dans la première phase, les LP n'émettent que des messages de simulation. Ils font progresser la simulation en itérant sur la consommation du message reçu sur la file d'entrée ayant la plus petite date. Quand la file ayant la plus petite date est vide le LP se bloque. Un processus de contrôle détecte alors l'inter-blocage grâce un algorithme (comme celui de

[Dijkstra & Scholten 1980] ou celui de [Chandy et al.1983]). Quand le processus de contrôle a détecté l'inter-blocage, il demande aux LPs de démarrer un calcul réparti qui permet de déterminer le, ou les LPs pouvant redémarrer sans introduire de violation du principe de causalité. Il faut noter qu'il existe toujours au moins un LP pouvant reprendre la simulation. En effet, parmi tous les messages en attente dans les files d'attente de tous les LPs, celui qui a la plus petite estampille T peut être consommé par son destinataire car plus rien ne peut modifier l'état du système avant la date T. L'algorithme de guérison consiste alors à calculer cette valeur T. Quelques améliorations de cet algorithme de guérison ont été développées (voir [Chandy & Misra 1981] et [Fujimoto 1990]).

3.1.2.iii. Stratégies optimistes

Alors que les approches pessimistes évitent toute violation de la contrainte de causalité locale, les approches optimistes l'autorisent et prévoient un mécanisme de réparation. Ainsi, chaque LP traite les messages au fil de leur réception, sans se préoccuper de savoir si toutes ses files de réception sont non vides. Il est donc possible qu'il reçoive ultérieurement, sur une de ses files vides, un message dont l'estampille sera inférieure à son LVT, provoquant ainsi une erreur de causalité. Ce message est appelé *trainard*, et le LP doit alors revenir à un passé antérieur à l'estampille du trainard pour pouvoir le traiter correctement. Ce retour au passé est appelé *Roll back*.

Beaucoup de mécanismes optimistes ont été proposés dans la littérature, mais le *Time Warp* [Jefferson 1985] en est le fondateur. Lorsqu'un LP traite un message dont l'estampille T est inférieure au LVT, il annule tous les messages qu'il a déjà expédiés avec une estampille supérieure à T (appelons-les « mauvais messages »), et restaure le LP dans l'état antérieur à T le plus proche du LVT. Pour cela, il doit gérer un historique de ses états successifs, y compris ses files de message. L'annulation des mauvais messages consiste à demander aux LPs récepteurs de ces mauvais messages, de les annuler de leurs files de messages reçus. Pour ce faire, le LP envoie des anti-messages, qui possèdent exactement les mêmes caractéristiques que les mauvais messages, à l'exception d'un drapeau signalant leur caractère d'anti-message. Un anti-message lorsqu'il est reçu par un LP, provoque l'annulation du message normal correspondant si ce dernier est encore présent dans la file des messages reçus, sinon provoque à son

tour un Rollback de ce LP. Cette stratégie nécessite donc de conserver les états successifs des variables de chaque LP, y compris la liste de tous les messages émis et reçus. En fait, le Rollback peut se résumer ainsi : (1) Défaire une action locale consiste simplement à revenir à un ancien état des variables locales du LP, ancien état que l'on aura sauvegardé. (2) Une émission de message sera défaite en émettant un anti-message vers le même destinataire, avec la même estampille que le message initial. La réception d'un anti-message provoque chez le récepteur, soit l'élimination du message initial s'il n'avait pas encore été consommé, soit un Rollback jusqu'à la date correspondant à l'estampille de cet anti-message. (3) Les actions définitives (i.e., celles sur lesquelles on ne peut pas revenir, comme les entrées/sorties), sont différées jusqu'à ce que la simulation ait progressé au point où on est sûr de ne pas avoir à les défaire. L'algorithme du Time Warp est composé de deux parties :

(1) un mécanisme de contrôle local (implémenté sur chaque LP), et
(2) un mécanisme de contrôle global.

Contrôle local

Les opérations de Rollback sont déclenchées par la réception soit d'un *traînard* ou d'un anti-message. A la réception d'un *trainard*, les actions suivantes sont effectuées : (1) insertion du traînard dans la file d'entrée ; (2) annulation des évènements d'estampille supérieure à celle du traînard ; (3) restauration de l'état courant du LP à la date antérieure la plus proche de celle du *traînard* ; (4) suppression de tous les états sauvegardés dans l'historique des états successifs du LP dont les dates sont supérieures à celle du *traînard* ; (5) remise du LVT à la date de l'état restauré ; (6) envoi des anti-messages pour tous les messages envoyés avec une estampille supérieure ou égale à celle du *traînard*.

A la réception d'un anti-message, trois cas de figure se présentent :

- Si le message correspondant a été traité alors il est annulé, et le LP effectue un Rollback à la date de l'événement qui précède ce message.
- Si le message correspondant n'a pas encore été traité alors le message et l'anti-message s'annulent tout simplement.
- Si le message correspondant n'était pas encore arrivé au destinataire, alors l'anti-message est placé dans la file de réception des messages, dans l'attente de la réception et de la neutralisation du message.

Contrôle global

Le mécanisme de contrôle global repose sur la notion de Temps Virtuel Global (GVT pour Global Virtual Time). A tout instant de la simulation, le GVT représente une borne inférieure sur les dates de Rollback possibles. Ainsi, toutes les actions effectuées à une date simulée inférieure au GVT ne seront jamais défaites. Les sauvegardes des états ou messages ayant une date inférieure au GVT peuvent être oubliées (libérant ainsi de l'espace mémoire, action appelée *collecte de fossiles*) ; de même, les entrées/sorties dont la date est antérieure au GVT peuvent être réellement réalisées en toute confiance. Certaines variantes du Time Warp calculent le GVT de manière périodique, d'autres seulement lorsque la simulation manque de mémoire.

3.1.2.iv. Stratégies réparties hybrides

Les techniques de simulation répartie sont certes arrivées à maturité, mais leurs performances (vitesse d'exécution, charge des échanges de messages…) sont loin d'être maîtrisées, et ce en dépit des nombreux bancs de tests réalisés. Plusieurs critères affectent ces performances, dont surtout la structure des modèles de représentation des systèmes concernés. Des exemples très significatifs d'étude de performances des approches réparties sont [Fujimoto 1988] et [Lubachevsky & Weiss 1990], où des tests antérieurs sont également référencés et leurs résultats synthétisés. En pareille circonstance, d'autres domaines auraient exploré l'hybridation de ces stratégies pour combiner les bénéfices de chaque catégorie et en compenser les inconvénients. Ceci reste, ici, une possible voie de recherche non explorée.

Néanmoins, d'importants efforts ont été fournis pour définir des environnements de simulation répartie, intégrant des langages de spécification et qui visent à affranchir le concepteur de l'environnement hôte. Un état de l'art des outils, langages et environnements est donné dans [Low et al. 1999]. Les travaux sur le network-centric simulation [Colvin & Beaumariage 1998] ont des préoccupations similaires, l'intérêt étant d'agréger la puissance de plusieurs hôtes en une seule machine.

Par ailleurs, optimiser le compromis entre le nombre d'unités autonomes et la charge en flux d'échanges s'apparente à un problème de partition dans un graphe [Stopper & Böszörmenyi 1995], problème réputé NP-complet [Garey et al. 1976]. Maximiser le nombre d'unités répond au double besoin d'augmenter, d'une part, le potentiel de parallélisme et d'amoindrir, d'autre part, la complexité par réductionnisme ; alors que minimiser les flux d'échanges (antagoniste) vise à réduire les coûts de dépendance inter unités.

3.1.3. Sémantiques opérationnelles multiples

Nous avons introduit en Chapitre 2 la notion de modélisation multi-analyse, comme moyen d'utiliser un modèle unique pour à la fois simuler, analyser formellement et émuler un système. Ce modèle pourrait également être un objet algébrique manipulable de manière symbolique. Dans cette veine, nous avons introduit la syntaxe abstraite de HiLLS, langage de haut niveau destiné à servir cet objectif. Deux sémantiques seront définies ici pour HiLLS :

- une sémantique opérationnelle pour simulation (i.e., sur l'hypothèse d'usage d'un temps virtuel), et
- une sémantique opérationnelle pour émulation (i.e., sur l'hypothèse d'usage du temps mural).

L'idée globale est de toujours tirer profit des cadres théoriques existants, ainsi que des outils développés pour soutenir ces cadres. Ainsi, au lieu de définir de nouvelles stratégies de simulation, nous établissons des règles de correspondance entre HiLLS et DEVS, permettant d'obtenir de toute spécification HiLLS son équivalent DEVS (il est alors possible d'utiliser l'un des nombreux environnements de simulation DEVS pour exécuter ce modèle). De manière similaire, nous définissons des règles permettant d'obtenir la description UML de l'émulateur du système (il est alors possible de générer son code et de l'exécuter directement).

3.1.3.i. Sémantique pour simulation

Ici également, nous adoptons dans le cadre de cet ouvrage, une démarche plus intuitive que formelle. En effet, bien que les règles de correspondance entre

HiLLS et DEVS soient établies de manière rigoureuse et formalisée, nous utilisons la Table 4 pour énoncer les principes guidant cette correspondance. L'automatisation de la génération de modèle DEVS à partir de modèle HiLLS utilise les techniques de transformation de modèle exposées précédemment.

Table 4. Correspondance entre HiLLS et DEVS

Concepts DEVS	***Concept HiLLS***
modèle atomique	modèle HiLLS sans composant système (i.e., avec des ports d'entrée/sortie, mais des composants objet peuvent exister)
X	ensemble des ports d'entrée
Y	ensemble des ports de sortie
S	ensemble des configurations
δ_{int}	ensemble des transitions internes (lorsque des nœuds de décision existent, chacun correspond à 2 transitions)
δ_{ext}	ensemble des transitions externes (chaque nœud de décision correspond à 2 transitions)
δ_{conf}	ensemble des transitions confluentes (chaque nœud de décision correspond à 2 transitions)
λ	ensemble des schémas définissant les *outputEvents*
ta	ensemble des durées de vie des configurations
modèle couplé	modèle HiLLS avec des composants système (et éventuellement des composants objet)
X, Y	ports d'entrée/sortie
D	ensemble des noms des composants système
$\{M_d / d \in D\}$	ensemble des spécifications des composants système
EIC	ensemble des prédicats des configurations du modèle liant ses ports d'entrée à ceux de ses composants système
EOC	ensemble des prédicats des configurations du modèle liant ses ports de sortie à ceux de ses composants système
IC	ensemble des prédicats des configurations du modèle liant entre eux les ports d'entrée et les ports de sortie de ses composants système

Il existe plusieurs environnements de simulation DEVS, chacun ayant un format spécifique pour la description des modèles. Comme implémentations majeures DEVS, citons :

- ADEVS [Nutaro 2010] qui fournit une bibliothèque C++ basée sur PDEVS et dynDEVS (extension pour les systèmes à structure variable).
- DEVS-C++ [Zeigler et al. 1996] et DEVSJava [Sarjoughian & Zeigler 1998] qui sont des simulateurs basés sur Classic DEVS, et implémentés respectivement en C++ et Java.
- GALATEA [Davila & Uzcategui 2000] qui est une architecture orientée objet pour la M&S des systèmes multi-agents avec DEVS.
- JAMES [Uhrmacher 2001] qui est un outil Java de M&S multi-agents basé sur PDEVS.
- PyDEVS [Delara & Vangheluwe 2002] qui est une implémentation de DEVS en Python. Par ailleurs, ATOM3 est un environnement de méta-modélisation [Delara & Vangheluwe 2002], dont dérive ATOM3-DEVS, outil pour la modélisation multi-paradigme, la construction de modèles DEVS et la génération de code Python pour le simulateur PyDEVS.
- PowerDEVS [Kofman et al. 2003] qui est un outil de M&S DEVS des systèmes hybrides. Il est implémenté en C++.
- CD++ [Wainer 2002] qui est un outil implémentant en C++, CDEVS, PDEVS, et Cell-DEVS (extension pour les automates cellulaires).
- SimStudio [Traore 2008] qui est notre Framework DEVS.
- Bien d'autres implémentations existent (tels, SimBeans [Praehofer et al. 1999], SmallDEVS [Janousek et al. 2006], JDEVS [Filippi et al. 2002], DEVS-Scheme [Zeigler & Kim 1993], etc.).

La Table 4 donne donc un schéma générique de mise en correspondance, qui doit être explicitée en détail dans le cas de chaque environnement. Cette table dit ce qu'il faut faire, mais pas comment il faut le faire. La Figure 38, elle, schématise la manière dont, dans SimStudio, le modèle DEVS de l'exemple du BVM est extrait de sa spécification HiLLS :

- Les correspondances en rouge donnent les transitions, avec une distinction entre les internes, les externes et les confluentes.
- Les correspondances en bleu donnent les états dans DEVS.
- Les correspondances en vert donnent les sorties effectuées par le modèle.
- Les correspondances en jaune donnent l'avancement du temps.
- Les correspondances en orange donnent les interfaces du modèle.
- Il faut noter que les activités sont ignorées dans DEVS (sauf les opérations de modification des valeurs des variables d'état).

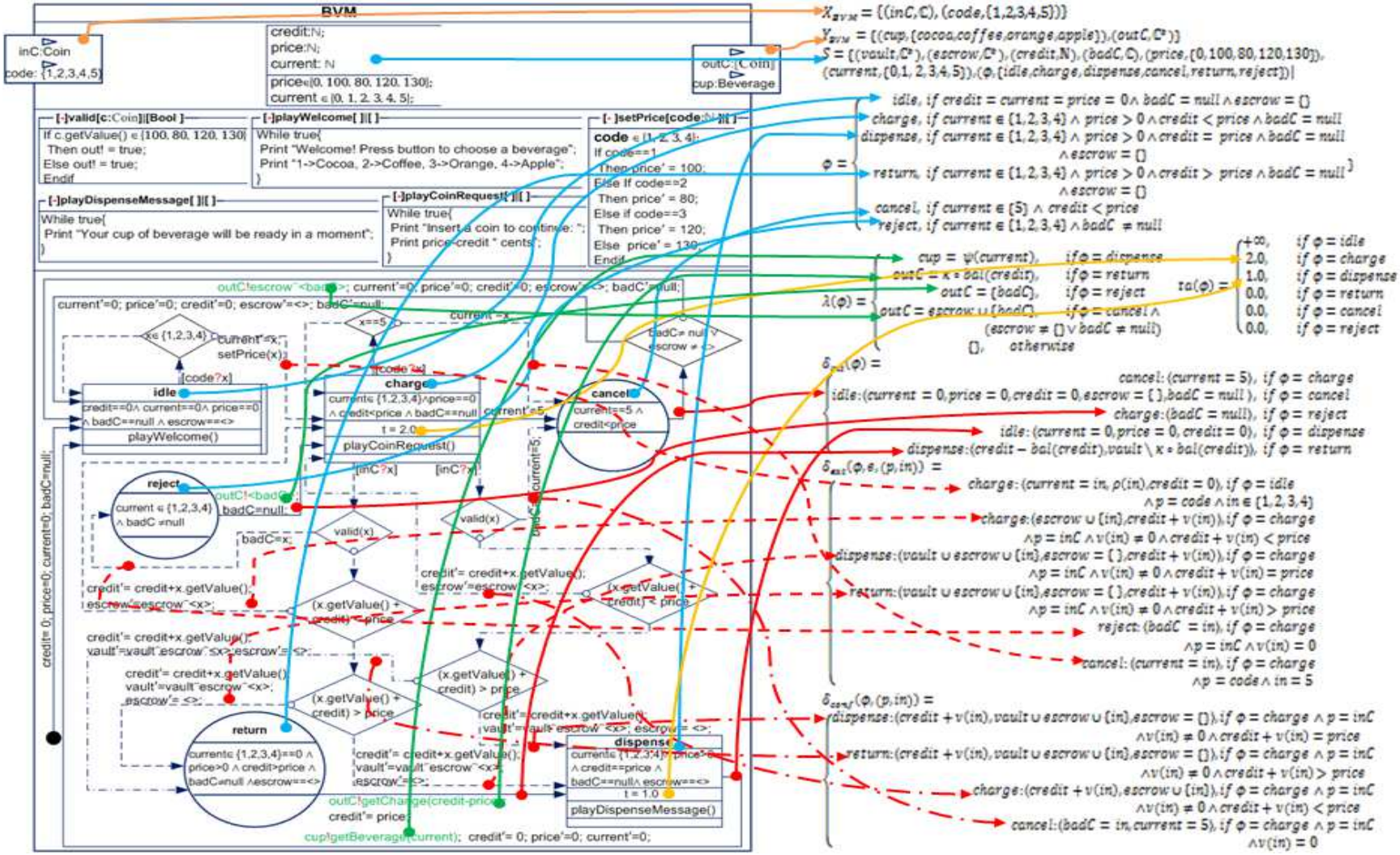


Figure 38. Correspondance entre modèle HiLLS et modèle DEVS du BVM

3.1.3.ii. Sémantique pour émulation

Afin que les modèles HiLLS soient directement exécutable comme spécifications de systèmes logiciels, une sémantique opérationnelle a été définie, comme décrite par le méta-modèle fourni en Figure 39. Le protocole d'émulation associé est donné par la Figure 40. L'émulateur prend ses informations du modèle (configurations, transitions, avancée du temps, etc.) et procède de manière similaire à DEVS, à la différence que le temps n'est plus virtuel mais est celui de l'horloge murale.

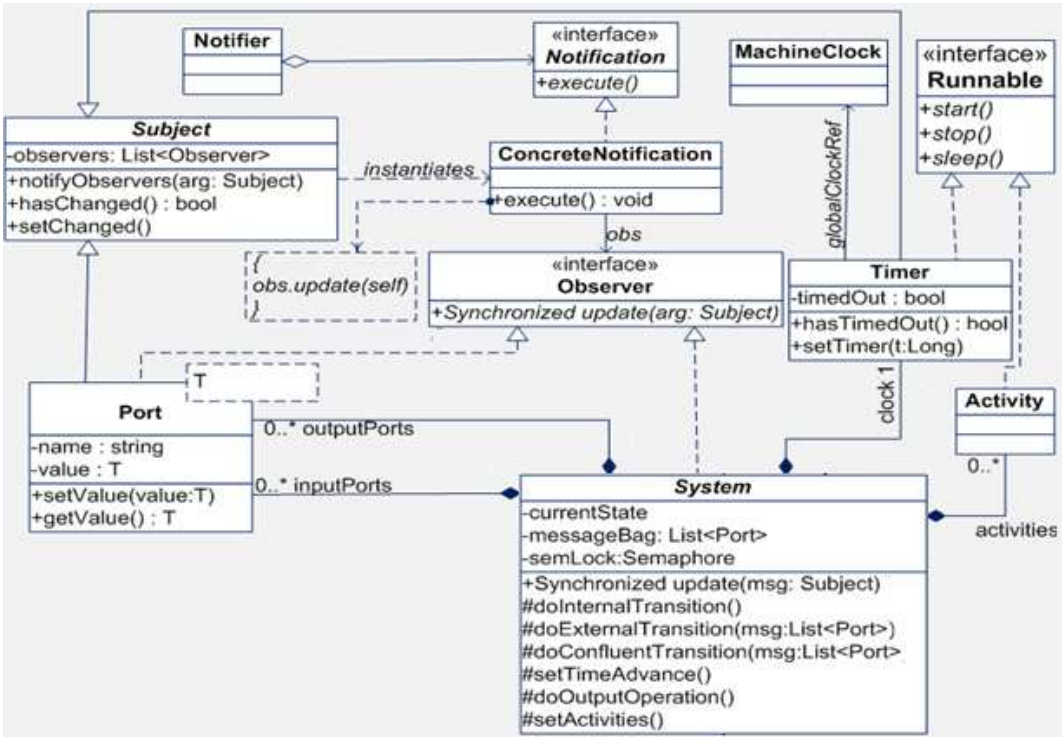


Figure 39. Méta-modèle d'émulation de modèle HiLLS

Cette sémantique s'appuie sur l'utilisation du design pattern Observer [Gamma et al. 1995]. Ce motif de conception permet de définir un sujet et des observateurs de ce sujet, de sorte que toute modification sur le sujet entraîne une notification de ses observateurs. Cette idée est utilisée ici à trois niveaux :

(1) le système est un observateur de ses ports d'entrée (ainsi, il est immédiatement notifié de toute nouvelle donnée arrivant sur n'importe lequel de ses ports d'entrée) ; c'est pourquoi, la classe *System* implémente l'interface *Observer* et la classe *Port* dérive de la classe *Subject* ;

(2) tout port d'entrée (du système ou de ses composants système) est un observateur des autres ports qui doivent l'approvisionner (ainsi, il est notifié de la présence de donnée sur ces ports) ; c'est pourquoi la classe *Port* implémente également l'interface *Observer* ;
(3) le système est un observateur de son horloge (ainsi, ce dernier lui notifie l'avancée du temps, permettant ainsi de gérer les transitions internes) ; c'est pourquoi la classe *Timer* dérive de la classe *Subject*.

Par ailleurs, les activités à réaliser, tout comme l'horloge, sont des tâches autonomes concurrentes. C'est pourquoi les classes *Timer* et *Activity* implémentent l'interface *Runnable*. Le protocole de la Figure 40 fait usage du mécanisme de sémaphore pour s'assurer que toutes les notifications faites de manière concurrente à une entité à un moment donné, sont toutes prises en compte par l'entité notifiée.

```
1: function UPDATE(Subject msg)                          ▷ a message msg is received
2:     while semLock_waitingQueue ≠ φ do                 ▷ loop to treat all concurrent notifications
3:         SemaphoreWait(semLock, 1);                    ▷ one notification thread has access at a time
4:         if msg = portMessage then
5:             messageBag ← messageBag ∪ {msg};          ▷ add only portMessages to message bag
6:         end if
7:         SemaphoreSignalAll(semLock);                  ▷ release lock and notify all waiting threads
8:     end while          ▷ all concurrent messages have been saved, next is system's reaction
9:     if messageBag = φ ∧ clock.hasTimedOut then        ▷ only a clockMessage was received
10:        doOutputOperation();                          ▷ send output events if any
11:        doInernalTransition();                        ▷ fire internal state transition operation
12:    end if
13:    if messageBag ≠ φ ∧ ¬clock.hasTimedOut then       ▷ only portMessage(s) received
14:        doExternalTransition(messageBag);             ▷ fire external state transition operation
15:    end if
16:    if messageBag ≠ φ ∧ clock.hasTimedOut then  ▷ clockMessage & portMessage(s) received
17:        doOutputOperation();                          ▷ send output events if any
18:        doConfluentTransition(messageBag);            ▷ fire confluent state transition operation
19:    end if
20:    flush messageBag;                                 ▷ clear the content of message bag
21:    setTimeAdvance();                  ▷ compute timeAdance of new state & set clock timer
22:    runActivity();                                    ▷ start execution of state's activity if any
23: end function
```

Figure 40. Protocole d'émulation

A la différence de DEVS, HiLLS ne différentie pas modèles atomiques et modèles couplés, dans leur traitement. Ce qui est un modèle couplé DEVS, est vu, s'il n'est pas à structure variable, comme un modèle HiLLS possédant une

unique configuration qui décrit cette structure ; s'il est à structure variable, cette variabilité est exprimée par son diagramme de transition de configurations.

3.2. Connecteur de simulation

Le souhait d'agréger des sources de simulation distantes accompagne le glissement progressif des infrastructures matérielles monolithiques vers les solutions réparties grande échelle (Web, grille de calcul, environnements mobiles et embarqués…). La dimension algorithmique de la question s'en trouve doublée d'une dimension technologique non moins importante d'interfaçage logiciel. En effet, d'une part, comme vu dans précédemment, les communications entre automates distants, pour être prises en compte correctement par les stratégies de simulation adoptées dans les différents codes, nécessitent l'introduction de mécanismes additionnels de synchronisation. D'autre part, ces communications induisent également un problème de génie logiciel à travers la nécessité de mise en place d'un protocole réseau, qui est forcément assujetti aux types d'architectures matérielles en jeu.

Dans ce dernier cas (qui nous intéresse ici), le gain de performances n'est pas la priorité, mais plutôt le besoin scientifique de pouvoir utiliser des modèles de simulation existants et distants, dans des simulations de plus large envergure. Les technologies matures de Génie Logiciel s'imposent immédiatement à l'esprit pour une application au cas de la M&S : CORBA, DCOM, EJB, etc. [Sommerville 2006], tout au moins, pour réaliser des solutions ad-hoc. La barrière de complexité est due à la difficulté et au coût d'implémentations répétées de ces solutions. Plusieurs voies technologiques alternatives, que les avancées du Génie Logiciel produisent, ont été mises à contribution (voir par exemple [Page et al. 1997], [Cho 2001], [Boukerche & Zhang 2008]). De nouvelles directions émergent également, comme celle des services Web. Les premiers travaux sur la simulation par le Web [Fishwick 1996], [Davis et al. 1999] sont restés timides, jusqu'à l'arrivée récente de SOAP (Service Oriented Architecture Protocol). Une forte tendance en cours est à l'utilisation de ce nouveau protocole d'échange pour rendre interopérables des simulateurs hétérogènes, dans le contexte de DEVS [Mittal et al. 2007].

L'état de l'art fait apparaître deux démarches de fédération de codes de simulation hétérogènes en langage d'implémentation et/ou plateforme :

- Fédérer des automates hétérogènes par une technologie d'interopérabilité. Il s'agit de construire une architecture de connexion qui permette à ces automates d'échanger des messages. Chaque automate de simulation est responsable de la garantie de sa simulation locale, ainsi que des synchronisations nécessaires pour prendre en compte correctement les messages reçus des autres automates de simulation. L'architecture d'interopérabilité est garante de l'acheminement des messages et de la standardisation des formats d'échange.
- Réaliser un automate d'interopérabilité, permettant de fédérer des automates de simulation hétérogènes. Ici, les synchronisations nécessaires pour garantir la justesse de la simulation globale sont réalisées dans l'automate d'interopérabilité alors que chaque automate de simulation garantit la justesse de sa simulation locale.

Ces deux démarches se retrouvent dans les nombreux protocoles d'interopérabilité qui ont été développés par la communauté scientifique, et qui peuvent être classés entre :

- Point-à-Point (ou P2P pour Peer-to-Peer),
- Client/Serveur, et
- Producteur/Consommateur (forme plus large du Client/Serveur).

3.2.1. Connecteurs Point à Point

Les technologies les plus visibles dans ce domaine sont :

- SIMNET (SIMulator NETworking) qui a été développé et utilisé par l'armée américaine [Miller & Thorpe 1983]. Son développement a commencé dans les années 80 et a été utilisé pour l'entraînement, avant que les standards successeurs voient le jour dans les années 1990.
- DIS (Distributed Interactive Simulation) qui est un standard développé à l'Université de Floride pour améliorer les applications tournantes de SimNET. Ses principales implémentations sont OpenDIS, KDIS, DIS 4.0 et Mik (commerciale).

3.2.2. Connecteurs Client/Serveur

Dans ce domaine, les efforts les plus notables sont :

- ALSP (Aggregate Level Simulation Protocol) qui a été implémenté avec succès dans les jeux de combats en ligne [Fujimoto 2000].
- CORBA, développée par l'OMG (Object Management Group), et qui vise à mettre en place une norme d'architectures distribuées ouvertes (et pas pour la simulation répartie uniquement), permettant de faire communiquer des applications en environnement hétérogène (i.e., plusieurs systèmes d'exploitation et plusieurs langages de programmation) [OMG 2000].

3.2.3. Connecteurs Producteur/Consommateur

Les efforts les plus emblématiques ici, sont :

- HLA (High Level Architecture) [Dahman et al. 1998], qui a été développé par le bureau de Modélisation et de Simulation en Défense (DMSO) du Département américain de la Défense (DoD). HLA est un standard défini pour permettre à plusieurs programmes de simulation de constituer par fédération, un plus gros modèle dédié à une application englobante. Dans le protocole HLA, chaque simulateur participant, dit fédéré, interagit avec d'autres fédérés au sein d'une collection de simulateurs dite fédération. Les fédérés communiquent entre eux au sein de la fédération à travers un middleware appelé RTI (Run Time Infrastructure) [Fujimoto 2000]. Un fédéré peut être un simple modèle ou un modèle plus complexe ; il peut s'exécuter sur une ou plusieurs machines ; il peut être autre chose qu'un modèle de simulation pure (collecte de données, interrogation, réception, visualisation, participation humaine à une simulation, etc.). Chaque fédéré possède un seul point de connexion au RTI, et ce point constitue le seul moyen d'interaction avec les autres fédérés. Par ce point, le RTI offre à chaque fédéré une interface appelée RTIambassador, et chaque fédéré offre au RTI une interface appelée FederateAmbassador. La manière d'implanter ces interface est libre ; en l'occurrence, les deux interfaces peuvent être implantées au sein de chaque fédéré ; bien entendu, il est également possible que chaque fédéré implante l'interface qu'il offre, et que les interfaces offertes par le RTI soient implantées dans le RTI. L'implantation du RTI est également libre. Quelques implémentations de

HLA sont : YaRTI (implémenté en ADA95 dans le domaine de l'Aerospatiale), Portico (implémentée en C++ et Java dans le domaine de la Défense en Australie), GERTICO (implémenté en CORBA dans le domaine de la Défense en Allemagne), CERTI (implémenté en C++ dans le domaine de la Défense en France), RTI-NG (implémenté en C, C++, JAVA, CORBA... dans le domaine de la Défense aux USA), EODISP (implémenté en Java dans le domaine spatial en Europe), Calytrix (pour le jeu en réseau), etc.

- DDS (Data Distribution Service), standard spécifié par l'OMG dont le rôle est de proposer une technologie évoluée d'échanges de données sur des réseaux allant des systèmes embarqués vers les réseaux à grande distance. DDS fournit de nouveaux aspects non inclus dans HLA. JacORB-DDS, Poccapsule-DDS, RTI-DDS, CoreDX, et OpenSplite DDS en sont quelques implémentations.
- SOA (Service Oriented Architecture), qui est un protocole orienté services permettant de faire communiquer deux sous-systèmes/applications de manière indépendante des plateformes hôtes utilisées. Elle offre une approche d'interopérabilité plus flexible car elle utilise des protocoles de communication standards ouverts comme XML, HTTP et SOAP pour le mode communication entre le producteur et le consommateur.

3.2.4. Machine virtuelle de simulation

Le concept initial de machine virtuelle (MV) désigne la réunion sur une seule architecture hôte de plusieurs environnements d'exécution, dont chacun émule l'hôte (via l'hyperviseur, couche logicielle qui s'interface entre le système d'exploitation de l'hôte et les différents environnements). Puis, progressivement, dans l'entendement commun, l'hyperviseur est devenu lui-même la MV (i.e., un émulateur de système d'exploitation tiers sur un système d'exploitation initial différent). Plus récemment, le terme de MV a été utilisé pour désigner un fédérateur de ressources de calcul qui se présente comme une unique ressource de calcul.

Dans son sens le plus commun, la MV désigne une interface logicielle qui isole l'application utilisée par l'utilisateur des spécificités de l'architecture et de l'environnement hôte. Cette indirection permet de rendre une application

disponible sur un grand nombre d'hôtes sans les contraintes habituelles à l'élaboration d'un logiciel portable tournant directement sur l'hôte.

La notion de MV pour la M&S est l'application de ce principe de transparence à la fois vis-à-vis de l'utilisateur humain (concepteur de modèle) que de programmes tiers (autres modèles de simulation, autres applications). Elle concerne la virtualisation du noyau même de la simulation, mais aussi celle des utilitaires (API) périphériques à la mise en œuvre d'une entreprise complète de M&S (génération de nombres aléatoires, visualisation et animation…). Elle reste une direction de recherche non aboutie, même si le potentiel d'interopérabilité y est quasi optimal. En effet, elle s'apparenterait, dans la perspective de la séparation généralisée des préoccupations présentées en Chapitre 1, à un standard dont les couches Automates et Connecteurs seraient enterrées (quoique bien séparées) dans un environnement portable sur n'importe quelle plateforme, et qui accepterait tout modèle (exprimé dans un langage de haut niveau, équivalent de ce que représenterait le pseudo-code Java pour la machine virtuelle de Java).

Chapitre 4.
Qualité

Les modèles de simulation sont approximatifs par essence. Leur qualité impacte donc le degré de confiance du processus décisionnel qui les mobilise.

L'ingénierie des modèles de M&S pose le problème mal maîtrisé de la qualité des modèles de simulation. C'est la garantie que les hypothèses de conception, les données et les algorithmes utilisés résolvent bien le problème identifié et non un autre. En effet, un modèle étant un " perçu " du réel, il ne reflète pas tous les aspects inclus dans le réel, mais seulement ceux qui ont une pertinence vis-à-vis des questions motivant l'étude. Les phénomènes physiques n'étant pas invariants à la simplification, cette approximation doit rester crédible dans le cadre de l'étude envisagée. Ceci implique la validité de la simplification faite, tant lors du passage du problème au modèle abstrait (appelée qualification ou validation conceptuelle), que de la transformation du modèle abstrait en code (dite vérification ou validation « amont »), et de la confrontation du code au problème réel (dite validation opérationnelle ou validation « aval »). En d'autres termes, la question de la crédibilité doit être examinée le long de tout le cycle de vie M&S, de l'élucidation du problème à l'utilisation des résultats de simulation. Toute déficience constatée est le plus souvent due à l'une des deux causes suivantes : (i) le modèle adopté n'est pas adéquat au problème posé, (ii) les données ne sont pas valides.

4.1. Vérification & validation

La densité d'expérience de recherche dans le domaine de la crédibilité des modèles de simulation, a permis d'établir des principes de base, et des techniques associées pour leur mise en œuvre, rassemblés sous l'étiquette V&V (*"Verification and Validation"*), ou VV&T (*"Verification, Validation and*

Test"), ou même VV&A (*"Verification, Validtion and Accreditation"*). Quelques travaux très représentatifs dans ce domaine sont [Balci 1997], [Robinson 1999] et [Lewis 1993]. Elles rassemblent un certain nombre de techniques, que la littérature classe en :

- Crédibilité réplicative, où les techniques classiques reposent sur la comparaison des trajectoires de sortie respectives du modèle et du système. On y retrouve l'analyse statistique standard de moyenne et variance, les tests T^2 de Hotteling, l'analyse multi-variance, l'analyse de régression, l'analyse spectrale, l'analyse autorégressive, les tests de fonction d'autocorrélation, l'analyse d'erreur, et les méthodes non paramétriques. Des modèles particuliers ont fait l'objet d'études intensives de crédibilité réplicative ; ce sont les générateurs de nombres aléatoires (ou RNG, pour Random Number Generation), modèles représentatifs de l'aléa, du hasard et de phénomènes stochastiques, et qui en réalité sont fortement biaisés par la nature récurrente des formules qu'ils utilisent (voir par exemple [De Matteis & Pagnutti 1988], [Coddington 1997], [Traoré & Hill 2001 b], [Pawlowski 2003], et [Reuillon et al. 2008], parmi la profusion de travaux dans ce domaine).
- Crédibilité prédictive, où les méthodes classiques sont de trois types :
  (i) les techniques subjectives, amplement pratiquées surtout dans les champs économiques et sociaux (comparaison graphique des données comportementales entre modèle et système, concordance et fluctuation des valeurs de segments, tests de Turing où des experts du système sont invités à différencier les résultats issus du système et ceux issus du modèle) ;
  (ii) l'analyse de sensibilité du comportement du modèle aux variations de paramètres, qui permet d'identifier les paramètres d'entrée ou internes qui doivent être affinés pour obtenir une certaine précision de sortie ; et
  (iii) le calibrage, qui concerne le réglage des données qui sont fournies comme paramètres au modèle.
- Crédibilité structurelle, qui couvre les techniques de programmation et de débogage. Une voie en phase de gestation, qui est en prise directe avec cette forme de crédibilité, est la notion de reproductibilité [].

Pour la plupart des méthodes utilisées ici, les réponses sont élaborées à partir de l'automate de simulation. Elles sont donc appropriées pour apprécier des

métriques de type opérationnel. Pour l'examen de questions de nature sémantique (telles, l'incohérence ou l'incomplétude d'une spécification, ou encore la composabilité d'un modèle avec un autre, etc.), des capacités de manipulation symbolique des modèles (et non des automates) sont nécessaires. L'analyse formelle de propriété en est la forme la plus emblématique.

4.2. Analyse formelle de propriétés

Rendre les modèles de simulation accessibles à l'analyse formelle permet de répondre à des questions telles que :

- La spécification est-elle cohérente ? N'y-a-t-il pas des propriétés contradictoires ?
- Existe-t-il un état bloquant dans lequel le modèle peut tomber et ne plus jamais évoluer ?
- Le modèle possède-t-il une propriété donnée ? L'intégrité des informations est-elle maintenue (garantie des invariants par exemple) ?
- Avant d'ajouter une propriété à la spécification, est-il possible de s'assurer qu'elle n'est pas déjà déductible de la spécification existante ?
- Est-il est possible de réduire un ensemble de propriétés du modèle à un ensemble de moindre cardinalité (réduction de modèle) ?
- Le modèle est-il équivalent à un autre ?
- Telle trace est-elle une trace valide pour le système ?

Ces questions non exhaustives, mettent en exergue la question de la qualité du modèle, en termes de propriétés statiques, i.e., des propriétés vérifiables de manière indépendante d'un instant précis de la vie du système. Elles mobilisent donc à la fois les traces du système (i.e., son comportement, comme pour les méthodes d'étude des propriétés dynamiques), mais aussi et surtout sa structure. Les méthodes utilisées sont connues sous le label « méthodes formelles ».

Les méthodes formelles sont des techniques mathématiques appliquées à la spécification, à l'analyse, à la conception et à la mise en œuvre de logiciels et de matériels complexes [Clarke & Wing 1996], [Kuhn et al. 2003]. Elles reposent sur des spécifications formelles (donc des langages formels, tels Z [Spivey 1992], CSP [Hoare 1985], VDM [Dines & Jones 1978], B [Abrial 1996], CTL

[Emerson 1991], ou les Statecharts [Harrel 1987]) et les techniques qu'elles développent (et qui sont parfois supportées par des outils) sont réparties en deux grandes classes, à savoir :

- La vérification formelle, technique automatisée pour vérifier l'absence de violation, par un modèle spécifié formellement, d'une propriété spécifiée formellement. Typiquement, ces propriétés sont l'absence de blocage du système et les invariants du système. La vérification formelle peut être utilisée même sur des spécifications partielles, et peut produire des contre-exemples qui sont généralement de subtiles erreurs de conception. Le gros problème de cette approche est le risque d'explosion combinatoire, son principe reposant sur un examen exhaustif des états du système. Des exemples d'outils de vérification formelle sont SPIN [Holzmann 1997], SMV [McMillan 1992], et UPPAAL [Larsen et al. 1997].
- La preuve de théorème, technique où le système et ses propriétés souhaitées sont exprimés sous forme de formules logiques, en termes d'axiomes et de règles d'inférence. La preuve du théorème est le processus consistant à démontrer la véracité d'une propriété à partir de ces axiomes et de ces règles, et parfois de définitions dérivées et de lemmes intermédiaires. Cette technique n'est pas sujette au risque d'explosion combinatoire, mais peut s'avérer lourde et difficile à mener. Des exemples d'outils de preuve de théorèmes sont Z/EVES [Saaltink 1997], FDR [Broadfoot & Roscoe 2000], HOL [Gordon & Melham 1993], Isabelle [Paulson 1993], et PVS [Owre et al. 1992].

Toutes ces méthodes tombent dans une classification (celle de [Lamsweerde 2000]) qui distingue cinq catégories, à savoir :

(1) Les méthodes historique-centrées, qui s'intéressent aux traces du système et font usage de la logique temporelle.
(2) Les méthodes états-centrées, qui caractérisent un système par des invariants, des opérations et des conditions amont et aval d'application de ces opérations (dites conditions pré et post).
(3) Les méthodes transitions-centrées, qui caractérisent un système par des fonctions de transition donnant pour un état et un événement stimulant, la sortie correspondante du système.
(4) Les méthodes fonctionnelles, qui caractérisent un système par une collection de fonctions formant une théorie algébrique.

(5) Les méthodes opérationnelles, qui caractérisent un système par une collection structurée de processus exécutables sur une machine abstraite.

Notons que ce qui empêche des formalismes puissants comme DEVS d'appartenir à l'une de ces catégories (comme celle des méthodes états-centrées, ou celle des méthodes transitions-centrées), c'est l'absence de théorie de preuve. D'importantes contributions ont été faites pour y remédier en établissant un lien entre DEVS et les automates temporisés [Giambiasi et al. 2003], [Châne et al. 2004], [Hernandez & Giambiasi 2005], [Dacharry & Giambiasi 2007]. D'autres travaux notables jetant un pont entre DEVS et méthodes formelles sont [Cristia 2007], [Posse 2008], [Hong et al. 1997], [Hwang & Zeigler 2009], [Saadawi & Wainer 2009], [Weisel et al. 2005] et [Traoré 2006]. Pour une approche générale d'intégration des méthodes formelles et de la M&S, très peu de travaux sont disponibles dans la littérature, dont [Kuhn et al. 2003] et [Stevenson 2003].

Par ailleurs, le problème d'hétérogénéité de perspectives inhérente aux systèmes complexes (comme mentionné dans le Chapitre 2) pousse, ici aussi, à combiner des méthodes formelles de différentes catégories pour couvrir le spectre des propriétés analysables. Des exemples notables sont :

- CSP-Z [Fischer 2000], extension de Z pour spécifier les processus CSP et dont l'outil support est FDR.
- ZCCS [Galloway et Stoddart 1997], combinaison de Z et de Calculus of Communicating Systems (CCS) [Milner 1980].
- CSP-OZ, intégration d'Object-Z et de CSP, dont une alternative a été définie dans [Smith 1992].
- CSP2B [Butler 1999] et CSP||B [Schneider & Treharne 2002], deux variantes combinant CSP et B.
- Timed CSP [Reed & Roscoe 1986], élargissement de CSP pour inclure des aspects temporels.

4.3. Vers un cadre algébrique de M&S

Les efforts entrepris dans le cadre du formalisme HiLLS s'intéressent au pont qu'il est possible d'établir entre l'universalité en simulation et l'analysabilité. Il s'agit donc d'une fusion de la théorie des systèmes (dépourvue de théorie de

preuve) avec les approches de spécification formelle (qui, elles, autorisent le raisonnement symbolique mais ne sont pas déclinées de manière spécifique pour la M&S), en mettant l'accent sur les méthodes formelles pour lesquelles des outils de preuve automatique sont déjà disponibles, pour éviter d'investir des efforts supplémentaires dans la construction de tels outils.

4.3.1. Base de travail : notion de contexte

Un modèle de simulation est toujours conçu pour un objectif initial donné et sous des hypothèses et des contraintes spécifiques. Il est donc valide dans un contexte particulier, le plus souvent (voire toujours) non explicite, qui constitue le cadre sémantique auquel il adhère naturellement. Aussi, son utilisation dans un quelconque autre contexte devrait avoir pour préalable la vérification de son adhésion à ce contexte. Rendre explicite la dualité sémantique qui lie un modèle à un contexte (pris dans ce sens) ouvrirait la voie au raisonnement symbolique sur la structure et le comportement du modèle lorsque situé dans ce contexte.

4.3.1.i. Point de départ : contexte expérimental

La notion généralisée de contexte provient de la notion de contexte expérimental, elle-même enracinée dans la notion de cadre expérimental. La notion de cadre expérimental a été introduite dans [Zeigler 1976] et développée dans [Zeigler 1984] pour traduire la dépendance entre l'expression du modèle et les objectifs de l'étude qui l'utilise. Un modèle est valide dans son cadre expérimental, et peut cesser de l'être dans un autre cadre expérimental. Zeigler qualifie de modèle de base un modèle qui serait valide dans n'importe quel cadre expérimental. Dans l'absolu, un modèle de base n'est pas concevable. Par contre, en se plaçant dans des contextes très larges, il est possible de constituer des bibliothèques de modèles, adaptables à un grand nombre de cadres expérimentaux. Un modèle issu d'une telle bibliothèque, devient un modèle dit « taillé » dès qu'il est aménagé pour s'adapter à un cadre expérimental donné.

L'approche « plug-in » proposée dans [Traoré & Muzy 2006] suggère une formalisation de la notion de cadre expérimental, puis sa généralisation à la

notion plus large de contexte. Ce point de départ est illustré par une étude simulation-centrée de propagation de feu réalisée à l'aide de modèles cellulaires. Le principe adopté est de considérer l'ensemble que constituent le modèle et son cadre expérimental comme un système à spécifier, puis à soustraire de cette spécification la part correspondant à la spécification du modèle (i.e., une sorte d'opération de « soustraction » dans l'espace des systèmes). Le cadre du modèle, selon cette approche, se présente alors comme montré par la Figure 41, i.e. un système ayant une interface avec son environnement (*Frame inputs*, et *Frame outputs*), mais présentant également une interface dédiée aux modèles pouvant lui être greffés (*Frame-to-Model inputs* et *Model-to-Frame outputs*).

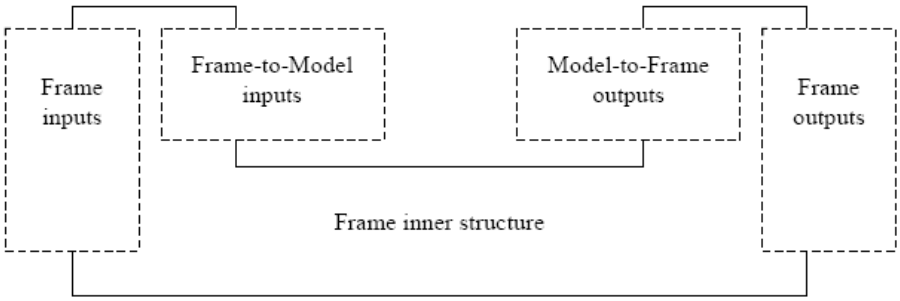


Figure 41. Approche « plug-in »

Chaque cellule du modèle cellulaire à greffer (appelé greffon), peut être indemne (*unburned*), chaude (*heating*), brûlante (*onFire*) ou brûlée (*burned*), et obéit à l'équation de la Figure 42 donnant sa température T_{ij}. Le temps est représenté par t, tandis que t_{ig} est la date à laquelle la cellule a pris feu et $\sigma_{v,0}$ est la masse combustible initiale. Au-delà du seuil T_{ig} la combustion se déclenche et se termine en deçà du seuil T_f. Les coefficients a, b, c et d dépendent du pas de temps de simulation et de la granularité de l'espace discrétisé.

$$T_{i,j}^{k+1}=a(T_{i-1,j}^{k}+T_{i+1,j}^{k})+b(T_{i,j-1}^{k}+T_{i,j+1}^{k})+c\sigma_{v,0}e^{-\alpha(t-t_{ig})}+dT_{i,j}^{k}$$

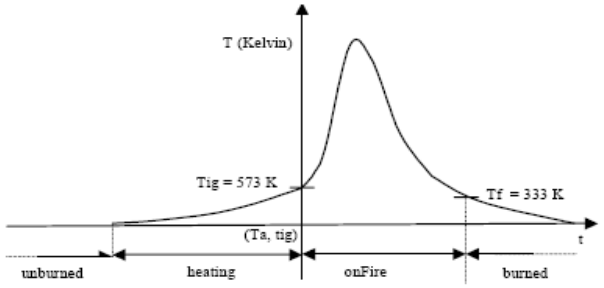


Figure 42. Modèle mathématique de feu

Le cadre expérimental destiné à faire des expérimentations sur un modèle et à comparer ses résultats avec des valeurs de référence (issues d'une expérience précédente ou d'une base de données) est schématisé par la Figure 43. Le contrôle de la simulation est réalisé à travers le port *start/stop*. Les composants internes sont : (1) un générateur qui envoie des ordres de mise à feu de cellule au modèle à greffer, (2) un traducteur qui récupère la grille de température de l'espace cellulaire et calcule l'écart avec des valeurs de référence, et (3) un assesseur qui valide ou pas l'expérience en fonction de la durée de la simulation et de l'écart avec les valeurs de référence. Deux greffons ont été définis : un modèle classique à base d'automates cellulaires (CA) et un modèle appelé DSCA (pour Dynamic Structure Cellular Automata) [Barros & Mendes 1997], [Muzy et al. 2003] qui prend appui sur des formalismes pour systèmes à structure variable [Barros 1997]. DSCA limite les calculs à chaque étape de vie du modèle aux seules cellules actives, contrairement à CA qui effectuent les calculs pour toutes les cellules.

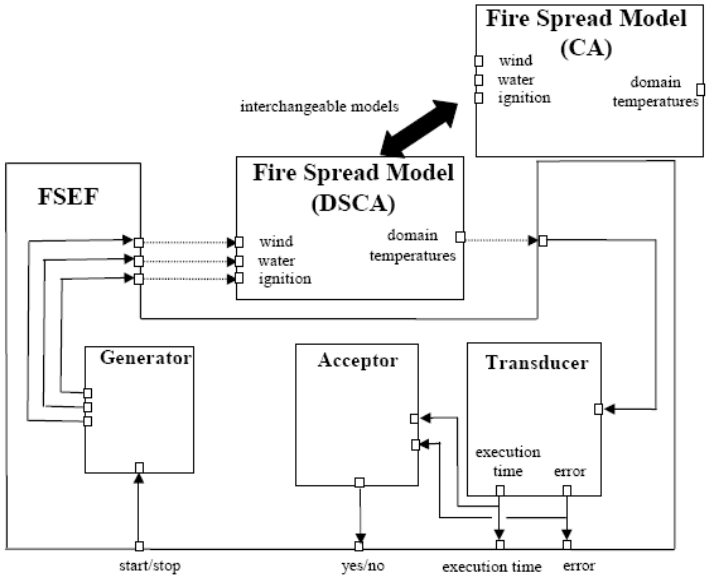


Figure 43. Cadre expérimental de l'étude des feux

Les expérimentations des différents modèles cellulaires avec le même cadre expérimental produisent les mêmes motifs d'évolution du front de feu (Figure 44). Cette figure montre également les performances comparées (en utilisant un *Pentium III 500 MHz*) des deux modèles cellulaires conçus. Les validations et invalidations apparaissent sur la courbe de comparaison.

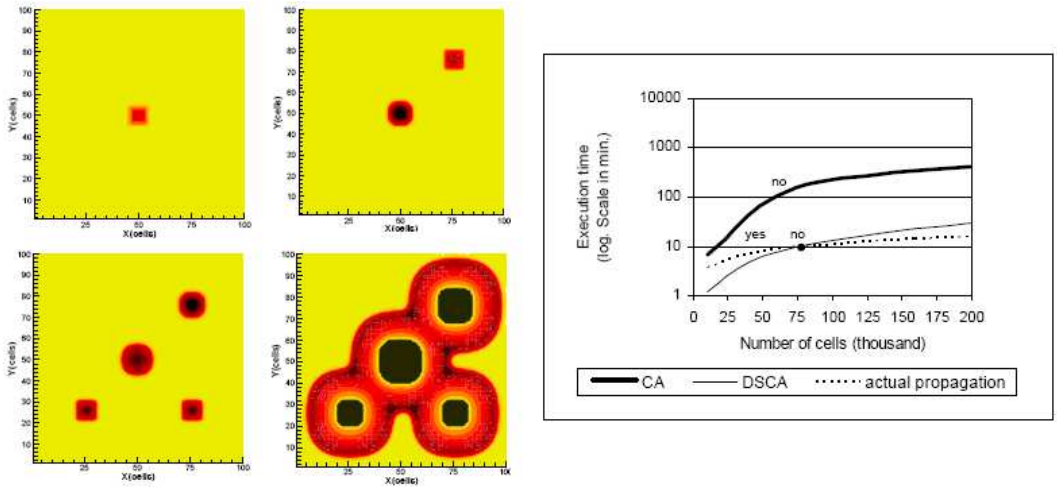


Figure 44. Résultats de la mise en œuvre du cadre expérimental

4.3.1.ii. Contexte généralisé

Partons maintenant de la question du couplage de ce cadre expérimental avec d'autres modèles. Deux questions majeures viennent immédiatement à l'esprit :

(i) Le couplage est-il opérationnellement réalisable ? En d'autres mots, les deux modèles (cadre et greffon) sont-ils interopérables ? Cette question se ramène à un problème de compatibilité d'interfaces et se règle le plus souvent au niveau de l'implémentation logicielle.

(ii) Le couplage est-il sémantiquement correct ? Cette question, plus complexe, en appelle au sens même du couplage. Une première interrogation immédiate à ce niveau est celle de la cohérence des échelles de temps entre les deux modèles, en nature de flux (flux tous discrets, tous continus, ou l'un discret et l'autre continu) et en unité temporelle. Il est possible de renvoyer cette interrogation, soit à la question d'interopérabilité (tant il est vrai que le temps est une dimension naturellement incluse dans la structure des interfaces des modèles de simulation, puisque les messages échangés sont estampillés par leur temps dans la simulation), soit à la seconde interrogation immédiate (plus générale, elle) relative à la cohérence des processus résultant des deux modèles au regard du contexte global considéré et des contextes d'origine de ces modèles.

Derrière la dualité modèle/contexte transparaît donc plusieurs préoccupations. Par conséquent, la notion de contexte peut être surchargée, conduisant ainsi à

plusieurs entendements différents. Aussi, est-il important de distinguer au moins les types suivants de contexte :

- Le contexte d'expérimentation, usuellement reconnu sous le nom de cadre expérimental en mode génératif, et qui décrit les dispositifs nécessaires à l'expérimentation sur un modèle. Le couplage entre un modèle et son contexte d'expérimentation définit une famille d'expériences faisables sur le modèle.
- Le contexte d'observation, usuellement reconnu sous le nom de cadre expérimental en mode acceptant, et qui décrit l'ensemble des dispositifs nécessaires à la simple prise de mesures observatoires. Le couplage entre un modèle et son contexte d'observation ne sert qu'à alimenter une base de données du comportement de ce modèle (sans qu'une expérience ne soit particulièrement initiée).
- Le contexte d'utilisation, qui décrit le processus de mise en œuvre du modèle dans un dispositif de résolution de problème. Un tel composant, spécifiable sous forme de modèle de résolution selon une approche de couplage simulation-centrée (comme décrit dans le Chapitre 2), doit être intégré dans un environnement de prise de décision (couplage avec des outils de RO et/ou d'IA…) avec des boucles de rétroaction entre la simulation et les méthodes de recherche et de calcul de décision.
- Le contexte de validité logique (ou simplement contexte logique), qui lui exprime les conditions associées à la sémantique du modèle : ce que le modèle est censé produire et les hypothèses et contraintes présumées. C'est à ce niveau que se traite réellement la question de la qualité des modèles. Le contexte de validité décrit l'ensemble des circonstances dans lesquelles le modèle est une représentation valide du système réel. Une telle spécification, présente en principe sous forme de conditions logiques de validité, doit être soumise à des outils de vérification et de preuve de théorème, pour établir la cohérence du modèle (complétude de la spécification du modèle au regard des objectifs visés, légitimité de son utilisation dans ces circonstances, etc.).

La manipulation de la notion de dualité entre contexte et modèle pose la question de l'applicabilité d'un contexte donné à un modèle donné. Selon les types de contexte susmentionnés, cette question en traduit plusieurs autres, comme par exemple :

- La substituabilité d'un modèle par un autre nécessite leur équivalence vis-à-vis du contexte logique correspondant.
- La réutilisabilité d'un modèle nécessite sa compatibilité avec le contexte logique correspondant, et son interopérabilité avec le contexte expérimental correspondant. Il faut noter que la substitution de contexte expérimental revient à de la réutilisation de modèle, et que la substitution de modèle revient à de la réutilisation de contexte expérimental.
- La composabilité de deux modèles peut être vue comme un problème de double compatibilité entre un modèle et son homologue pris comme contexte d'utilisation.

4.3.2. Sémantique pour analyse formelle

L'idée générale de la dualité entre modèle et contexte est que l'analyse formelle de propriété s'interprète comme une confrontation entre spécifications (celles du modèle et celle de son contexte logique). Deux cas de figure peuvent se présenter :

- le modèle spécifie la structure du système et le contexte spécifie les exigences du système ; ou alors
- le modèle spécifie le comportement du système (i.e., ses traces) et le contexte spécifie les propriétés statiques attendues de ces traces.

Il est possible de réaliser l'une ou l'autre des approches avec HiLLS. Il existe une sémantique Z de HiLLS, que nous ne développons pas dans cet ouvrage, qui permet de réaliser le premier cas. Nous nous focalisons ici sur le second cas, le bénéfice recherché est d'avoir une syntaxe concrète complète (i.e., incluant la spécification de modèle de système et celle de modèle d'exigences dans un cadre homogène).

4.3.2.i. Logique temporelle

La logique temporelle (TL) est un terme général utilisé pour décrire les cadres logiques permettant de représenter et de raisonner sur le temps et l'information temporelle. Elle est utilisée pour la spécification et la vérification des propriétés

des exécutions de programmes et de systèmes informatiques [Lamport 1977], [Lamport 1994], [Goranko & Galton 2015]. Selon [Lamport 1994], la TL est particulièrement conçue pour raisonner sur les séquences d'états représentatifs des changements de valeur des variables des algorithmes en cours d'exécution. La sémantique opérationnelle de l'algorithme est la collection de toutes ses exécutions possibles. Dans le cas de la théorie des systèmes, cette collection décrit le comportement d'un système. Par conséquent, à l'aide de techniques de vérification formelle [Baier et al. 2008], [Clarke et al. 1999], il est possible de vérifier si un modèle donné du système réel satisfait ou non les propriétés requises. Les propriétés temporelles généralement décrites sont classées en trois grandes catégories [Lamport 1977] : sécurité (i.e., évitement de la survenue d'événements indésirables), vivacité (i.e., garantie de réalisation de propriétés spécifiques) et équité (i.e., assurance de la survenue d'événements spécifiques sous des conditions données). La TL étend la logique des prédicats à la prise en compte de la dimension temporelle [Smith 1992]. Néanmoins, elle ne fait pas référence au moment précis d'occurrence d'un événement ou d'un état. En ce sens, le concept de temps dans la TL est logique plutôt que physique. Ce temps peut d'ailleurs être linéaire (on parle alors de TL linéaire, ou LTL pour Linear Temporal Logic) ou en arbre (on parle alors de TL arborescente, ou CTL pour Computation Tree Logic) [Baier et al. 2008]. Il existe des outils tels que NuSMV [Cimatti et al. 2000], NuSMV2 [Cimatti et al. 2002], TSMV [Markey & Schnoebelen 2004], SPIN [Hol97], TLAPS [Chaudhuri et al. 2010], et TaLiRo [Fainekos et Pappas 2008] pour automatiser la vérification rigoureuse des propriétés temporelles des systèmes de manière transparente à l'utilisateur. Néanmoins, ce dernier doit pouvoir spécifier correctement les propriétés à vérifier dans le formalisme de spécification supporté par l'outil de vérification choisi. Il lui faut donc un certain niveau d'expertise dans la manipulation des idiomes de la logique et des mathématiques discrètes pour lire correctement et/ou écrire les propriétés traduisant des exigences complexes. Le manque de cette expertise a été largement reconnu par la communauté scientifique comme l'un des principaux inhibiteurs de l'adoption généralisée d'outils de vérification formelle et, par conséquent un verrou à la traduction des résultats de recherche en méthodes formelles dans la pratique.

Dans un effort pour proposer une solution à ce problème, et inspirés par le succès des *Design Patterns* en Génie logiciel, certains chercheurs ont proposé des motifs paramétrés, dans un format indépendant du formalisme que

l'utilisateur pourrait adopter, qui dresse une cartographie des solutions de spécification à des exigences récurrentes [Dwyer et al. 1999]. Ces motifs paramétrés ont été élaborés à partir de (et pour) plusieurs centaines de modèles de propriété reconnus dans cinq langages formels : LTL, CTL, QRE [Dwyer & Clarke 1994], GIL [Melliar-Smith 1988] et INCA [Corbett & Avrunin 1995]. Avec leur succès grandissant auprès des praticiens de méthodes formelles, ils ont ensuite été reproduits dans ACTL [Ferro 1994] et μ-calculus [Kozen 1983]. La philosophie de ces motifs, qui est adoptée dans HiLLS également, utilise deux composantes de spécification :

- la spécification de la propriété à vérifier (i.e., le motif lui-même) ;
- la spécification de sa portée (i.e., la période pendant laquelle la propriété doit être satisfaite).

4.3.2.ii. Spécification HiLLS de propriétés temporelles

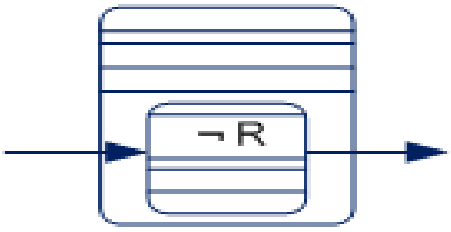
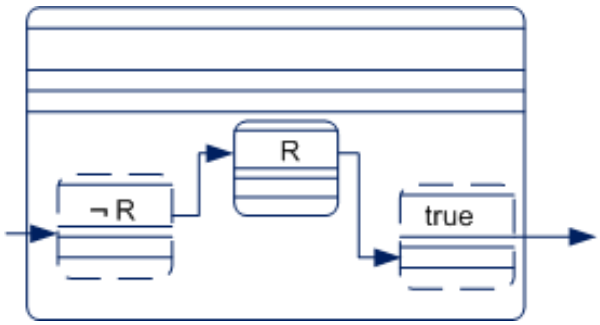
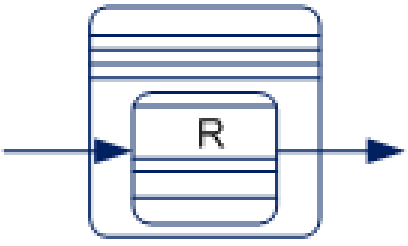
Nous pensons, à l'instar de [Meyers et al. 2013], que l'uniformité de notation entre le modèle du système et celui de ses d'exigences, est un atout. C'est pourquoi, les propriétés temporelles sont exprimées en HiLLS en utilisant les mêmes éléments de syntaxe (configurations, transitions, etc.) introduits précédemment. Toutefois, deux concepts additionnels sont apportés (comme montré en Figure 45) :

- La configuration générique (montrée à gauche dans la Figure 45) est une configuration abstraite, dont les angles arrondis indiquent qu'elle représente toute succession de configurations (passives, instantanées ou régulières) dans lesquelles la proposition spécifiée est vraie. Seules les caractéristiques communes à toutes les configurations de cette séquence sont spécifiées.
- La configuration hypothétique (montrée à droite dans la Figure 45) est une configuration abstraite, dont les contours en pointillé indiquent qu'elle représente l'existence ou non d'une configuration générique.

Figure 45. Configurations générique et hypothétique dans HiLLS

Les Tables 5 et 6 présentent les spécifications HiLLS des motifs fondamentaux. La Table 5 présente les motifs dits d'occurrence (i.e., en lien avec l'existence ou l'absence d'une propriété), tandis que la Table 6 présente les motifs dits d'ordre (i.e., en lien avec les règles de précédence entre propriétés). Il faut noter que les transitions entre les configurations génériques et/ou hypothétiques sont abstraites, sans déclencheurs, ou événements de sortie. Par conséquent, ils n'indiquent pas quels types de transition de configuration sont en jeu.

Table 5. Spécification de motifs d'occurrence

Nom	***Motif***	***Description***
Absence		*Jamais R* : le système ne vérifie jamais la propriété R.
Existence		*R existe :* le système vérifiera la propriété R à un moment de son existence. Avant, il pourrait ne pas l'avoir vérifié (¬R), et après tout peut arriver (*true*).
Universality		*Toujours R* : le système vérifie toujours la propriété R.

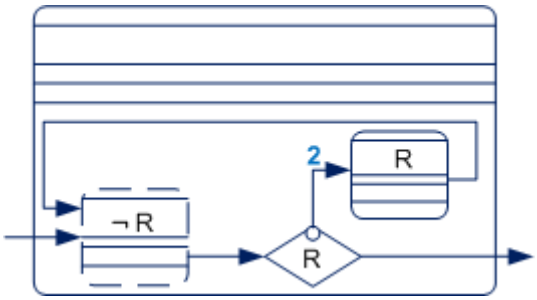
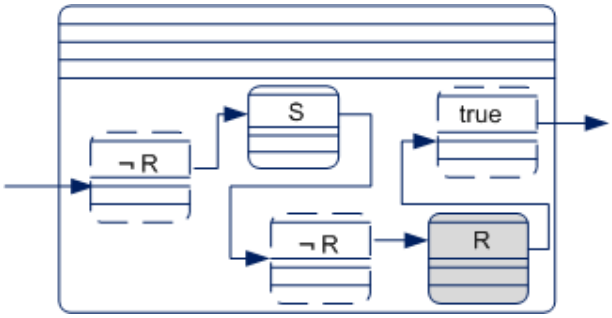
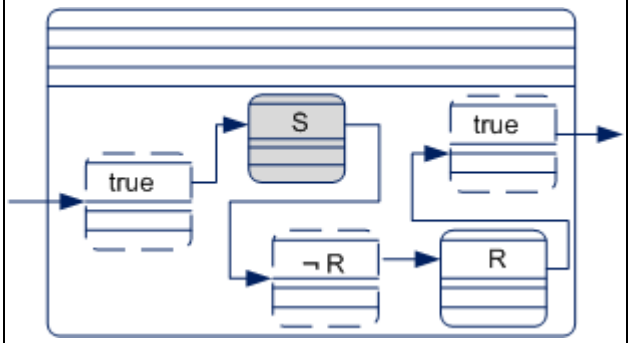
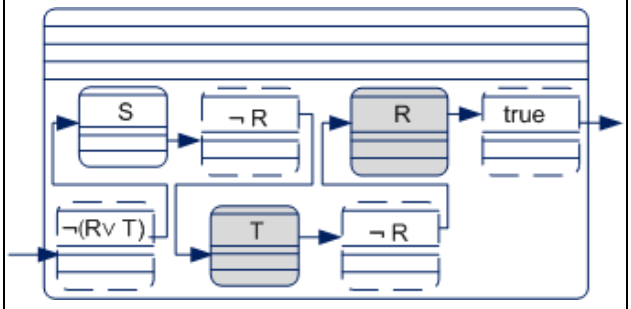
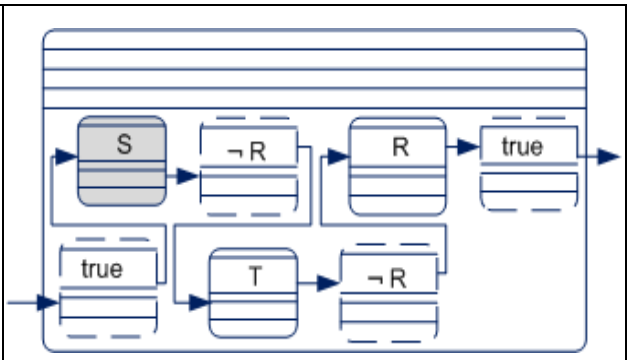
Bounded existence		*R existe au plus n fois* : le système vérifiera la propriété R n fois (ici 2) et pas plus.

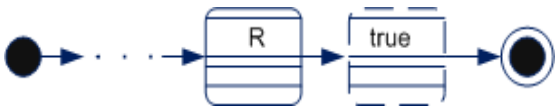
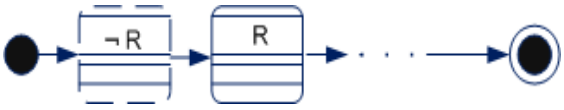
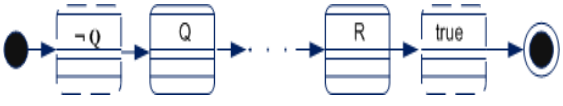
Table 6. Spécification de motifs d'ordre

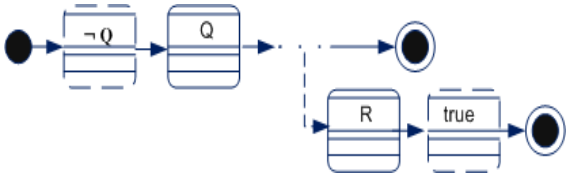
Nom	***Motif***	***Description***
Precedence		*S précède R* : le système vérifie les propriétés S et R, mais dans cet ordre exclusivement.
Response		*R répond à S* : chaque fois que le système vérifie S, il doit vérifier R.
Precedence chain		*S précède T et R* : le système vérifie les propriétés S, T et R, mais la satisfaction de S doit précéder celle des autres, et T doit précéder R.

Response chain		*T et R répondent à S* : chaque fois que le système vérifie S, il doit vérifier T et R, et dans cet ordre.

La Table 7 présente les spécifications HiLLS des portées. L'utilisateur combine motifs et portées en reportant les premières dans les secondes, à l'emplacement des points de suspension, pour produire une spécification complète d'exigences. De même, c'est en combinant ces spécifications à base de motif qu'il pourra exprimer des exigences complexes de manière aussi pratique que la combinaison des *Design Patterns* permet de proposer un modèle de conception des systèmes informatiques [Gamma et al. 1995].

Table 7. Spécification de portée

Nom	***Portée***	***Description***
Globally		*Partout* : la propriété à vérifier doit être satisfaite partout
Before		*Avant R* : la propriété à vérifier doit être satisfaite avant que R ne soit satisfaite.
After		*Après R* : la propriété à vérifier doit être satisfaite après que R soit satisfaite.
Between		*Entre Q et R* : la propriété à vérifier doit être satisfaite après que Q soit satisfaite et avant que R ne soit satisfaite.

After until		*De Q à R* : la propriété à vérifier doit être satisfaite après satisfaction de Q et jusqu'à satisfaction de R.

La sémantique logique de ces spécifications est donnée par la Table 8 pour l'ensemble des motifs (d'occurrence et d'ordre). Cette sémantique est exprimée, à la fois en LTL et en CTL. Les variables p, q, r, s et t sont les prédicats définis par l'utilisateur. A noter que dans ces expressions W désigne l'opérateur défini par l'une ou l'autre des trois équations équivalentes suivantes (Y désignant le quantificateur *Until*, □ désignant *Always*, et ◊ désignant *Eventually*) :

$$pWq = (\Box p) \vee (pYq)$$

$$pWq = \Diamond(\neg p) \Rightarrow (pYq)$$

$$pWq = pY(q \vee \Box p)$$

A titre d'illustration, la Figure 46 montre les spécifications HiLLS respectives des trois exigences suivantes pour le BVM précédemment présenté :

(1) Le BVM ne délivre pas de boisson tant que le solde est insuffisant.

(2) Le BVM rend toujours la monnaie sur un solde excédentaire.

(3) Un achat est irrévocable dès que le solde est suffisant.

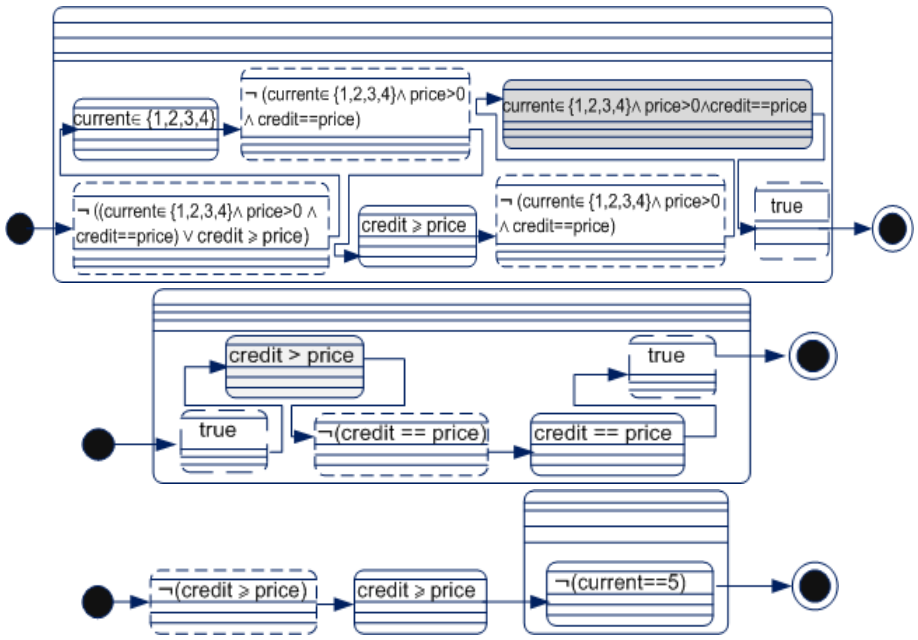


Figure 46. Spécifications HiLLS de trois exigences du BVM

Table 8. Sémantique logique des spécifications d'exigences

Motif	Portée	Correspondance en LTL	Correspondance en CTL
Absence	Partout	$\Box(\neg p)$	$\forall\Box(\neg p)$
	Avant *r*	$\Diamond r \Rightarrow(\neg p \mathrm{Y} r)$	$\forall[(\neg p \vee \forall\Box(\neg r)) \mathrm{W} r]$
	Après *q*	$\Box(q \Rightarrow \Box(\neg p))$	$\forall\Box(q \Rightarrow \forall\Box(\neg p))$
	Entre *q* et *r*	$\Box((q \wedge \neg r \wedge \Diamond r) \Rightarrow (\neg p \mathrm{Y} r))$	$\forall\Box(q \wedge \neg r \Rightarrow \forall[(\neg p \vee \forall\Box(\neg r)) \mathrm{W} r])$
	De q à r *r*	$\Box((q \wedge \neg r) \Rightarrow (\neg p \mathrm{W} r))$	$\forall\Box(q \wedge \neg r \Rightarrow \forall[\neg p \mathrm{W} r])$
Existence	Partout	$\Diamond p$	$\forall\Diamond p$
	Avant *r*	$\neg r \mathrm{W}(p \wedge \neg r)$	$\forall[\neg r \mathrm{W}(p \wedge \neg r)]$
	Après *q*	$\Box(\neg q) \vee \Diamond(q \wedge \Diamond p))$	$\forall[\neg q \mathrm{W}(q \wedge \forall\Box(p))]$
	Entre *q* et *r*	$\Box(q \wedge \neg r \Rightarrow (\neg r \mathrm{W}(p \wedge \neg r)))$	$\forall\Box(q \wedge \neg r \Rightarrow \forall[\neg r \mathrm{W}(p \wedge \neg r)])$
	De q à *r*	$\Box(q \wedge \neg r \Rightarrow (\neg r \mathrm{Y}(p \wedge \neg r)))$	$\forall\Box(q \wedge \neg r \Rightarrow \forall[\neg r \mathrm{Y}(p \wedge \neg r)])$
Universality	Partout	$\Box p$	$\forall\Box(p)$
	Avant *r*	$\Diamond r \Rightarrow (p \mathrm{Y} r)$	$\forall[(p \vee \forall\Box(\neg r)) \mathrm{W} r]$
	Après *q*	$\Box(q \Rightarrow \Box(p))$	$\forall\Box(q \Rightarrow \forall\Box(p))$
	Entre *q* et *r*	$\Box((q \wedge \neg r \wedge \Diamond r) \Rightarrow (p \mathrm{Y} r))$	$\forall\Box(q \wedge \neg r \Rightarrow \forall[(p \vee \forall\Box(\neg r)) \mathrm{W} r])$
	De q à *r*	$\Box(q \wedge \neg r \Rightarrow (p \mathrm{W} r))$	$\forall\Box(q \wedge \neg r \Rightarrow \forall[p \mathrm{W} r])$
Bounded Existence	Partout	$(\neg p \mathrm{W}(p \mathrm{W}(\neg p \mathrm{W}(p \mathrm{W} \Box \neg p))))$	$\neg\exists\Diamond(\neg p \wedge \exists\circ(p \wedge \exists\Diamond(\neg p \wedge \exists\circ(p \wedge \exists\Diamond(\neg p \wedge \exists\circ(p))))))$
	Avant *r*	$\Diamond r \Rightarrow ((\neg p \wedge \neg r) \mathrm{Y}(r \vee ((p \wedge \neg r) \mathrm{Y}(r \vee ((\neg p \wedge \neg r) \mathrm{Y}(r \vee ((p \wedge \neg r) \mathrm{Y}(r \vee (\neg p \mathrm{Y} r)))))))))$	$\neg\exists[\neg r \mathrm{Y}(\neg p \wedge \neg r \wedge \exists\circ(p \wedge \mathrm{E}[\neg r \mathrm{Y}(\neg p \wedge \neg r \wedge \exists\circ(p \wedge \exists[\neg r \mathrm{Y}(\neg p \wedge \neg r \wedge \exists\circ(p \wedge \neg r))])))])]$
	Après *q*	$\Diamond q \Rightarrow (\neg q \mathrm{Y}(q \wedge (\neg p \mathrm{W}(p \mathrm{W}(\neg p \mathrm{W}(p \mathrm{W} \Box \neg p))))))$	$\neg\exists[\neg q \mathrm{Y}(q \wedge \exists\Diamond(\neg p \wedge \exists\circ(p \wedge \exists\Diamond(\neg p \wedge \exists\circ(p \wedge \exists\Diamond(\neg p \wedge \exists\circ(p))))))))]$
	Entre	$\Box((q \wedge \Diamond r) \Rightarrow ((\neg p \wedge \neg r) \mathrm{Y}(r \vee ((p \wedge \neg p \mathrm{Y}(r \vee ((\neg p \wedge \neg r)$	$\forall\Box(q \Rightarrow \neg\exists[\neg r \mathrm{Y}(\neg p \wedge \neg r \wedge \mathrm{E}\circ(p \wedge \exists[\neg r \mathrm{Y}(\neg p \wedge \neg r \wedge \exists$

	q et r	$\mathrm{Y}(r\vee((p\wedge\neg r)\mathrm{Y}(r\vee(\neg p\mathrm{Y}r))))))))))$	$\circ(p\wedge\exists[\neg r\mathrm{Y}(\neg p\wedge\neg r\wedge\exists\circ(p\wedge\neg r\wedge\exists\Diamond(r)))])))])$
	De q à r	$\Box(q\Rightarrow((\neg p\wedge\neg r)\mathrm{Y}(r\vee((p\wedge\neg r)\mathrm{Y}(r\vee((\neg p\wedge\neg r)$ $\mathrm{Y}(r\vee((pf\neg r)\mathrm{Y}(r\vee(\neg p\mathrm{w}r)\varpi\Box p))))))))$	$\forall\Box(q\Rightarrow\neg\exists[\neg r\mathrm{Y}(\neg p\wedge\neg r\wedge\exists\circ(pf\exists[\neg r\mathrm{Y}(\neg p\wedge\neg r\wedge\exists$ $\circ(p\wedge\exists[\neg r\mathrm{Y}(\neg p\wedge\neg r\wedge\exists\circ(p\wedge\neg r))])))])))])$
Precedence	Partout	$\neg p\mathrm{w}s$	$\forall[\neg p\mathrm{w}s]$
	Avant r	$\Diamond r\Rightarrow(\neg p\mathrm{Y}(s\vee r))$	$\forall[(\neg p\vee\forall\Box(\neg r))\mathrm{w}(s\vee r)]$
	Après q	$\Box\neg q\vee\Diamond(qf(\neg p\mathrm{w}s))$	$\forall[\neg q\mathrm{w}(q\wedge\forall[\neg p\mathrm{w}s])]$
	Entre q et r	$\Box((q\wedge\neg r\wedge\Diamond r)\Rightarrow(\neg p\mathrm{Y}(s\vee r)))$	$\forall\Box(q\wedge\neg r\Rightarrow\forall[(\neg p\vee\forall\Box(\neg r))\mathrm{w}(s\vee r)])$
	De q à r	$\Box(q\wedge\neg r\Rightarrow(\neg p\mathrm{w}(s\vee r)))$	$\forall\Box(q\wedge\neg r\Rightarrow\forall[\neg p\mathrm{w}(s\vee r)])$
Response	Partout	$\Box(p\Rightarrow\Diamond s)$	$\forall\Box(p\Rightarrow\forall\Diamond(s))$
	Avant r	$\Diamond r\Rightarrow(p\Rightarrow(\neg r\mathrm{Y}(s\wedge\neg r)))\mathrm{Y}r$	$\forall[((p\Rightarrow\forall[\neg r\mathrm{Y}(s\wedge\neg r)])\vee\forall\Box(\neg r))\mathrm{w}r]$
	Après q	$\Box(q\Rightarrow\Box(p\Rightarrow\Diamond s))$	$\forall[\neg q\mathrm{w}(q\wedge\forall\Box(p\Rightarrow\forall\Diamond(s)))]$
	Entre q et r	$\Box((q\wedge\neg r\wedge\Diamond r)\Rightarrow(p\Rightarrow(\neg r\mathrm{Y}(s\wedge\neg r)))\mathrm{Y}r)$	$\forall\Box(q\wedge\neg r\Rightarrow\forall[((p\Rightarrow\forall[\neg r\mathrm{Y}(s\wedge\neg r)])\vee\forall\Box(\neg r))\mathrm{w}r])$
	De q à r	$\Box(q\wedge\neg r\Rightarrow((p\Rightarrow(\neg r\mathrm{Y}(s\wedge\neg r)))\mathrm{w}r)$	$\forall\Box(q\wedge\neg r\Rightarrow\forall[(p\Rightarrow\forall[\neg r\mathrm{Y}(s\wedge\neg r)])\mathrm{w}r])$
Precedence chain	Partout	$(\Diamond(s\wedge\circ\Diamond\mathrm{T}))\Rightarrow((\neg s)\mathrm{Y}p))$	$\neg\exists[\neg p\mathrm{Y}(s\wedge\neg p\wedge\exists\circ(\exists\Diamond(t)))]$
	Avant r	$\Diamond r\Rightarrow((\neg(s\wedge(\neg r)\wedge\circ(\neg r\mathrm{Y}(t\wedge\neg r))))\mathrm{Y}(r\vee p))$	$\neg\exists[(\neg p\wedge\neg r)\mathrm{Y}(s\wedge\neg p\wedge\neg r\wedge\exists\circ(\exists[\neg r\mathrm{Y}(t\wedge\neg r)]))]$
	Après q	$(\Box\neg q)\vee((\neg q)\mathrm{Y}(q\wedge((\Diamond(s\wedge\circ\Diamond t))\Rightarrow((\neg s)\mathrm{Y}p)))$	$\neg\exists[\neg q\mathrm{Y}(q\wedge\exists[\neg p\mathrm{Y}(s\wedge\neg p\wedge\exists\circ(\exists\Diamond(t)))])]$
	Entre q et r	$\Box((q\wedge\Diamond r)\Rightarrow((\neg(s\wedge(\neg r)\wedge$ $\circ(\neg r\mathrm{Y}(t\wedge\neg r))))\mathrm{Y}(r\vee p)))$	$\forall\Box(q\Rightarrow\neg\exists[(\neg p\wedge\neg r)\mathrm{Y}(s\wedge\neg p\wedge\neg r\wedge$ $\exists\circ(\exists[\neg r\mathrm{Y}(t\wedge\neg r\wedge\exists\Diamond(r))])])])$
	De q à r	$\Box(q\Rightarrow(\neg(s\wedge(\neg r)\wedge\circ(\neg r\mathrm{Y}(t\wedge\neg r)))$ $\mathrm{Y}(r\vee p)\vee\Box(\neg(s\wedge\circ\Diamond t))))$	$\forall\Box(q\Rightarrow\neg\exists[(\neg p\wedge\neg r)\mathrm{Y}(s\wedge\neg p\wedge\neg r\wedge$ $\exists\circ(\exists[\neg r\mathrm{Y}(t\wedge\neg r)])])])$
Response chain	Partout	$\Box(p\Rightarrow\Diamond(s\wedge\circ\Diamond t))$	$\forall\Box(p\Rightarrow\forall\Diamond(s\wedge\forall\circ(\forall\Diamond(t))))$
	Avant r	$\Diamond r\Rightarrow(p\Rightarrow(\neg r\mathrm{Y}(s\wedge\neg r\wedge\circ(\neg r\mathrm{Y}t))))\mathrm{Y}r$	$\neg\exists[\neg r\mathrm{Y}(p\wedge\neg r\wedge(\exists[\neg s\mathrm{Y}r]\vee$ $\exists[\neg r\mathrm{Y}(s\wedge\neg r\wedge\exists\circ(\exists[\neg t\mathrm{Y}r]))])))]$
	Après	$\Box(q\Rightarrow\Box(p\Rightarrow(s\wedge\circ\Diamond t)))$	$\neg\exists[\neg q\mathrm{Y}(q\wedge\exists\Diamond(p\wedge(\exists\Box(\neg s)\vee$

	q		$\exists\Diamond(s\wedge \exists\circ(\exists\Box(\neg t))))))]$
	Entre q et r	$\Box((q\wedge\Diamond r)\Rightarrow(p\Rightarrow(\neg r\,\mathrm{Y}(s\wedge\neg r\wedge\circ(\neg r\,\mathrm{Y}t))))\mathrm{Y}r)$	$\forall\Box(q\Rightarrow\neg\exists[\neg r\,\mathrm{Y}(p\wedge\neg r\wedge(\exists[\neg s\,\mathrm{Y}r]\vee$ $\exists[\neg r\,\mathrm{Y}(s\wedge\neg r\wedge \exists\circ(\exists[\neg t\,\mathrm{Y}r]))])])])$
	De q à r	$\Box(q\Rightarrow(p\Rightarrow(\neg r\,\mathrm{Y}(s\wedge\neg r\wedge\circ$ $(\neg r\,\mathrm{Y}t))))\mathrm{Y}(r\vee\Box(p\Rightarrow(s\wedge\circ\Diamond t))))$	$\forall\Box(q\Rightarrow\neg\exists[\neg r\,\mathrm{Y}(p\wedge\neg r\wedge(\exists[\neg s\,\mathrm{Y}r]\vee$ $\exists\Box(\neg s\wedge\neg r)\vee \exists[\neg r\,\mathrm{Y}(s\wedge\neg r\wedge \exists$ $\circ(\exists[\neg t\,\mathrm{Y}r]\vee\exists\Box(\neg t\wedge\neg r)))])])])$

Conclusion

L'ingénierie modèle-centrée des systèmes a, au cours des dernières décennies, utilisé des modèles destinés à diverses formes d'analyse, telles que la simulation, les méthodes formelles et l'émulation. Les résultats de ces analyses ont souvent révélé des connaissances subtiles qui ont aidé à améliorer la compréhension de systèmes existants ou éviter des erreurs coûteuses dans la conception de nouveaux systèmes. Force est de constater que certaines questions sont mieux répondues par des méthodes d'analyse spécifiques (tels, l'évaluation des performances par la simulation, la vérification de la sécurité par les méthodes formelles, ou encore l'apprentissage de l'humain dans la boucle par l'émulation). Par conséquent, une étude exhaustive d'un système complexe nécessite l'utilisation de différentes méthodes d'analyse pour produire des réponses complémentaires. Nul doute qu'une combinaison de méthodes d'analyse offre des capacités plus grandes que n'importe laquelle des méthodes appliquée seule. A la tâche herculéenne de créer et de gérer plusieurs modèles d'un même système pour les différentes analyses combinées (dont l'épineux problème d'alignement sémantique entre les différents modèles), peut être opposée une approche alternative, i.e., celle de créer et de gérer un seul modèle pouvant servir pour les besoins de ces analyses. Ceci est possible via un formalisme possède des domaines sémantiques propices à ces analyses.

Cet ouvrage esquisse les bases de ce travail, dans une approche plus intuitive que formelle. Il inscrit ces bases dans un cycle de vie, et une séparation des préoccupations qui fragmente les barrières de complexité et les ramène à trois problématiques majeures : spécification, synthèse de code, et qualité de ces modèles. L'élaboration du langage de modélisation multi-analyse HiLLS se situe à la croisée de trois apports : celle de la théorie des systèmes, celle du génie logiciel et celle les méthodes formelles. La théorie mathématique des systèmes dynamiques fournit, à travers le paradigme DEVS, un socle de formalisation transversale à toutes les applications de M&S. Les méthodes formelles permettent l'examen, par manipulation symbolique, des propriétés temporelles des modèles élaborés. Les techniques avancées du génie logiciel permettent, elles, de réaliser le prototypage rapide de ces modèles.

Bibliographie

[Abrial 1996] Abrial J.R. 1996. The B-Book: Assigning Programs to Meanings. *Cambridge University Press*.

[Adiga & Glassey 1991] Adiga S., Glassey C.R. 1991. Object-Oriented Simulation to Support Research in Manufacturing Systems. *International Journal of Production Research* 29(12): 2529-2542.

[Aliyu & Traoré 2015] Aliyu H.O., Traoré M.K. 2015. Toward an Integrated Framework for the Simulation, Formal Analysis and Enactment of Discrete Event Systems Models. *Proceedings of the 2015 Winter Simulation Conference (WSC'15):* 3090-3091, December 6-9, 2015, Huntington Beach, CA, USA. IEEE Press.

[Aliyu et al. 2015] Aliyu H.O., Maïga O., Traoré M.K. 2015. A Framework for Discrete Events Systems Enactment. *Proceedings of 29th European Simulation and Modeling Conference (ESM'15)*: 149-156, October 26-28, 2015, Leicester, UK, EUROSIS-ETI.

[Aliyu et al. 2016] Aliyu H.O., Maïga O., Traoré M.K. 2016. The High Level Language for System Specification: A Model-Driven Approach to Systems Engineering. *International Journal of Modeling, Simulation, and Scientific Computing*, *7*(01), 1641003.

[Arango & Prieto-Diaz 1991] Arango G., Prieto-Diaz R. 1991. Domain Analysis: Concepts and Research Directions. *Domain Analysis and Software Systems Modeling*. Prieto-Diaz R. and Arango G. (Editors), IEEE Computer Press, Washington DC: 9-32.

[Armstrong et al. 1995] Armstrong R., Gu S., Li L. 1995. A Greedy Algorithm to Determine the Number of Transporters in a Cyclic Electroplating Process. *INRIA/IEEE Symposium on Emerging Technologies and Factory Automation* 1: 460-474.

[Artiba & Elmaghraby 1996] Artiba A., Elmaghraby S.E. 1996. Planning and Scheduling of Production Systems. Methodologies and Applications. *Kluwer Academic Publishers*, Boston.

[Balci 1997] Balci O. 1997. Principles of Simulation Model Validation, Verification, and Testing. *Transactions of the SCS* 13(1): 3-12.

[Baptiste et al. 1994] Baptiste P., Legeard B., Manier M-A., Varnier C. 1994. A Scheduling Problem Optimization Solved with Constraint Logic

Programming. *2nd International Conference on the practical application of Prolog*: 47-66.

[Barros & Mendes 1997] Barros F.J., Mendes M.T. 1997. Fire Modelling in the DELTA Simulation Environment. *Simulation Practice and Theory* 5: 185-197.

[Barros 1997] Barros F.J. 1997. Modelling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modelling and Computer Simulation* 7(4): 501-515.

[Basnet & Mize 1995] Basnet C., Mize J.H. 1995. A Rule-Based, Object-Oriented Framework for Operating Flexible Manufacturing Systems. *International Journal of Production Research*: 33(5): 1417-1431.

[Belkhiter et al. 1997] Belkhiter M., Gourgand M., Traoré M.K. 1997. Dynamic Scheduling and Planning of the Production in a Hybrid Flowshop: Simulation and Optimization of a Multicriterion Objective. *Proceedings of the 9th European Simulation Symposium*. Oct. 19-22, Passau, Germany: 358-362.

[Baier et al. 2008] Baier C., Katoen J.P., Larsen K.G. 2008. Principles of Model Checking. *MIT press*.

[Blazewicz et al. 1994] Blazewicz J., Ecker K., Schmidt G., Weglarz J. 1994. Scheduling in Computer and Manufacturing Systems. *Springer Verlag*.

[Borshchev & Filippov 2004] Borshchev A., Filippov A. 2004. From System Dynamics and Discrete Event to Practical Agent Based Modeling: Reasons, Techniques, Tools. *The 22nd International Conference of the System Dynamics Society*, July 25-29, Oxford, England.

[Boukerche & Zhang 2008] Boukerche A., Zhang M. 2008. Towards Peer-to-Peer Based Distributed Simulations on a Grid Infrastructure. *Proceedings of the 41st Annual Simulation Symposium*: 212-219.

[Box & Wilson 1951] Box G.E.P., Wilson K.B. 1951. On the Experimental Attainment of Optimum Conditions. Journal of the Royal Statistical Society Series B13(1): 1–45.

[Bracker & Chapman 1985] Bracker W.E., Chapman W.L. 1985. Optimization of Programmable Hoists Using Linear Programming. *Printed Circuit Fabrication* 8: 39-42.

[Broadfoot & Roscoe 2000] Broadfoot P., Roscoe B. 2000. Tutorial on FDR and its Applications. *SPIN Model Checking and Software Verification*. Springer Berlin Heidelberg: 322-322.

[Bruno 1984] Bruno G. 1984. Using Ada for Discrete Event Simulation. *Software Practice & Experience*. 14(7): 685-695.

[Bryant 1977] Bryant R.E. 1977. Simulation of Packet Communication Architecture Computer Systems. *Massachusetts Institute of Technology*. Cambridge, MA, USA.

[Buck et al. 1994] Buck J., Ha S., Lee E.A. 1994. Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *International Journal of Computer Simulation* 4: 155-182.

[Bussow et al. 1997] Bussow R., Geisler R., Grieskamp W. and Klar M. 1997. The µSZ Notation. Version 1. *Technical Report*, TU Berlin, Germany.

[Butler 1999] Butler M. CSP2B: A Practical Approach to Combining CSP and B. *FM'99 Formal Methods*: 490-508. Springer Berlin Heidelberg.

[Corbett & Avrunin 1995] Corbett J.C., Avrunin G.S. 1995. Using Integer Programming to Verify General Safety and Liveness Properties. *Formal Methods in System Design 6*(1): 97-123.

[Carlier & Chretienne 1988] Carlier J., Chretienne P. 1988. Problème d'Ordonnancement. Modélisation/Complexité/Algorithme. *Masson ERI.*

[Caux et al. 1995] Caux C., Fleury G., Gourgand M., Kellert P. Couplage Méthodes d'ordonnancement - Simulation pour l'Ordonnancement de Systèmes Industriels de Traitement de Surface. *Operations Research*. 29(4): 391-413.

[Cimatti et al. 2000] Cimatti A., Clarke E., Giunchiglia F., Roveri M. 2000. NuSMV: a New Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer 2*(4): 410-425.

[Cimatti et al. 2002] Cimatti A., Clarke E., Giunchiglia E., Giunchiglia F., Pistore M., Roveri M., Sebastiani R., Tacchella A. 2002. NuSMV 2: An Opensource Tool for Symbolic Model Checking. *International Conference on Computer Aided Verification:* 359-364. Springer Berlin Heidelberg.

[Chaudhuri et al. 2010] Chaudhuri K., Doligez D., Lamport L., Merz S. 2010. Verifying Safety Properties with the TLA+ Proof System. *International Joint Conference on Automated Reasoning*: 142-148. Springer Berlin Heidelberg.

[Cea 1968] Cea J. 1968. Les Méthodes de Descente Dans la Théorie de l'Optimisation. *Revue Française de Recherche Opérationnelle*, Série rouge. 2(3) : 79-101.

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

[Clarke et al. 1999] Clarke E.M., Grumberg O., Peled D. 1999. Model checking. *MIT press*.

[Chandy & Misra 1979] Chandy K.M., Misra J. 1979. Distributed Simulation: A case Study in Design and Verification of Distributed Programs. *IEEE transactions on Software Engineering*, 5(5) :440-452.

[Chandy & Misra 1981] Chandy K.M., Misra J. 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *ACM* 24(4): 198-206.

[Chandy et al. 1983] Chandy K.M., Misra J., Haas L. 1983. Distributed Deadlock Detection. *ACM Trans. Comput. Syst.* I(2): 144-156.

[Châne et al. 2004] Châne F., Giambiasi N., Paillet J-L. 2004. From DEVS Model to Timed Automata. *Software Engineering Research and Practice*: 498-504.

[Chen & Chu 1994] Chen H., Chu J-M. 1994. Cyclic Scheduling of a Hoist With Time Window Constraints. *INRIA Research Report* (2307).

[Cho 2001] Cho Y.K. 2001. RTDEVS/CORBA: Distributed Real-Time Object Computing Environment. *Ph.D. Dissertation*, Electrical and Computer Engineering, The University of Arizona, Tucson, AZ.

[Chow 1996] Chow A. 1996. Parallel DEVS: a Parallel, Hierarchical, Modular Modeling Formalism and its Distributed Simulator. *SCS Transaction on Simulation* 13(2): 55-102.

[Clarke & Wing 1996] Clarke E., Wing J.M. 1996. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys* 28(4): 626-643.

[Coddington 1997] Coddington P.D. 1997. Random Number Generators for Parallel Computers. *NHSE Review* 1996(2), Northeast Architectures Center.

[Colvin & Beaumariage 1998] Colvin K., Beaumariage T. 1998. Network-Centric Simulation Using NCOS. *Simulation* 70(6): 396-409.

[Comfort 1984] Comfort J.C. 1984. The Simulation of a Master-Slave Event Set Processor. *Simulation* 42(3): 117-124

[Concepcion 1989] Concepcion A.I. 1989. A Hierarchical Computer Architecture for Distributed Simulation. *IEEE Transactions on Computer* 38(2): 311-319.

[Coquillard & Hill 1997] Coquillard P., Hill D.R.C. 1997. Modélisation et Simulation d'Ecosystèmes. Des Modèles Déterministes aux simulations à Evénements Discrets. *Masson*, Paris, SA.

[Cristia 2007] Cristia M. 2007. A TLA+ Encoding of DEVS Models. *International Modeling and Simulation Multiconference*, Buenos Aires (Argentina): 17–22.

[Cubert & Fishwick 1998] Cubert M.C., Fishwick P.A. 1998. OOPM: an Object-Oriented Multimodeling and Simulation Application Framework. *Simulation* 70(6): 379-395.

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

[Dwyer et al. 1999] Dwyer M.B., Avrunin G.S., Corbett J.C. 1999. Patterns in Property Specifications for Finite-State Verification. *Proceedings of the IEEE International Conference on Software Engineering:* 411-420.

[Dacharry & Giambiasi 2007] Dacharry H.P., Giambiasi N. 2007. A Formal Verification Approach for DEVS. *Proceedings of the Summer Computer Simulation Conference*: 312-319.

[Dahman et al. 1998] Dahman J., Kuhl F., Weatherly R. 1998. Standards for Simulation: as Simple as Possible but not Simpler. The High Level Architecture for Simulation. *Simulation* 71(6): 378-387.

[Davila & Uzcagegui 2000] Davila J., Uzcategui M. 2000. GALATEA: A Multi-agent Simulation Platform. *International Conference on Modeling, Simulation and Neural Networks*. MSSN'2000.

[Davis et al. 1999] Davis W.J., Chen X., Brook A., Awad F.A. 1999. Implementing On-Line Simulation With the World Wide Web. *Simulation* 73(1): 40-54.

[Dwyer & Clarke 1994] Dwyer M.B., Clarke L.A. 1994. Data Flow Analysis for Verifying Properties of Concurrent Programs. *ACM SIGSOFT Software Engineering Notes* 19(5): 62-75.

[De Matteis & Pagnutti 1988] De Matteis A., Pagnutti S. 1988. Parallelization of Random Number Generators and Long-Range Correlations. *Numer. Math.* (53): 595-608.

[Delara & Vangheluwe 2002] De Lara J., Vangheluwe H. 2002. ATOM3: A tool for Multi-formalism Modeling and Meta-modeling. *European Joint Conference on Theory and Practice of Software*. Grenoble, France.

[Dijkstra & Scholten 1980] Dijkstra E.W., Scholten C.S. 1980. Termination Detection for Diffusing Computations. *Proc. Lett.* 12: 1-4.

[Dines & Jones 1978] Dines B., Jones C.B. 1978. The Vienna Development Method: The Meta-language. *Springer*.

[Emerson 1991] Emerson E.A. 1991. Temporal and Modal Logic. *Handbook of Theoretical Computer Science*, vol. B (Van Leeuwen J. Ed.). MIT Press, Cambridge, MA, USA.

[Ferro 1994] Ferro G. 1994. AMC: ACTL Model Checker. Reference Manual. *lnternal Report B4-47*.

[Ferber & Drogoul 1992] Ferber J., Drogoul A. 1992. Using Multi-Agent Systems in Simulation and Problem Solving. *Distributed Artificial Intelligence: Theory and Practice*, Gasser L. and Avouris N. (Editors), Kluwer: 53-80.

[Filippi et al. 2002] Filippi J.B., Bernardi F., Delhom M. 2002. The JDEVS Modeling and Simulation Environment. *Integrated Assessment and Decision Support Conference (IEMSS'02):* 283-288. Lugano, Switzerland.

[Fischer 2000] Fischer C. Combination and Implementation of Processes and Data: from CSP-OZ to Java. *PhD thesis, University of Oldenburg*.

[Fishwick 1995] Fishwick P. 1995. Simulation Model Design and Execution. Building Digital Worlds. *Prentice Hall*.

[Fishwick 1996] Fishwick P.A. 1996. Web-Based Simulation: Some Personal Observations. *Proceedings of the Winter Simulation Conference*: 772-779.

[Forester 1961] Forester J. 1961. Industrial Dynamics. *MIT Pess*, Cambridge, MA.

[Fainekos & Pappas 2008] Fainekos G.E., Pappas G.J. 2008. A User Guide for TaLiRo. *Technical report, Department of CIS, University of Pennsylvania*.

[Friesen et al. 1998] Friesen V., Nordwig A., Weber M. 1998. Object-Oriented Specification of Hybrid Systems Using UMLh and ZimOO. *Proceedings of ZUM'98*: 328-346.

[Fujimoto 1988] Fujimoto R.M. 1988. Performance Measurements of Distributed Simulation Strategies. *Proceedings of the SCS MultiConference on Distributed Simulation*: 14-20.

[Fujimoto 1990] Fujimoto R.M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10): 30-53.

[Fujimoto 2000] Fujimoto R.M. 2000. Parallel and Distributed Simulation Systems. *Wiley Series on Parallel and Distributed Computing. A.Y. Zomaya (Editor)*.

[Gamma et al. 1995] Gamma E., Helm R., Johnson R.,Vlissides J. 1995. Design patterns. Elements of Reusable Object-Oriented Software. *Addison-Wesley*.

[Garey et al. 1976] Garey M.R., Johnson D., Stockmeyer L. 1976. Some Simplified NP-Complete Graph Problems. *Theoretical Computer Science* 1: 237-267.

[Ge & Yih 1995] Ge Y., Yih Y. 1995. Crane Scheduling with Time Windows in Circuit Board Production Lines. *International Journal of production Research* 33(5): 1187-1189.

[Goranko & Galton 2015] Goranko V, Galton A. 2015. Temporal Logic. *The Stanford Encyclopedia of Philosophy*.

[Giambiasi et al. 2003] Giambiasi N., Paillet J-L., Châne F. 2003. Simulation and Verification II: From Timed Automata to DEVS PModels. *Proceedings of the Winter Simulation Conference*: 923-931.

[Godart & Perrin 2009] Godard C., Perrin O. 2009. Les Processus Métiers. Concepts, Modèles et Systèmes. *Godard & Perrin (direction). Hermes Sciences publications. Lavoisier*.

[Gordon & Melham 1993] Gordon M.J.C., Melham T.F. Introduction to HOL A Theorem Proving Environment for Higher Order Logic. *Cambridge University Press*.

[Gourgand et al. 2003] Gourgand M., Lacomme P., Traoré M.K. 2003. Design of a Monitoring Environment for Manufacturing Systems Management and Optimization. *International Journal of Computer Integrated Manufacturing*. 16(1): 61-80.

[Gravel et al. 1992] Gravel M., Martel J.M., Nadeau R., Price W., Trembley R. 1992. A Multicriterion View of Optimal Resource Allocation in Job-Shop Production. *European Journal of Operations Research,* 61: 230-244.

[Gregoriades & Karakostas 2004]. Gregoriades A., Karakostas B. 2004. Unifying Business Objects and System Dynamics as a Paradigm for Developing Decision Support Systems. *Decision Support Systems* 37(2): 307-311.

[Hanen & Munier 1994] Hanen C., Munier A. Pilotage Périodique d'une Ligne de Galvanoplastie à Plusieurs Robots. *LITP Research Report*.

[Hoare 1985] Hoare C.A.R. 1985. Communicating Sequential Processes. *Prentice Hall*.

[Harrel 1987] Harel D. 1987. Statecharts: A Visual Formalism for Complex Systems. *Science of computer programming* 8(3): 231-274.

[Hernandez & Giambiasi 2005] Hernandez A., Giambiasi N. 2005. State Reachability for DEVS Models. *Proceedings of Argentine Symposium on Software Engineering*.

[Hill 1993] Hill D. 1993. Analyse Orientée Objet et Modélisation par Simulation. *Addison-Wesley*.

[Holzmann 1997] Holzmann G.J. 1997. The Model Checker SPIN. *IEEE Transactions on Software Engineering* 23(5): 279.

[Hong et al. 1997] Hong J., Song H., Kim T., Park K. 1997. A Real-Time Discrete-Event System Specification Formalism for Seamless Real-Time Software Development. *Discrete Event Systems: Theory and Applications* 7: 355–375.

[Hunsucker & Shah 1992] Hunsucker J.L., Shah J.R. 1992. Performance of Priority Rules in a Due Date Flowshop. *OMEGA* 20(1): 73-89.

[Hwang & Zeigler 2009] Hwang M.H., Zeigler B.P. 2009. Reachability Graph of Finite and Deterministic DEVS Networks. *IEEE Transactions on Automation Science And Engineering* 6 (3).

[Ingels & Raynal 1990] Ingels P., Raynal M. 1990. Simulation Répartie: Schémas d'Exécution Pour un Modèle à Processus. *Technique et Science Informatiques:* 383-397.

[Janousek et al. 2006] Janousek V., Polásek P., Slavícek P. 2006. Towards DEVS Meta Language. *ISC*: 69-73. Zwijnaarde, BE.

[Jefferson 1985] Jefferson D.R. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems* 7(3): 404-425.

[Kellert et al. 1997] Kellert P., Tchernev N., Force C. 1997. Object-Oriented Methodology for FMS Modelling and Simulation. *International Journal of Computer Integrated Manufacturing* 10(6): 405-434.

[Kiviat 1967] Kiviat P.J. 1967. Digital Computer Simulation: Modeling Concepts. *RM-5378-PR*, RAND Corporation, Santa Monica, CA.

[Klir 1969] Klir G.J. 1969. An Approach to General Systems Theory. *Van Nostrand Reinhold*, New York.

[Kofman et al. 2003] Kofman E., Lapadula M., Pagliero E. 2003. PowerDEVS: a DEVS-based Environment for Hybrid System Modeling and Simulation. *Technical Report LSD0306*, University Nacional de Rosario.

[Kozen 1983] Kozen D. 1983. Results on the Propositional μ-Calculus. *Theoretical computer science 27*(3): 333-354.

[Kreutzer 1986] Kreutzer W. 1986. System Simulation Programming Languages and Styles. *Addison-Wesley*, Reading, MA.

[Krishnamurthi et al. 1985] Krishnamurthi M., Chandrasekaran U., Sheppard S. 1985. Two Approaches to the Implementation of a Distributed Simulation System. *Proceedings of the 17th Winter Simulation Conference* 42(3), San Francisco, CA: 435-444

[Kuhn et al. 2003] Kuhn D.R., Craigen D., Saaltink M. 2003. Practical Application of Formal Methods in Modeling and Simulation. *Proceedings of SCSC'03 (invited). Summer Simulation Multiconference*, Montreal, Canada.

[Kusiak & Chen 1988] Kusiak A., Chen M. 1988. Expert Systems for Planning and Scheduling Manufacturing Systems. *European Journal of Operations Research,* 34 :113-130.

[Kwon et al. 1996] KwonY., Park H., Jung S., Kim T. 1996. Fuzzy-DEVS Formalism: Concepts, Realization and Applications. *AI, Simulation and Planning in High AutonomySystems* EPD, University of Arizona, San Diego.

[Lackner 1964] Lackner M.R. 1964. Digital Simulation and System Theory. *System Development Corportion SDC SP-1612*, Santa Monica CA, 6 April.

[Lacoste & Baptiste 1992] Lacoste M.A., Baptiste P.A. 1992. Constraint Logic Programming Approach for the Nperiodic Hoist Scheduling Problem. *3rd International Workshop on Project Management and Scheduling*.

[Lamport 1977] Lamport L. 1977. Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering* (2): 125-143.

[Lamport 1994] Lamport L. 1994. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16(3): 872-923.

[Lamothe et al. 1995] Lamothe J., Correge C., Delmas J. 1999. A Dynamic Heuristic for the Real Time Hoist Scheduling Problem. *Conference on Emerging Technologies and Factory Automation*: 161-168.

[Lamothe et al. 1996] Lamothe J., Thierry C., Delmas J. 1996. A Multihoist Model for the Real Time Hoist Scheduling Problem. *IMACS Symposium on Discrete Events and Manufacturing Systems*: 461-466.

[Lamsweerde 2000] Lamsweerde A.V. 2000. Formal Specification: a Roadmap. *International Conference on Software Engineering*. *Proceedings of the Conference on the Future of Software Engineering*: 147-159.

[Langlois & Phipps 1997] Langlois A., Phipps M. 1997. Automates Cellulaires. Application à la Simulation Urbaine. *Editions Hermes*, Paris.

[Larsen et al. 1997] Larsen K.G., Pettersson P., Wang Y. 1997. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)* 1(1): 134-152.

[Lei & Wang 1989] Lei L., Wang T.J. A Proof: the Cyclic Scheduling Problem is NP-complete. *OSM Working Paper* 89-0016, Rutgers University.

[Lei & Wang 1991] Lei L, Wang T.J. 1991. The Minimum Common Cycle Algorithm for Cycle Scheduling of Two Material Handling Hoists With Time Window Constraint. *Management Science* 37(12): 1629-1639.

[Leroudier 1980] Leroudier J. 1980. La Simulation à Evénements Discrets. *Editions Hommes et Techniques*.

[Lewis 1993] Lewis R.O. 1993. Verification, Validation and Accreditation (VV&A) of Models and Simulations Used in Distributed Interactive Environment. *Proceedings of European Simulation Multiconference on Modelling and Simulation*: 633-636.

[Low et al. 1999] Low Y-H., Lim C-C., Cai W., Huang S-Y., Hsu W-J., Jain S., Turner S.J. 1999. Survey of Languages and Runtime Librairies for Parallel Discrete-Event Simulation. *Simulation* 72(3): 170-186.

[Lubachevsky & Weiss 1990] Lubachevsky B., Weiss A. 1991. An Analysis of Rollback-Based Simulation. *ACM Transactions on Modeling and Computer Simulation* 1(2): 154-193.

[Mac Carthy & Liu 1993] Mac Carthy B.L., Liu J. 1993. Adressing the Gap in Scheduling Research : a Review of Optimization and Heuristic Methods in Production Scheduling. *International Journal of Operations Research,* 31(1) : 59-79.

[MacDougall 1987] MacDougall M.H. 1987. Simulating Computer Systems: Techniques and Tools. *MIT Press*, Cambridge.

[Mahzer 2000] Mazher A.K. 2000. Toward A Unified Approach to Modeling and Computer Simulation of Social Systems. Part I: Methodology of Model Construction. *Workshop on norms and institutions in multi-agent*, Barcelona, Spain, June 3-4.

[Maïga et al. 2015] Maïga O., Aliyu H.O., Traoré M.K. 2015. A New Approach to Modeling Dynamic Structure Systems. Proceedings of 29th *European Simulation and Modeling Conference (ESM'15*): 141-148*,* October 26-28, 2015, Leicester, UK, EUROSIS-ETI.

[Manier & Baptiste 1994] Manier M-A., Baptiste P. Etat de l'Art: Ordonnancement de Robot de Manutention en Galvanoplastie. *APII* 28(1): 7-35.

[McHaney 1999] McHaney R. 1999. Integration of the Genetic Algorithm and Discrete-Event Computer Simulation for Decision Support. *Simulation* 72(6): 401-411.

[McMillan 1992] McMillan K.L. 1992.Symbolic Model Checking. An Approach to the State Explosion Problem. *Ph.D. Thesis in Computer Science, CMU-CS-92-131*, Carnegie Mellon University.

[Melliar-Smith 1988] Melliar-Smith P.M. 1988. A Graphical Representation of Interval Logic. *International Conference on Concurrency:* 106-120. Springer Berlin Heidelberg.

[Merkuryeva & Merkuryev 1994] Merkuryeva G., Merkuryev Y. 1994. Knowledge-Based Simulation Systems – A Review. *Simulation* 62(2): 74-89.

[Merkuryeva & Merkuryev 1999] Merkuryeva, G., Merkuryev Y. 1999. Metamodelling in Computer Simulation: State of Art. Preprints of the Advanced Summer Institute'99 on Life Cycle Approaches to Production Systems: Management, Control, and Supervision, Lueven, Belgium.

[Miller & Thorpe 1983] Miller D.C. and Thorpe J.A. 1983. SIMNET: The Advent of Simulator Networking. *IEEE* 83(8): 1114-1123.

[Milner 1980] Milner R. 1980. A Calculus of Communicating Systems. *Springer Verlag*.

[Misra 1986] Misra J. 1986. Distributed Discrete Event Simulation. *Computing Surveys* 18(1): 39-65.

[Mittal et al. 2007] Mittal S., Risco-Martín J.L., Zeigler B.P. 2007. DEVSML: Automating DEVS Execution Over SOA Towards Transparent Simulators. *Spring Simulation Multiconference*, Norfolk, Virginia, USA: 287-295.

[Mokkarala et al. 1985] Mokkarala V.R., Fan A., Apte R. 1985. A Unified Approach to Simulation and Timing Verification at the Functional Level. *Proceedings of the 22nd ACM/IEEE Design Automation Conference*: 757-776.

[Monsef 1997] Monsef Y. 1997. *Modelling and Simulation of Complex Systems. Concepts, Methods and Tools. Frontiers in Simulation*, Kerckhoffs E., Lehmann A., Pierreval H. and Zobel R. (Editors), SCS International.

[Markey & Schnoebelen 2004] Markey N., Schnoebelen P. 2004. TSMV: A Symbolic Model Checker for Quantitative Analysis of Systems. *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST'04):* 330-331. IEEE Computer Society Press.

[Muzy et al. 2003] Muzy A., Innocenti E., Barros F. Aïello A., Santucci J.F. Efficient Simulation of Large-scale Dynamic Structure Cell Spaces. *Proceedings of SCSC'03*, Montréal: 378-383.

[Meyers et al. 2013] Meyers B., Wimmer M., Vangheluwe H., Denil J. 2013. Towards Domain-Specific Property Languages: The ProMoBox Approach. *Proceedings of the ACM Workshop on Domain-specific Modeling:* 39-44.

[Nance 1981] Nance R.E. 1981. The Time and State Relationships in Simulation Modeling. *Communications of the ACM*: 24(4): 173-179.

[Nance 1987] Nance R.E. 1987. The Conical Methodology: a Framework for Simulation Model Development. *SCS Simulation Series*: 19(1): 38-43, Orlando, FL, April 6-9.

[Nandy & Loucks 1993] Nandy B., Loucks W.M. 1993. On a Parallel Partitioning Technique for Use With Conservative Parallel Simulation. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation* 23(1), May 16-19, San Diego. Bagrodia R. and Jefferson D. (Editors): 43-51.

[Ng 1996] Ng W.C. A Branch-and-bound Algorithm for Hoist Scheduling of a Circuit Board Production Line. *International Journal of Flexible Manufacturing Systems* 8(1): 45-65.

[Nutaro 2010] Nutaro J. 2010. Building Software for Simulation: Theory, Algorithms, and Applications in C++. *Wiley*. ISBN 0-470-41469-3.

[Nygaard & Dahl 1978] Nygaard K., Dahl O-J. 1978. The Development of the SIMULA Languages. *History of Programming Languages I:* 439-480, ACM, New York, NY.

[O'Keefe 1986] O'Keefe R. 1986. Simulation and Expert Systems – A Taxonomy and Some Examples. *Simulation* 46(1): 10-16.

[OMG 2000] Object Management Group. 2000. The Common Object Request Broker: Architecture and Specification. *Object Management Group, 2.4 edition*.

[OMG 2003] Object Management Group. 2003. The Model-Driven Architecture, Guide Version 1.0.1. *OMG/2003-06-01*.

[Ören 1989] Ören T.I. 1989. A Paradigm for Artificial Intelligence in Software Engineering. *Advances in Artificial Intelligence in Software Engineering* Ören T.I. (Editor), Vol. 1, JAI Press.

[Overstreet & Nance 1986] Overstreet C.M., Nance R.E. 1986. World View Based Discrete Event Model Simplification. *Modeling and Simulation Methodology in the Artificial Intelligence Era*. Elzas M., Ören T. and Zeigler B.P. (Editors): 165-179. North-Holland.

[Overstreet et al. 1994] Overstreet C.M., Page E.H., Nance R.E. 1994. Model Diagnosis Using The Condition Specification: From Conceptualization to Implementation. *Winter Simulation Conference*: 566-573.

[Owre et al. 1992] Owre S., Rushby J.M., Shankar N. 1992. PVS: A Prototype Verification System. *Automated Deduction—CADE-11:* 748-752. Springer Berlin Heidelberg.

[Page et al. 1997] Page E.H., Moose R.L. Jr., Griffin S.P. 1997. Web-Based Simulation in Simjava Using Remote Method Invocation. *Web-based Simulation at the Winter Simulation Conference*, Atlanta, GA, 7-10 December: 468-474.

[Paige 1997] Paige R.F. 1997. A Meta-Method for Formal Method Integration. *Proceedings of the 4th International Symposium of Formal Methods Europe*: 473-494.

[Paulson 1993] Paulson L.C. 1993. The Isabelle Reference Manual. *University of Cambridge, Computer Laboratory*.

[Pawlowski 2003] Pawlowski K. 2003. Do Not Trust All Simulation Studies of Telecommunication Networks. *Information Networking*: 899-908. Springer.

[Phillips & Unger 1976] Phillips L.W., Unger P. Mathematical Programming Solution of a Hoist Scheduling Program. *AIIE Transactions* 8(2): 219-225.

[Posse 2008] Posse E. 2008. Modeling and simulation of dynamic structure discrete-event systems. *PhD Thesis, MCGill University*.

[Powers & Nute 1984] Powers W.S., Nute T. 1984. Implementing a Simulator as a Set of Ada Tasks. *Proceedings of the Winter Simulation Conference*: 7-17.

[Praehofer et al. 1999] Praehofer H., Sametinger J., Stritzinger A. 1999. Discrete Event Simulation Using the JavaBeans Component Model. *International Conference on Web-Based Modeling & Simulation*. San Francisco, CA. USA.

[Pratt et al. 1994] Pratt D.B., Farrington P.A., Basnet C.B., Bhuskute H.C., Kamath M., Mize J.H. 1994. The Separation of Physical, Information and Control Elements for Facilitating Reusability in Simulation Modeling. *International Journal of Computer Simulation* 4(3): 327-342.

[Pritsker 1979] Pritsker A. Compilation of Definitions of Simulation. *Simulation* 33:61-63.

[Proust & Grünenberger 1994] Proust C., Grünenberger E. 1994. Ariane 2000 : Planification de Production Dans un Contexte de Flowshop Généralisé à Deux Etages. *Journée d'Etude Ordonnancement et Entreprise:* 289-304. Toulouse, 16-17 juin.

[Rabelo et al. 2005] Rabelo L., Helal M., Jones A., Min H-S. 2005. Enterprise Simulation: a Hybrid System Approach. *International Journal of Computer Integrated Manufacturing* 18(6): 498-508.

[Raina & Muthukumar 2009] Raina A., Muthukumar V. 2009. Fuse-N: Framework for Unified Simulation Environment for Network-on-Chip.

Proceedings of the 6th International Conference on Information Technology: New Generations: 71-876.

[Reed & Roscoe 1986] Reed G.M., Roscoe A.W. 1986. A Timed Model for Communicating Sequential Processes. *Automata, Languages and Programming:* 314-323. Springer Berlin Heidelberg.

[Reuillon et al. 2008] Reuillon R., Hill D.R.C., El Bitar Z. and Breton V. 2008. Rigorous Distribution of Stochastic Simulations Using the DistMe Toolkit. *IEEE Transactions on Nuclear Science* 55(1): 595-603.

[Righter & Walrand 1989] Righter R. and Walrand J.C. Distributed Simulation of Discrete Event Systems, *Proceedings of the IEEE*: 99-113

[Rinnooykan 1976] Rinnooykan A.H.G. 1976. Machine Scheduling Problems: Classification, Complexity and Computations. *Nijhof* , The Hague.

[Roberts & Dessouky 1998] Roberts C.A. and Y.M. Dessouky. 1998. An Overview of Object-Oriented Simulation. *Simulation*: 70(6): 359-368.

[Robinson 1999] Robinson S. 1999. Simulation Verification, Validation and Confidence: a Tutorial. *Transaction* 16(2): 63-69.

[Rumbaugh et al. 1999] Rumbaugh J., Jacobson I., Booch G. 1999. The Unified Modeling Language Reference Manual. *Addison-Wesley*.

[Saadawi & Wainer 2009] Saadawi H., Wainer G. 2009. Verification of Real-Time DEVS Models. *Proceedings of SpringSim Multi Simulation Conference*, San Diego, CA March.

[Saaltink 1997] Saaltink M. 1997. The Z/EVES system. ZUM'97: Z Formal Specification Notation. *Lecture Notes in Computer Science 1212*. Springer-Verlag: 72–85.

[Sadowski 1989] Sadowski R.P. 1989. The Simulation Process: Avoiding the Problems and Pitfalls. *Proceedings of the Winter Simulation Conference*: 72-79.

[Sarjoughian & Zeigler 1998] Sarjoughian H.S., Zeigler B. 1998. DEVSJAVA: Basis for a DEVS-based Collaborative M&S Environment. *International Conference on Web-Based Modeling and Simulation* 5:29-36. San Diego, CA. USA.

[Schneider & Treharne 2002] Schneider S., Treharne H. 2002. Communicating B machines. *ZB 2002: Formal Specification and Development in Z and B*. Springer Berlin Heidelberg.

[Schöf 1995] Schöf S. 1995. Quality Considerations for Mapping Strategies of Dynamic Models in Time Warp Simulation. *Proceedings of the 7th European Simulation Symposium,* October 26-28, Erlangen-Nuremberg: 445-449.

[Shanon 1975] Shannon R. E. 1975. *System Simulation: The Art and Science*, Prentice-Hall.

[Shapiro & Nuttle 1988] Shapiro G.W., Nuttle H.LW. Hoist Scheduling for a PCB Electroplating Facility. *IEEE Transactions* 20(2): 157-167.

[Sheppard et al. 1984] Sheppard S, Friel P., Reese D. 1984. Simulation in Ada: An Implementation of Two World Views. *Proceedings of the Winter Simulation Conference*: 3-6.

[Simon 1969] Simon H.A. 1969. The Sciences of the Artificial. *MIT Press.* Cambridge, MA.

[Smith 1992] Smith G. 1992. An Object-Oriented Approach to Formal Specification. *Doctoral dissertation*, University of Queensland.

[Smith 2000] Smith R. 2000. Simulation. The Engine Behind the Virtual World. *Simulation 2000 Series* (1).

[Solcany et al. 1995] Solcany V., Skultety R., Safarik J. 1995. Simulation Model Decomposition in Conservative Parallel Discrete Event Simulation. *Proceedings of the European Simulation Multiconference on Modelling and Simulation:* 41-45.

[Sommerville 2006] Sommerville I. 2006. Software Engineering. *Addison-Wesley Educational Publishers Inc*.

[Song & al. 1993] Song W., Zabinsky Z.B. & Storch R.L. An Algorithm for Scheduling a Chemical Processing Tank Line. *Production Planning and Control* 4(4): 323-332.

[Spivey 1992] Spivey J.M. 1992. The Z Notation: A Reference Manual. *Prentice Hall*.

[Stevenson 2003] Stevenson D.E. 2003. From DEVS to Formal Methods: a Categorical Approach. *Proceedings of SCSC'03, Summer Simulation Multiconference*, Montreal, Canada: 1-6.

[Stopper & Böszörmenyi 1995] Stopper A., Böszörmenyi L. 1995. Acceleration of Distributed, Object-Oriented Simulations Using a Graph-Optimizing Approach. *Proceedings of the 7th European Simulation Symposium.* Erlangen-Nuremberg, October 26-28: 387-391.

[Thesen & Lei 1986] Thesen A., Lei L. An Expert System for Scheduling Robots in a Flexible Electroplating System With Dynamically Changing Workloads. *Proceedings of the 2nd ORSA-TIMS Conference on FMS: Operational Research Models and Applications*: 555-566.

[Thesen & Lei 1990] Thesen A., Lei L. An Expert Scheduling System for Material Handling Hoists. *Journal of Manufacturing System* 9(3): 247-252.

[Tocher 1963] Tocher K.D. 1963. The Art of Simulation. *Hodder and Stoughton, London*.

[Traoré 2006] Traoré M.K. 2006. Making DEVS Models Amenable to Formal Analysis. *Proceedings of the 2006 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*. Society for Computer Simulation International.

[Traoré & Hill 2001 b] Traoré M.K., Hill D.R.C. The Use of Random Number Generators for Stochastic Distributed Simulation: Application to Ecological Modeling. *Proceedings of the 13th European Simulation Symposium, Simulation in Industry*. Marseille, France, 18-20 Octobre: 555-559.

[Traoré & Muzy 2006] Traoré M.K., Muzy A. Capturing the Dual Relationship Between Simulation Models And Their Context. *Simulation Modelling Practice and Theory (SIMPRA)*. 14(2): 126-142.

[Traore 2008] Traoré, M.K. 2008. SimStudio. A Next Generation Modeling and Simulation Framework. *SIMUTools'08*, ISBN 978-963-9799-20-21, Marseille, France, March 3-7.

[Turner & Baker 2008] Turner R.C., Baker M.J. A Unified Simulation Approach to Structural System Reliability Analysis. *Probabilistic Mechanics and Structural and Geotechnical Reliability*: 104-107.

[Uhrmacher 2001] Uhrmacher A.M., 2001. Dynamic Structures in Modeling and Simulation: a Reactive Approach. *TOMACS*, 11(2): 206-232.

[Van Looveren et al. 1986] Van Looveren A.J., Gelders L.F., Van Wassenhove L.N. 1986. A Review of FMS Planning Models. *Modelling and Design of Flexible Systems* (Kusiak Ed.), *Elsevier Science Publishers*.

[Vangheluwe & De Lara 2002] Vangheluwe H.L., De Lara J. 2002. Meta-Models Are Models Too. *Proceedings of the 2002 Winter Simulation Conference*, IEEE Computer Society Press, San Diego, CA: 597-605.

[Vangheluwe 2000] Vangheluwe, H.L. 2000. DEVS as a Common Denominator for Multi-formalism Hybrid Systems Modeling. *IEEE International Symposium on Computer-Aided Control System Design*, ed. Varga, 129-134. IEEE Computer Society Press. Anchorage, Alaska.

[Vangheluwe et al. 2002] Vangheluwe H.L., De Lara J., Mosterman P.J. 2002. An Introduction to Multi-Paradigm Modeling and Simulation. *Proceedings of the 2002 AI Simulation and Planning in High Autonomy Systems*, Lisbon, Portugal: 9-20.

[Varnier 1996] Varnier C. Extensions du Hoist Scheduling Problem Cyclique - Résolution Basée sur un Traitement des Contraintes Disjonctives en

Programmation en Logique avec Contraintes. *Thèse de Doctorat*, Université de Franche-Comté (Besançon).

[Varnier et al. 1995] Varnier C., Grunder O., Baptiste P. Improving the Productivity of Electroplating Lines by Changing the Layout of the Tanks. *IEEE Conference on Emerging Technologies and Factory Automation* 2: 441-450.

[Wainer 2002] Wainer G. 2002. CD++: a Toolkit to Develop DEVS Models. *Software – Practice and Experience* 32: 1261-1306.

[Weilkiens 2008] Weilkiens T. 2008. Systems Engineering With SysML/UML. *Morgan Kaufmann Publishers, ISBN 0123742749*.

[Weisel et al. 2005] Weisel E.W., Petty M.D., Mielke R.R. 2005. A Comparison DEVS Semantic Composability Theory. *Proceedings of the Spring Simulation Interoperability Workshop.*

[Wild & Pignatiello 1994] Wild R.H.,Pignatiello J.J. 1994. Finding Stable System Designs: a Reverse Simulation Technique. *Communications of the ACM* 17(10): 87-98.

[Yih 1990] Yih Y. Trace-driven Knowledge Acquisition (TDKA) for Rule-based Real Time Scheduling Systems. *Journal of Intelligent Manufacturing* 1: 217-229.

[Young & Rato 2008] Young P.C, Ratto M. 2008. A Unified Approach to Environmental Systems Modeling. *Stochastic Environmental Research and Risk Assessment*.

[Zeigler & Chi 1992] Zeigler B.P., Chi S.D. 1992. Symbolic Discrete Event System Specification. *IEEE Transaction on Systems, Man and Cybernetics*, December: 1428-1443.

[Zeigler & Kim 1993] Zeigler B.P., Kim J. 1993. Extending the DEVS-Scheme Knowledge-Based Simulation Environment for Real-Time Event-Based Control. *IEEE Transaction on Robotics and Automation* 9(3): 351-356.

[Zeigler 1976] Zeigler B.P. 1976. Theory of Modeling and Simulation. *Wiley-Interscience*, New York.

[Zeigler 1984] Zeigler B.P. 1984. Multifaceted Modeling and Discrete Event Simulation. *Academic Press*.

[Zeigler et al. 1996] Zeigler B., Moon Y., Kim D., Kim J.G. 1996. DEVS-C++: A High Performance Modeling and Simulation Environment. *The 29th Hawaii International Conference on System Sciences*.

[Zeigler et al. 2000] Zeigler B.P., Kim T.G., Praehofer H. 2000. Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. *Academic Press, 2nd Edition*, New York.

[Zhang & Zeigler 1989] Zhang G., Zeigler B.P. 1989. DEVS-Scheme Supported Mapping of Hierarchical Models Onto Multiple Processor Systems. *Proceedings of the SCS Multiconference on Distributed Simulation* 21(2): 64-69.