Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes

Benjamin Camus

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes

THÈSE

présentée et soutenue publiquement le 27 Novembre 2015

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Benjamin CAMUS

Composition du jury

Président :	Abderrafiâa Koukam	
Rapporteurs :	Bernard Espinasse	Professeur, Aix-Marseille Université
	René Mandiau	Professeur, Université de Valenciennes et du Hainaut-Cambrésis
Examinateurs :	Abderrafiâa Koukam	Professeur, Université de Technologie de Belfort-Montbéliard
	Claire Gardent	Directeur de Recherche, CNRS, LORIA, Nancy
	Christine Bourjot	Maître de Conférences, Université de Lorraine
Directeur de thèse :	Vincent Chevrier	Professeur, Université de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Tout d'abord, je ne remercierai jamais assez mes encadrants Vincent Chevrier et Christine Bourjot pour leur confiance, leur disponibilité, leur implication et leur pédagogie qui ont permis de rendre ce sujet de recherche si passionnant. Merci pour ces années de recherche si riches.

Je veux également remercier d'une part Messieurs Bernard Espinasse et René Mandiau d'avoir accepté de lire et rapporter ce manuscrit, et d'autre part Monsieur Abderrafîâa Koukam et Madame Claire Gardent d'avoir accepté de faire partie du jury de cette thèse.

J'ai une pensée toute particulière pour mes deux amis et collègues de bureau, Messieurs Julien Vaubourg et Yannick Presse (l'ordre d'apparence ici ne dénotant en rien une quelconque préférence, n'en déplaise aux personnes concernées) qui m'ont supporté durant ces années de thèse et ont créé un environnement de travail dynamique et chaleureux que je ne suis pas prêt d'oublier. Cette thèse doit également beaucoup aux personnes ayant participé directement ou indirectement aux développements du logiciel MECSYCO, notamment Laurent Ciarletta, Julien Siebert, Thomas Paris, Victorien Elvinger, Alexandre Tan, Virginie Galtier, Jean-Philippe Tavella, Mathieu Caujolle, Pierre-Olivier Brissaud et Benjamin Vouillaume.

Un grand merci également à l'ensemble de mes collègues du LORIA : Raphaël Cherfan, Mihai Andries, Thomas Moinel, Maxime Rio, Nicolas Beaufort, Iñaki Fernandez, Xuan-Son Nguyen, Benoit Chappet-de-Vangel, Nassim Kaldé, Amandine Dubois, Mohamed Tlig, Arsène Fansi-Tchango, Olivier Buffet, Nazim Fatès, Alain Dutech, Vincent Thomas, Jilles Dibangoye, Olivier Bouré, Tomás Navarrete, Jano Yazbeck et Sylvain Castagnos.

Enfin, je témoigne toute ma reconnaissance à mes amis et ma famille pour leur soutien sans faille.

À Charlotte, Pierre et Patricia.

Sommaire

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Contexte de la thèse : modélisation et simulation des systèmes complexes

1.1.1 Les systèmes complexes

Ce travail de thèse porte sur la modélisation et la simulation numérique (M&S) des systèmes complexes. Ces systèmes constituent un type particulier de systèmes dynamiques définis comme étant *« composés d'un grand nombre d'entités hétérogènes en interaction, créant de par leurs interactions locales plusieurs niveaux de structures et d'organisations »* [Chavalarias et al., 2009].

Les systèmes complexes peuvent être naturels mais également artificiels. Dans la nature ils correspondent, entre autre, à des colonies d'insectes (e.g. fourmis, abeilles, termites) [Camazine et al., 2001], au corps humain ou encore à des phénomènes de déplacements collectifs (e.g. mouvements de foules, nuées d'oiseaux). A l'opposé, les villes [Batty, 2009, Bretagnolle et al., 2006], les réseaux de transport ou encore les réseaux électriques intelligents (Smart Grids) sont des exemples de systèmes complexes produits par l'homme.

La science des systèmes complexes s'attaque à deux tâches différentes mais complémentaires : d'un côté comprendre le fonctionnement de systèmes pré-existants, et de l'autre maîtriser la conception et contrôler l'évolution des systèmes complexes. Elle va par exemple chercher à répondre à des questions comme : Comment se forme une foule à partir des comportements individuels d'un ensemble de piétons ? Quels processus biologiques favorisent l'apparition de métastases dans le corps humain ? Comment concevoir un smart grid gérant de manière efficace et robuste la production d'électricité au niveau national ?

Pour répondre à ces questions, cette science va se heurter à plusieurs difficultés inhérentes à la complexité des systèmes qu'elle étudie. Les systèmes complexes ont en effet les caractéristiques suivantes [Chazelle, 2012] :

Ils sont difficiles à comprendre. Si cette caractéristique est la plus évidente, elle est toutefois subjective, et n'est donc pas suffisante pour qu'un système soit qualifié de système complexe (i.e. un système peut être difficile à comprendre sans être pour autant un système complexe).

Ils sont difficiles à prédire. Les systèmes complexes peuvent exhiber un comportement chaotique, c'est à dire qu'ils peuvent avoir une très forte sensibilité aux conditions initiales et une dynamique non linéaire [Charrier, 2009]. L'évolution d'un système complexe peut alors être sujette à des brusques changements de phase (e.g. se stabiliser subitement) dont les occurrences sont difficilement prévisibles.

Ils sont difficiles à décrire. En effet, la description d'un système complexe nécessite la prise en compte d'un grand nombre d'entités et d'interactions hétérogènes.

Ces entités sont, de plus, réparties sur différents niveaux imbriqués hiérarchiquement et évoluant à des échelles temporelles et spatiales différentes [Lane, 2006]. Ainsi, une ville peut être décomposée successivement en arrondissements, quartiers, immeubles et appartements [Pun-Cheng, 2000], alors que l'on peut décrire le corps humain en suivant une décomposition en organes, tissus, cellules, organites et molécules.

Plusieurs décompositions hiérarchiques sont en fait possibles pour un même système complexe, car ces systèmes offrent une multiplicité de points de vue. On peut par exemple décrire une ville comme un réseau de transport, un réseau électrique, une société ou encore une économie. Décrire un système complexe implique alors de prendre en compte ces différentes perspectives, chacune d'elles pouvant être relative à une discipline scientifique différente, et donc nécessitant une expertise spécifique [Anderson et al., 1972].

En réaction à ces difficultés, le cheminement emprunté pour étudier les systèmes complexes est en général celui de l'expérimentation car il permet de progresser face à la complexité par une suite d'essais et d'erreurs. Cependant, cette expérimentation va être rapidement limitée car elle va se heurter à des barrières de coût, de temps ou d'éthique.

Il serait par exemple inenvisageable de déclencher un mouvement de panique dans un stade bondé pour étudier les comportements de foule dans cette situation. De la même manière, on ne saurait déclencher une ou plusieurs pannes à l'heure de pointe dans le réseau de transport d'une grande agglomération simplement pour étudier la robustesse de ce dernier.

1.1.2 Modélisation et simulation

La démarche de modélisation et simulation (M&S) est un processus permettant l'étude d'un système dynamique à partir de modèles. Ces modèles sont des simplifications du système qui peuvent être manipulées à la place de ce dernier pour contourner les limites de l'expérimentation. En s'affranchissant de ces contraintes, la démarche apparaît comme un outil de choix pour étudier les systèmes complexes car elle permet de tester rapidement différents scénarios et alternatives.

La démarche de M&S est dirigée par un questionnement précis (e.g. Combien de pannes simultanées peut supporter ce réseau de transport ?), et son objectif est de fournir, grâce à la simulation, des réponses qui soient exploitables, c'est à dire transposables du modèle au système réel. Pour atteindre ce but, la démarche de M&S permet d'inscrire les activités de modélisation et simulation dans un cadre scientifique rigoureux. Elle s'attache en effet à prouver que le comportement du modèle est suffisamment similaire à celui du système étudié pour pouvoir en tirer des réponses exploitables. Elle cherche également à assurer la reproductibilité des expériences faites sur les modèles (i.e. les simulations), en décrivant d'une part les modèles de manière non ambigu,

et en vérifiant d'autre part que le dispositif expérimental (i.e. l'implémentation du modèle et sa simulation) n'introduit pas de biais dans les résultats. Elle s'assure alors que les résultats de simulation sont uniquement fonction des choix de modélisation.

1.2 Défis scientifiques et contributions

Lorsque la démarche de M&S est appliquée à l'étude des systèmes complexes, plusieurs défis scientifiques spécifiques sont rencontrés.

En effet, la plupart des questionnements sur ces systèmes nécessitent de prendre en compte plusieurs points de vue simultanément. Il faut alors considérer des phénomènes évoluant à des échelles (temporelles et spatiales) et des niveaux de résolutions (de microscopique à macroscopique) différents. De plus, l'expertise nécessaire pour décrire le système vient en général de plusieurs domaines scientifiques, chaque domaine ayant ses propres modèles et outils de M&S (formalismes et logiciels).

Les défis sont alors de concilier ces points de vues hétérogènes, et d'intégrer l'existant de chaque domaine, tout en restant dans le cadre rigoureux de la démarche de M&S. On parlera alors de multi-modèle pour désigner le modèle d'un système complexe.

La question qui nous anime dans cette thèse est alors : Comment décrire sans ambiguïté, et simuler de manière rigoureuse un multi-modèle ?

Pour répondre à cette question nous nous inscrivons dans la continuité des travaux de M&S existants autour de l'intergiciel de co-simulation Agents & Artifacts for Multi-Modeling (AA4MM), du formalisme Discret-EVent System specification (DEVS), et de la démarche d'Ingénierie Dirigée par les Modèles (IDM). Nous étudions en effet dans cette thèse en quoi ces approches sont complémentaires et permettent, une fois combinées, de répondre aux défis de la démarche de M&S des systèmes complexes.

Notre contribution consiste à proposer un approche de M&S des systèmes complexes qui suit une démarche d'IDM, et est composée :

- d'un méta-modèle multi-agent basé sur AA4MM et définissant un langage de multi-modélisation non ambigu,
- de l'intergiciel de co-simulation MECSYCO associant aux concepts de AA4MM, des spécifications opérationnelles basées sur DEVS, et permettant de simuler un multi-modèle de manière rigoureuse,
- de règles de transformation permettant de passer automatiquement (et donc en s'affranchissant des erreurs potentielles de l'implémentation manuelle) d'une description de multi-modèle faite avec le méta-modèle, à sa simulation numérique avec l'intergiciel.

1.3 Organisation du manuscrit

Ce manuscrit est organisé comme suit.

Les Chapitres 2 et 3 introduisent les notions fondamentales sur lesquelles nous nous appuyons dans cette thèse. Le Chapitre 2 présente les différentes étapes et pré-requis de la démarche de M&S. Nous introduisons notamment les notions de modèle, simulation, paradigme et formalisme. Nous montrons également la diversité de la M&S en comparant la conception et simulation de modèles équationnels, événementiels et multi-agents. Nous montrons enfin la nécessité dans la démarche de la vérification systématique des implémentations des modèles.

Le Chapitre 3 détaille les problèmes qui apparaissent spécifiquement lorsque la démarche de M&S est appliquée aux systèmes complexes. Nous introduisons notamment les notions de multi-

modèle et de co-simulation. Nous détaillons également le caractère hétérogène des multi-modèles, les problèmes liés à son intégration, et les différents choix possibles face à ces problèmes.

Le Chapitre 4 détaille les approches de M&S existantes qui offrent des solutions intéressantes au problème d'intégration de l'hétérogénéité d'un multi-modèle. Nous évaluons les avantages et les limites de ces approches au regard des défis de la M&S. Nous montrons alors la complémentarité de AA4MM et DEVS pour répondre à ce problème.

Les Chapitres 5 à 8 dressent nos contributions.

Le Chapitre 5 détaille rapidement notre positionnement face aux problèmes de la modélisation et simulation, à savoir (1) comment AA4MM et DEVS sont combinés sous la forme de l'intergiciel MECSYCO afin d'intégrer l'hétérogénéité d'un multi-modèle, et (2) comment l'approche d'IDM est utilisée pour systématiser le passage d'un multi-modèle à sa simulation dans MECSYCO.

Le Chapitre 6 développe les spécifications opérationnelles de l'intergiciel de co-simulation MECSYCO.

Le Chapitre 7 détaille notre démarche d'ingénierie dirigée par les modèles, et l'implémentation que nous en avons réalisée.

Le Chapitre 8 présente plusieurs preuves de concept nous permettant d'évaluer les propriétés de notre approche, à savoir (1) la systématisation de l'implémentation d'un multi-modèle, (2) l'exécution parallèle et décentralisée de MECSYCO et (3) l'intégration de l'hétérogénéité d'un multi-modèle.

Pour finir, le Chapitre 9 détaille comment l'approche a été utilisée avec succès par les membres du projet Multi-Simulation for Smart Grid (MS4SG) issu d'un partenariat entre EDF R&D et Inria pour simuler des réseaux de distribution d'électricité intelligents (smart grid).

Chapitre 2

Démarche de Modélisation et simulation

Sommaire

2.1 Introduction

La démarche de modélisation et simulation (M&S) est une activité permettant d'étudier un système par l'intermédiaire d'objets appelés modèles. Comme elle ne se base pas directement sur le système mais sur une simplification de celui-ci (un modèle), un certain nombre de précautions doivent être prises pour s'assurer de la pertinence des réponses obtenues grâce au modèle. C'est le but des activités de validation et vérification.

Pour ce faire, la démarche de M&S comporte plusieurs étapes distinctes, dont la mise en œuvre n'est pas triviale, et qu'il convient de détailler ici pour comprendre les enjeux de cette thèse. Elle mobilise également plusieurs notions propres au domaine de la M&S qu'il est nécessaire de définir clairement pour expliquer notre travail.

Dans ce chapitre, nous présentons tout d'abord les activités de modélisation et simulation. Nous montrons ensuite comment ces deux activités sont conjuguées avec les activités de vérification et validation pour s'assurer un cadre scientifique rigoureux. Nous présentons ensuite comment toutes ces activités s'articulent au sein de la démarche de modélisation et simulation.

En détaillant les différentes étapes de cette démarche, nous introduisons les notions de paradigme et de formalisme. Nous déclinons ensuite plusieurs exemples représentatifs de paradigmes et de formalismes pour montrer la diversité des modèles qui peuvent être produits par la démarche de modélisation et simulation.

2.2 Qu'est ce qu'un modèle ?

Il n'y a pas à l'heure actuelle de consensus quant à la définition du terme modèle. Nous retiendrons ici deux définitions.

La Définition 1, donnée par Zeigler apporte un éclairage sur la nature d'un modèle : un modèle est une représentation abstraite d'un système cible (voir Définition 2).

Définition 1 *Un **modèle** est une simplification du système étudié qui a un rôle de médiateur entre le réel et la théorie [Zeigler et al., 2000].*

Définition 2 *Nous appelons **système cible** le système que l'on souhaite étudier.*

La Définition 3, donnée par Minsky apporte un éclairage sur la fonction d'un modèle. Cette définition est importante car elle soulève une caractéristique fondamentale d'un modèle : il a été créé dans le but précis de répondre à une question sur le système cible. En conséquence, un modèle capture uniquement les aspects du système cible nécessaires pour répondre à cette question de modélisation. C'est pourquoi, un modèle ne peut pas, en général, être utilisé pour répondre à n'importe quelle question sur le système cible qu'il représente, mais uniquement à la question pour laquelle il a été conçu.

Définition 3 *« Pour un observateur B, un objet A^* est un **modèle** d'un système cible A, si A^* permet de répondre à une question de B sur A » [Minsky, 1965].*

La question de modélisation à laquelle le modèle va répondre peut être tant de l'ordre de la compréhension du système, que de la prédiction de son évolution [Duboz et al., 2012]. Dans le premier cas, c'est le fait d'abstraire dans un modèle les mécanismes de base du système cible, qui va permettre de comprendre le fonctionnement de celui-ci. La création du modèle en tant que tel permet alors de répondre à la question. Dans le deuxième cas, on va se servir du modèle pour prédire comment le système cible va évoluer.

Klir et ensuite Zeigler [Zeigler et al., 2000] ont défini plusieurs niveaux de spécification qu'un modèle peut atteindre. Plus le niveau est élevé, plus le modèle donne une description détaillée du système cible :

- **Niveau 0 :** les conditions d'observation du système sont définies, c'est à dire quelles variables doivent être observées, et comment les mesurer dans la durée.
- **Niveau 1 :** les entrées et sorties du modèle indexées dans le temps sont définies.
- **Niveau 2 :** on introduit la notion d'état du modèle comme intermédiaire pour lier les entrées et les sorties : chaque stimulus va, étant donné l'état du modèle, produire une sortie unique.
- **Niveau 3 :** on définit la dynamique du modèle, c'est à dire qu'on définit les transitions entre les états du modèle, mais aussi comment chaque état du modèle est affecté par une entrée.
- **Niveau 4 :** on définit la structure du modèle : celui-ci est composé d'un ensemble de composants interconnectés.

Plusieurs techniques existent pour se servir d'un modèle afin de répondre à une question sur le système cible, notamment la résolution analytique (qui n'est pas toujours possible), ou la simulation. Dans cette thèse nous nous intéressons à cette dernière.

2.3 Qu'est ce qu'une simulation ?

De même que pour un modèle, il n'y a pas de définition consensuelle d'une simulation. Tuncer Őren a collecté plus de 100 définitions différentes de ce terme [Ören, 2011b], et environ 400 types différents de simulations [Ören, 2011a]. Parmi ces définitions, nous retiendrons la définition de Korn et Waits (Définition 4) qui apparaît comme la plus communément acceptée [Uhrmacher, 2012].

Définition 4 *Une **simulation** est une expérimentation sur un modèle ([Ören, 2011b] d'après Korn et Waits).*

Cette définition permet d'expliciter le lien qui existe entre un modèle et sa simulation. Une simulation va consister à exécuter un modèle, c'est à dire à le faire évoluer dans le temps en changeant pas à pas son état en fonction de sa dynamique, tout en produisant les sorties associées à chaque état. Une simulation va alors générer des informations sur l'évolution du système dans le temps. Nous appellerons ces informations des résultats de simulation. Le modélisateur pourra alors exploiter les résultats de simulation pour répondre à la question de modélisation.

Il faut bien noter ici qu'une simulation va faire évoluer le modèle dans un temps distinct du temps réel. En effet, une simulation de l'évolution d'un système sur une année entière peut prendre en réalité quelques secondes à s'exécuter. On distinguera alors le temps simulé faisant référence au temps dans lequel le modèle évolue, du temps réel faisant référence au temps que prend effectivement la simulation pour s'exécuter [Fujimoto, 2001].

Une simulation est en général effectuée par un ordinateur, ce qui va nécessiter la traduction du modèle sous la forme d'un programme exécutable et son implémentation. Notons ici que la simulation requiert que le modèle soit doté d'un état et d'une dynamique clairement définis. Le modèle doit donc atteindre au moins le niveau 3 de spécification [Zeigler et al., 2000].

2.4 Validation et Vérification

Comme nous l'avons dit précédemment, un modèle est une représentation d'un système cible qui va permettre de répondre à une question de modélisation. Cependant, il convient de s'assurer que la réponse fournie par le modèle est correcte. En effet, dans le cas contraire, le modèle peut engendrer une mauvaise compréhension du système cible ou des prédictions inexactes de l'évolution de celui-ci. La vérification (Définition 5) et la validation (Définition 6) d'un modèle vont permettre de s'assurer que celui-ci peut être utilisé pour fournir une réponse correcte. Si c'est le cas, alors le modèle est dit valide.

Définition 5 *La **vérification** d'un modèle consiste à prouver que celui-ci a été traduit d'une manière suffisamment précise sous la forme d'un programme exécutable [Balci, 1998].*

Définition 6 *La **validation** est le fait de prouver, étant données les conditions d'observation du système cible, qu'un modèle se comporte d'une manière suffisamment cohérente par rapport au système cible [Balci, 1998].*

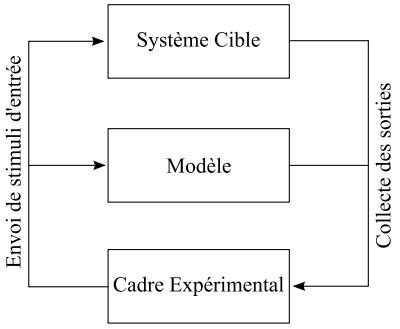


FIGURE 2.1 – Interaction du cadre expérimental avec un modèle et son système cible (Inspirée de [Zeigler et al., 2000]).

La vérification d'un modèle peut être assurée de deux manières différentes :

— **A posteriori**, en produisant manuellement le code de simulation avant de vérifier que celui-ci retranscrit fidèlement le modèle. Plusieurs techniques sont alors applicables en allant de la simple relecture du code par un tiers, à la preuve formelle du programme, en passant par son analyse syntaxique [Whitner and Balci, 1989].

— **A priori**, en définissant un processus automatique capable de transcrire correctement et de manière systématique les modèles sous forme exécutable. C'est alors le domaine de l'Ingénierie Dirigée par les Modèles (IDM) qui consiste à (1) formaliser le langage utilisé pour décrire le modèle afin de rendre ce dernier analysable par un ordinateur, et (2) définir ensuite un générateur de code capable de transcrire correctement la description du modèle sous forme exécutable [Gray et al., 2007].

Dans le cadre d'une simulation, la validation du modèle peut être effectuée en comparant des résultats de simulation générés par une exécution du modèle, avec les données collectées depuis le système cible [Balci, 1998].

La validation d'un modèle va être basée sur son cadre expérimental [Zeigler et al., 2000]. Celui-ci va décrire les conditions d'observation du système cible et permettre la génération des stimuli d'entrée et la collecte des données résultantes, pour le système cible et pour son modèle (Figure 2.1) [Traoré and Muzy, 2006]. Ces résultats pourront alors être comparés pour valider le modèle.

La création d'un modèle valide est une tâche non triviale, et il est en général rare qu'une première occurence d'un modèle soit valide. La conception d'un modèle progresse alors par une suite d'essais et d'erreurs qui s'articulent au sein de la démarche de modélisation et simulation. Nous présentons dans la suite cette démarche.

2.5 Démarche de modélisation et simulation

La démarche de modélisation et simulation [Quesnel, 2006] a pour but la production d'un modèle valide (Figure 2.2). Cette démarche est composée de trois processus successifs. Le processus de modélisation a pour but la production d'un modèle implémenté et simulable. Le processus de simulation est effectué par un ou plusieurs ordinateurs et a pour but d'exécuter le modèle et de produire des résultats de simulation. Enfin, le processus d'analyse consiste à vérifier la validité du modèle à partir des résultats de simulation.

Dans le cas où le modèle n'est pas valide, la démarche de modélisation va itérer : le processus de modélisation va rechercher des erreurs dans les choix de modélisation et les spécifications

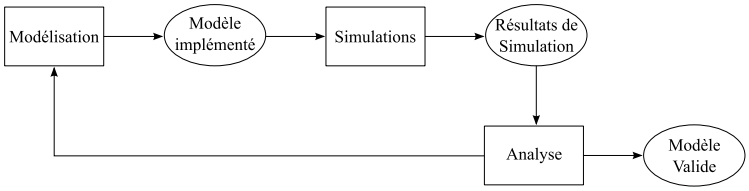


FIGURE 2.2 – La démarche de modélisation et simulation (Inspirée de [Quesnel, 2006])

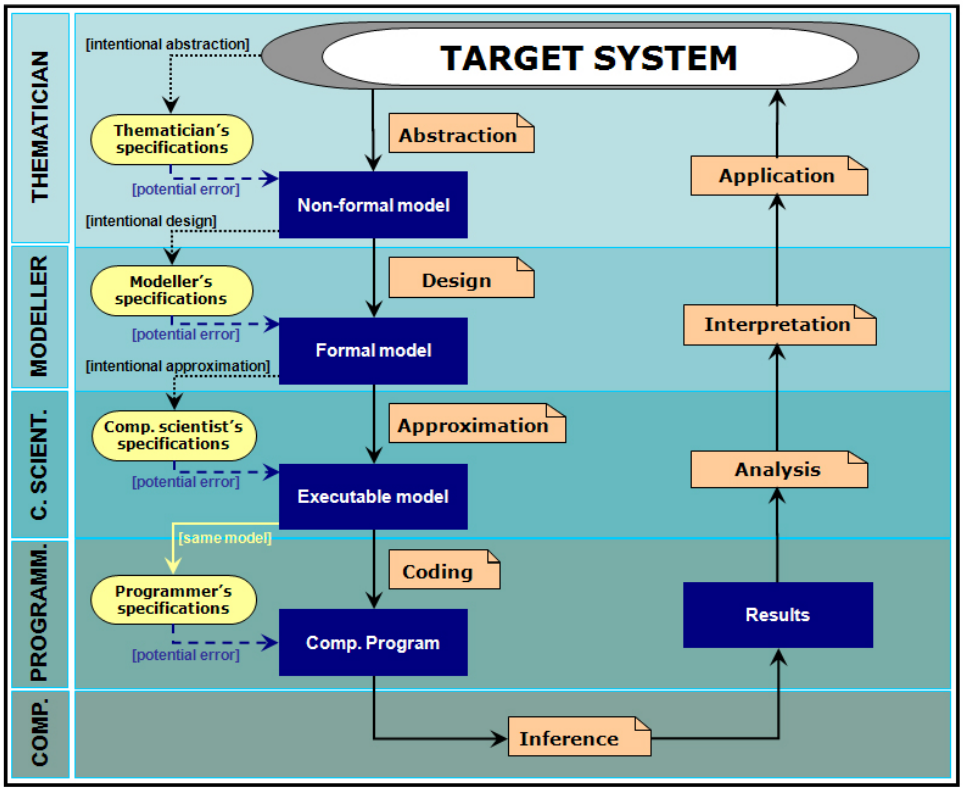


FIGURE 2.3 – Les différentes étapes du processus de modélisation. Source [Galán et al., 2009]

du modèle, et va modifier celui-ci en conséquence. Le modèle produit par le processus va alors s'affiner à chaque itération de la démarche et celle-ci va alors converger vers un modèle valide du système. Le processus de modélisation est alors le pivot de la démarche de modélisation et simulation.

Le processus de modélisation peut être divisé en quatre étapes (Figure 2.3) [Galán et al., 2009] : conceptualisation, formalisation, opérationalisation et numérisation. A chacune de ces étapes, le modèle va être re-spécifié de manière à le rapprocher d'une version exploitable par un ordinateur. Chaque spécification du modèle va être faite par un acteur différent de la modélisation et simulation et peut potentiellement contenir des erreurs qui vont entacher la validité et la vérification du modèle.

Les étapes du processus de modélisation et simulation sont les suivantes :

— La phase de conceptualisation est gérée par un expert du système cible : le thématicien. Celui-ci va définir une représentation du système : le modèle conceptuel. A ce niveau, des erreurs de modélisation nuisant à la validité du modèle peuvent apparaître si le thématicien définit une représentation qui n'abstrait pas de manière suffisamment précise le système

cible.
- La phase de formalisation est gérée par un spécialiste de la modélisation : le modélisateur. Celui-ci va formaliser la représentation du thématicien, et spécifier alors le modèle formel. De la même manière que le thématicien, le modélisateur peut également faire des erreurs de modélisation qui vont nuire à la validité du modèle.
- La phase d'opérationalisation est gérée par un informaticien. Celui-ci va rendre fonctionnel le modèle formel du modélisateur en le traduisant sous la forme d'un programme exécutable. L'informaticien définit alors le modèle exécutable. Si des erreurs sont commises pendant l'opérationnalisation du modèle, le programme exécutable obtenu pourra être une traduction non suffisamment précise du modèle formel et le modèle ne pourra alors pas être vérifié.
- La phase de numérisation est gérée par le programmeur. Celui-ci va créer le modèle numérique : il va implémenter le modèle exécutable de l'informaticien pour permettre la simulation du modèle. Là encore, la vérification du modèle peut être entachée par des erreurs d'implémentation qui font que le modèle numérique ne reflètera qu'imparfaitement le modèle.

Dans la suite nous détaillons plus précisément en quoi consistent ces quatre phases.

2.5.1 Le modèle conceptuel

Le thématicien va simplifier le système pour pouvoir le représenter. En effet, il va se positionner à un certain niveau de détail, considérer comment le système évolue uniquement dans les conditions spécifiées par le cadre expérimental [Duboz et al., 2012], et faire un ensemble d'hypothèses de manière explicite ou implicite sur son fonctionnement [Hofmann, 2005]. Ces simplifications vont constituer des choix de modélisation qui vont être faits en fonction des connaissances du thématicien, des informations disponibles sur le système, et surtout de la question de modélisation.

Lorsqu'il va modéliser le système, le thématicien va également (implicitement ou explicitement) choisir un paradigme (voir Définition 7). Ce paradigme va conditionner la manière dont le système est représenté. Le choix du paradigme peut être dicté tant par les caractéristiques du système que par le domaine scientifique duquel est issu le thématicien ou les connaissances de celui-ci.

Définition 7 *« Un **paradigme** est un cadre de pensée composé par un ensemble d'hypothèses fondamentales, de lois et de moyens sur la base desquels les modèles peuvent se développer » [Quesnel, 2006].*

Le thématicien définit en général le modèle conceptuel en utilisant le langage naturel. Le modèle conceptuel peut alors contenir des ambiguïtés qui devront être levées lors de la phase de formalisation [Galán et al., 2009]. Ces ambiguïtés peuvent également être limitées par l'utilisation d'une ontologie à la place du langage naturel pour définir le modèle conceptuel [Polhill and Gotts, 2006].

2.5.2 Le modèle formel

Une fois que le thématicien a défini le modèle conceptuel, le modélisateur va devoir définir le modèle formel pour :
- Décrire le modèle de façon à ce que celui-ci puisse ensuite être implémenté et simulé par un ordinateur.

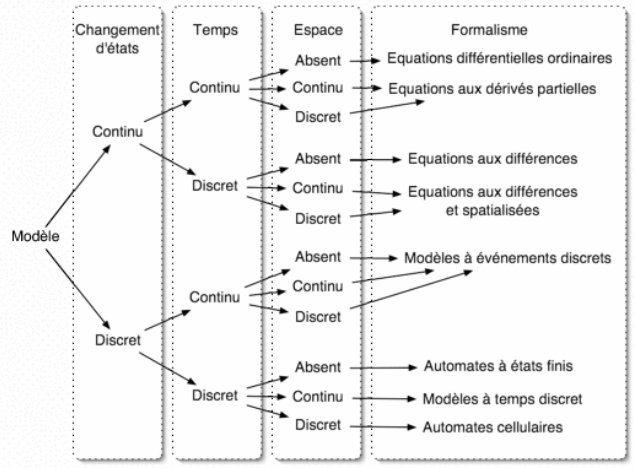


FIGURE 2.4 – Classification de certains formalismes. Source [Ramat, 2006]

— Décrire le modèle de manière non ambiguë et donc permettre de réutiliser et reproduire le modèle.

Selon [Quesnel, 2006] un formalisme est l'outil d'expression d'un paradigme. Il existe en général plusieurs formalismes différents associés à un même paradigme. Par exemple, plusieurs formalisations des systèmes multi-agents existent [Duboz et al., 2012]. La définition de formalisme que nous prendrons est donnée par la Définition 8. Selon cette définition, un modèle formel est une abstraction mathématique décrivant un système cible et pouvant être implémentée de manière univoque dans un ordinateur.

Définition 8 *Un **formalisme** est un langage formel permettant de spécifier un modèle de manière non ambiguë.*

Les différents formalismes existants ne vont pas représenter le système de la même manière : le temps, les changements d'état et l'espace, sont représentés de manière discrète ou continue (voir Figure 2.4) [Ramat, 2006]. Il convient d'expliciter la différence entre un temps et un changement d'état discret. Lorsque les changements d'état sont discrétisés, on considère que le système change d'état à un ensemble de points situés n'importe où dans le temps. Lorsque le temps est discrétisé, on met à jour l'état du système uniquement à une suite d'instants successifs dans le temps. Rien ne peut alors se produire entre deux instants : le modèle ne change pas d'état, ne produit pas de sortie et n'intègre pas d'entrée.

2.5.3 Le modèle exécutable

Une fois le modèle formalisé, l'informaticien va définir comment celui-ci sera implémenté et exécuté. Il va alors définir des spécifications opérationnelles du modèle. Ces spécifications opérationnelles incluent notamment la définition du simulateur abstrait (voir Définition 9). Ce dernier va avoir pour rôle de gérer l'exécution du modèle.

Chaque simulateur abstrait est spécifique à un formalisme. A l'inverse, chaque formalisme peut avoir plusieurs simulateurs abstraits différents [Uhrmacher, 2012]. En général, un formalisme dispose d'un ensemble de simulateurs abstraits prédéfinis que l'informaticien peut directement utiliser.

Différents simulateurs abstraits vont simuler un modèle avec des politiques d'exécution différentes, c'est à dire qu'ils vont faire évoluer le modèle dans le temps de différentes manières. Nous

distinguons ici trois politiques d'exécution : à pas de temps cyclique, à pas de temps acyclique et événementielle. Nous détaillons ces différentes exécutions à travers les exemples de plusieurs formalismes en Section 2.6

Définition 9 *Le **simulateur abstrait** d'un modèle correspond à l'algorithme de simulation permettant d'exécuter le modèle.*

2.5.4 Le modèle numérique

La tâche d'implémenter le modèle exécutable dans un ordinateur est effectuée par le programmeur. Celui-ci va avoir deux choix [Uhrmacher, 2012] :

— Ecrire lui même le modèle numérique à partir de rien. Dans ce cas, l'informaticien va devoir choisir un langage de programmation pour implémenter le modèle. Le choix du langage peut être basé tant sur les caractéristiques des modèles formel et exécutable (e.g. un langage offrant une bibliothèque mathématique sera préféré pour implémenter et simuler un modèle équationnel) que sur les aptitudes du programmeur.

— Utiliser un outil de simulation pré-existant adapté aux caractéristiques du modèle exécutable. Ces outils de simulation, qui sont en général spécifiques à un formalisme, fournissent des simulateurs abstraits déjà implémentés ainsi que des primitives logicielles facilitant l'implémentation du modèle.

En général, l'utilisation d'un outil de simulation est préférable car celui-ci permet de diminuer considérablement le temps de développement du modèle numérique, et de limiter les erreurs d'implémentation [Uhrmacher, 2012].

2.6 Différents paradigmes, leurs formalismes et leurs simulations

Nous avons vu que la démarche de modélisation et simulation pouvait amener, grâce à la diversité de ses paradigmes, à la production de modèles qui diffèrent selon leurs formalismes, leurs politiques d'exécution et leurs implémentations. Dans la suite nous décrivons certains de ces paradigmes, nous détaillons comment ils sont formalisés et exécutés par leurs simulateurs abstraits.

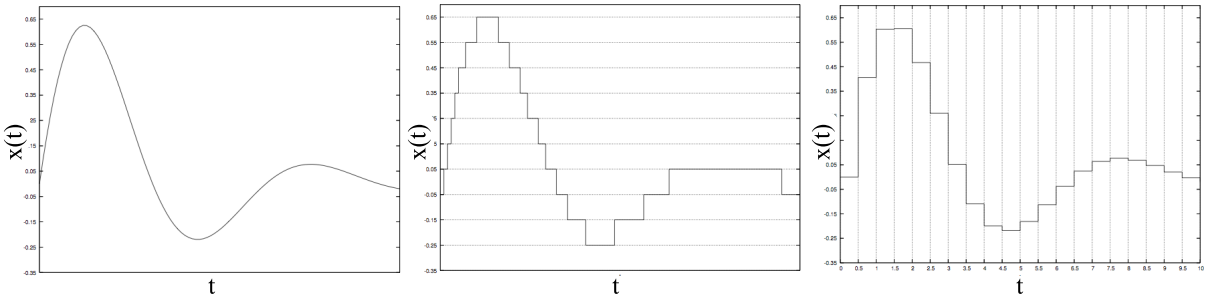
2.6.1 Le paradigme équationnel

Le paradigme équationnel est, de par son ancienneté, parmi les plus utilisés pour décrire mathématiquement l'évolution des systèmes dynamiques. Deux de ces formalisations sont communément utilisées en simulation numérique : les équations aux différences et les équations différentielles. Les simulateurs abstraits de ces formalismes, qui ont pour rôle de résoudre numériquement ces équations, sont appelés des solveurs d'équation.

Les équations aux différences

Une équation aux différences décrit les différents états d'un système à une suite d'instants successifs dans le temps : le temps est discrétisé de manière régulière [Sedaghat, 2007]. L'état du système à un instant donné est fonction des m états précédents du système. On dira alors que l'équation est d'ordre m. L'équation correspond à une suite mathématique avec une relation de récurrence. Une équation aux différences d'ordre m est alors de la forme :

$$x_n = f(x_{n-1}, x_{n-2}, ..., x_{n-m}) \tag{2.1}$$



(a) (b) (c)

FIGURE 2.5 – Les différentes discrétisations possibles d'équation différentielle. (a) L'évolution du système tel que décrit par l'équation différentielle (b) Discrétisation des changements d'état (c) Discrétisation du temps (Source [Nutaro, 2007]).

Il est en général facile d'écrire un solveur d'équations aux différences [Sedaghat, 2007]. En effet, pour simuler l'évolution d'un système s jusqu'à un instant t étant donné les m états initiaux $x_0, x_{0-1}, ..., x_{0-m+1}$ de s, il suffit de calculer successivement les états $x_1, x_2, ..., x_n$ en utilisant l'équation aux différences qui décrit sa dynamique. Le solveur d'équations aux différences simule alors l'évolution du système avec une politique d'exécution à pas de temps cyclique : le modèle évolue de manière régulière dans le temps.

Les équations différentielles

Une équation différentielle décrit la variation de l'état d'un système [Esquembre and Christian, 2007]. Ces équations sont plus compliquées à résoudre numériquement à cause de leurs représentations continues du temps et des changements d'état du système (Figure 2.5a). L'informaticien doit alors choisir de discrétiser une de ces deux représentations [Nutaro, 2007].

Discrétiser les changements d'état (Figure 2.5b) revient à définir un modèle événementiel qui peut être ensuite exécuté par un simulateur abstrait événementiel (voir Section 2.6.2). Pour discrétiser le temps (Figure 2.5c), plusieurs méthodes sont possibles, les plus répandues étant les méthodes d'Euler et de Runge-Kutta [Esquembre and Christian, 2007]. Ces méthodes permettent d'approximer une équation différentielle par une équation aux différences évoluant selon un pas de temps h. Le solveur d'équations différentielles peut alors simuler l'évolution d'un système (en partant de son état initial) par pas de temps cyclique comme avec une équation aux différences.

Pour fixer la valeur de h, l'informaticien doit trouver un compromis entre la performance de la simulation et l'exactitude des résultats [Esquembre and Christian, 2007]. En effet, plus h sera grand, moins la simulation nécessitera de pas de temps pour atteindre une date T ($h < T$). La simulation nécessitera donc moins de calculs. En revanche, il ne faut pas perdre de vue que la discrétisation temporelle d'une équation différentielle ne donne pas la solution exacte de l'équation mais uniquement une approximation de la solution. En conséquence, plus h sera grand, moins le solveur sera capable de prendre en compte des variations rapides de la solution, et donc plus le solveur donnera des résultats de simulation erronés (Figure 2.6).

En fait, la valeur optimale de h est fonction du profil de la solution : si celle-ci varie lentement, h doit être grand et si la solution varie rapidement, h doit être petit. Pour les solutions qui varient lentement sur certains intervalles et rapidement sur d'autres, il existe des méthodes d'ajustement automatique du pas de temps basées sur la méthode de Runge-Kutta [Esquembre and Christian, 2007]. Ces méthodes permettent de tirer le meilleur compromis entre performance de la simulation

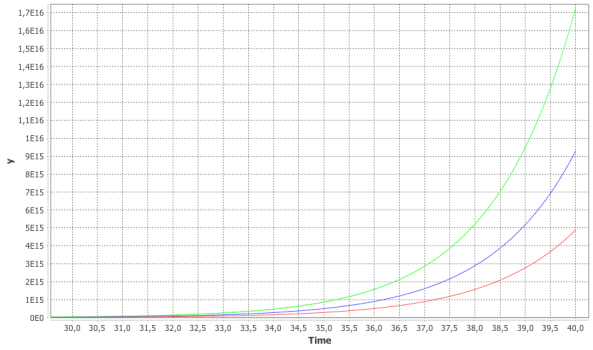


FIGURE 2.6 – Résolutions numériques de l'équation différentielle $\frac{dx}{dt} = 0.6x$ avec la méthode d'Euler pour $h = 0.2$ (courbe rouge), $h = 0.1$ (courbe bleue) et $h = 0.01$ (courbe verte).

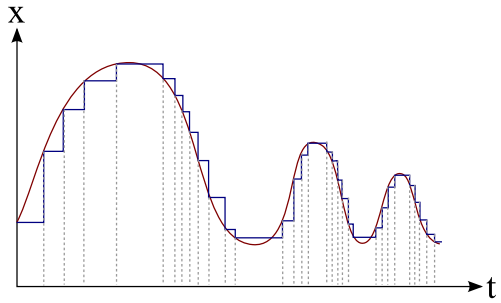


FIGURE 2.7 – Exemple de simulation à pas de temps variable d'une équation différentielle. La courbe rouge correspond à la solution de l'équation. La ligne brisée bleue correspond aux résultats de simulation à pas de temps variable.

et exactitude des résultats. Un solveur d'équations différentielles basé sur ces méthodes va donc s'exécuter par pas de temps de taille acyclique (i.e. de taille variable) (voir Figure 2.7).

2.6.2 Le paradigme événementiel

Les formalismes s'exécutant à pas de temps cycliques décrivent l'évolution du système de manière régulière : à chaque pas de temps le système change d'état. Cependant, lorsque l'on souhaite représenter un système, on peut vouloir observer les changements d'état du système uniquement à une suite d'instants qui ne sont pas forcément répartis de manière régulière dans le temps. Ces changements d'état peuvent être causés tant par la dynamique interne du système, que par des stimuli extérieurs pouvant arriver à n'importe quel moment dans le temps.

Dans cette optique, le paradigme événementiel permet de prendre le contrepied d'une vision à pas de temps : il représente l'évolution d'un système à un ensemble de points discrets qui peuvent être localisés n'importe où dans un temps continu. Ces points sont appelés des événements. Chaque événement va alors être associé à un changement de l'état du modèle [Kim, 2007].

La différence principale entre un modèle événementiel et un modèle à pas de temps acyclique repose sur la représentation continue du temps. Cette représentation continue rend possible l'arrivée d'entrées dans le système à n'importe quel moment (voir Figure 2.8).

Lors de l'exécution d'un modèle événementiel, le simulateur abstrait de ce dernier va faire des sauts dans le temps en allant directement au temps du prochain événement. Cet événement va ensuite être exécuté, c'est à dire que le simulateur va effectuer le changement de l'état associé

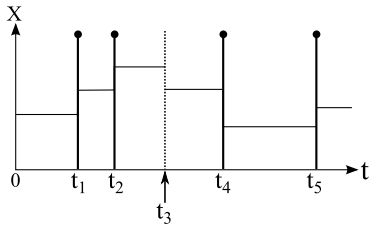


FIGURE 2.8 – Exemple de trajectoire d'état d'un modèle événementiel. La modèle change d'état à cause de sa dynamique interne aux temps t_1, t_2, t_4 et t_5, et reçoit une entrée (un stimulus perturbateur) au temps t_3.

à cet événement. Le simulateur suit alors une exécution événementielle.

Le modélisateur peut adopter trois stratégies pour décrire comment évolue un modèle événementiel (i.e. pour décrire comment le prochain événement est déterminé et planifié) [Hooper, 1986] :

- Dans une stratégie de **planification d'événements**, le modélisateur va décrire comment les événements se planifient dans le temps. Les événements planifiés sont en général stockés dans une pile. Le prochain événement est alors l'événement le plus imminent de la pile. L'exécution d'un événement peut modifier la pile d'événements en planifiant de nouveaux événements.
- Dans une stratégie de **scan d'activité**, le modélisateur va décrire comment le modèle évolue en fonction de son état. Le prochain événement est alors déterminé et planifié à partir de l'état du modèle.
- Dans une stratégie d'**interaction de processus**, le modélisateur va décrire le système en combinant les deux stratégies précédentes : le modélisateur va définir des séquences d'événements (processus) qui vont se déclencher en fonction de l'état du modèle. Lorsqu'un processus est déclenché, ses événements vont être planifiés dans la pile d'événements. Le prochain événement va alors être déterminé à partir de l'état du modèle et de l'état de la pile d'événements.

2.6.3 Le paradigme multi-agent

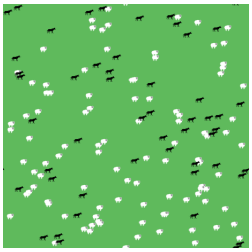
Le paradigme équationnel va en général considérer le système dans sa globalité en le représentant de manière agrégée. Les approches individu-centrées prennent le contrepied de cette vision et proposent, en considérant un certain niveau de résolution sur le système, de représenter individuellement les entités qui le composent, leurs interactions et leurs environnements.

Prenons l'exemple de la modélisation des évolutions conjointes de populations de proies et de prédateurs. Une approche équationnelle représente le système avec les équations de Lotka-Volterra (Figure 2.9a) : on représente l'évolution du nombre de proies (noté x) en fonction du nombre de prédateurs (noté y) et vice versa. A l'opposé, une approche individu-centrée considèrera chaque proie et chaque prédateur individuellement ainsi que leur environnement (Figure 2.9b).

Un modélisateur adoptant une approche individu-centrée va en général emprunter le paradigme multi-agent pour concevoir son modèle. Ce paradigme, permet de représenter chaque entité d'un système comme un agent autonome ayant un comportement propre et évoluant dans un environnement [Ferber, 1995]. On parlera alors de système multi-agent. En dehors de la modélisation et la simulation, le paradigme multi-agent a également des applications dans d'autres domaines comme l'ingenierie logicielle, l'intelligence artificielle et la robotique.

$$\begin{cases} \frac{dx}{dt} = \alpha x - \beta xy \\ \frac{dy}{dt} = \delta xy - \gamma y \end{cases}$$

(a)



(b)

FIGURE 2.9 – Différentes représentations d'un système proies/prédateurs. (a) Une approche équationnelle (b) Une approche individu-centrée (Source [Wilensky, 1997b]).

Un système multi-agent est défini en général selon quatres critères [Demazeau, 1995] :

Les agents sont les entités autonomes dirigées par un but. Un agent dispose de senseurs lui permettant de percevoir de manière incomplète son environnement, ainsi que d'effecteurs lui permettant de modifier son environnement et d'influencer la dynamique des autres agents. Un agent est une entité proactive mue par son comportement. Ce dernier est en général défini selon le pattern de la boucle perception/décision/action [Ferber, 1995] : l'agent acquiert des perceptions de son environnement via ses senseurs, décide de l'action à entreprendre en fonction de ses perceptions, et enfin agit via ses effecteurs. Selon la complexité du système de décision des agents, on distingue en général les agents cognitifs et les agents réactifs. Là où les agents réactifs associent simplement une perception à une action, les systèmes de décision des agents cognitifs sont plus complexes. Ainsi, un agent cognitif peut être doté de facultés d'apprentissage, de raisonnement, voire de planification d'actions sur le long terme. Les agents cognitifs peuvent alors être capables de s'adapter aux changements de leur environnement.

L'environnement est l'espace dans lequel les agents évoluent. Les agents vont avoir une localisation dans cet espace. L'environnement peut également contenir des ressources que les agents vont pouvoir utiliser pour atteindre leurs buts. Cet environnement peut être doté d'un état et d'une dynamique décrivant les lois d'évolution du système (e.g. représentation de la gravité).

Les interactions correspondent aux actions et influences que les agents exercent entre eux ou sur leur environnement. Les interactions entre agents peuvent être directes (e.g. un agent envoie directement un message à un autre agent) ou indirectes [Thomas, 2005], c'est à dire passer via l'environnement. Dans ce dernier cas, l'environnement est alors le médium d'interaction des agents. Les interactions entre agents vont alors modifier l'environnement, mais aussi être contraintes et potentiellement perturbées par les lois de ce dernier.

L'organisation décrit la structure des interactions entre les agents et comment ceux-ci se coordonnent pour atteindre leur but.

Nous pouvons voir ici que le paradigme multi-agent laisse beaucoup de degrés de liberté au modélisateur. Ce dernier doit en effet faire un nombre important de choix pour définir un système multi-agent. Le paradigme multi-agent est en fait composé de plusieurs sous-paradigmes qui offrent des réponses pré-établies à certains de ces choix. Ces sous-paradigmes constituent alors des outils expressifs dont le modélisateur peut se servir pour faciliter la création du système multi-agent. Dans la suite, nous présentons certains de ces sous-paradigmes.

Différents sous-paradigmes multi-agents

Le paradigme Agent Groupe Rôle (AGR) permet au modélisateur de décrire l'organisation d'un système multi-agent grâce aux concepts de Groupe et de Rôle [Ferber and Gutknecht, 1998]. Dans AGR, un agent appartient à un ou plusieurs groupes, et les agents peuvent dynamiquement rejoindre et quitter des groupes. A l'intérieur de chaque groupe, chaque agent joue un ou plusieurs rôles. La structure d'un groupe définit les rôles qui composent un groupe et quelles sont les interactions entre les agents qui jouent ces rôles.

Le paradigme Influence-Réaction précise les rapports entre les agents et leur environnement. Dans un système multi-agent, les agents peuvent agir simultanément dans l'environnement. Il se peut alors que les actions de plusieurs agents entrent en conflit. Ce peut être le cas par exemple si deux agents veulent simultanément prendre une même ressource dans l'environnement. Si ce cas n'a pas été traité par le modélisateur, le modèle comporte alors une ambiguïté qui peut amener à un état incohérent du système (e.g. les deux agents obtiennent la ressource), ou nuire à la reproductibilité des résultats de simulation (e.g. l'agent qui obtient la ressource varie selon les simulations).

Le paradigme Influence-Réaction [Michel, 2007] permet au modélisateur de lever cette ambiguïté. Dans ce paradigme, les agents n'agissent pas directement sur l'environnement mais émettent des influences. Ces influences représentent les actions que les agents ont prévus d'effectuer dans l'environnement (« je veux prendre cette ressource »). En réponse à ces influences, l'environnement va émettre des réactions : il va déterminer quel est l'impact des influences des agents (e.g. aucun agent n'obtient la ressource).

Le paradigme Agents & Artifacts (A&A) [Ricci et al., 2007b] considére que l'environnement peut ne pas simplement être le médium d'interaction des agents, mais qu'il peut également être un espace de travail facilitant les actions des agents et les aidant à atteindre leurs buts. Dans cet optique, A&A étend le paradigme multi-agent par l'ajout du concept d'artefact.

Dans A&A, l'environnement est composé d'un ensemble d'artefacts représentant les outils dont les agents peuvent se servir pour agir et percevoir. Un artefact est défini comme une entité passive de l'environnement encapsulant des mécanismes que les agents peuvent activer [Omicini et al., 2008]. Les artefacts proposent alors des services aux agents. Ces services peuvent permettre par exemple :

- De coordonner les activités des agents les unes avec les autres. Ce type de service peut être utilisé si les agents sont impliqués dans une tâche collective. On parlera alors d'artefact de coordination [Omicini et al., 2004].
- De modifier l'environnement.
- De médier les interactions entre les agents.
- D'effectuer des calculs.

A&A offre plusieurs intérêts pour le modélisateur :

- Les artefacts permettent de modéliser l'exécution des actions des agents comme des processus (i.e. une suite d'événements dans le temps). L'action d'un agent n'est plus alors forcément immédiate, mais peut prendre du temps pour se réaliser et avoir des effets à long terme [Ricci et al., 2011].
- L'environnement est décrit de manière modulaire. Les artefacts peuvent être réutilisés pour définir différents modèles. Le modélisateur peut alors disposer d'un ensemble d'artefacts pré-existants pour définir l'environnement de ses agents.

— L'environnement est décrit de manière décentralisée. Le système multi-agent peut alors être facilement distribué [Ricci et al., 2007a].

Simulation des systèmes multi-agents

Il n'existe pas de formalisation univoque des systèmes multi-agents [Duboz et al., 2012]. Il y a alors plusieurs façons d'implémenter et de simuler un système multi-agent.

Un système multi-agent peut être simulé de manière synchrone ou asynchrone [Kubera et al., 2007]. Les simulations synchrones, qui sont les plus répandues, vont considérer que les agents perçoivent, décident et agissent tous en même temps. Elles consistent alors à adopter une politique d'exécution à pas de temps cyclique. Ainsi, à chaque pas de temps, les agents sont sélectionnés séquentiellement les uns après les autres, et leur boucle perception/décision/action est exécutée. Afin d'éviter que l'ordre d'exécution des agents ait une incidence sur l'évolution du système multi-agent, chaque agent ne va pas se baser directement sur l'environnement mais sur une copie de celui-ci. Une simulation asynchrone du système multi-agent va correspondre à une exécution événementielle où l'on va considérer que chaque agent pourra s'exécuter en parallèle des autres.

2.7 Conclusion

Dans ce chapitre, nous avons présenté la démarche de M&S qui permet l'étude d'un système cible à partir d'un modèle de celui-ci. Cette démarche permet d'inscrire les activité de modélisation et simulation dans un cadre scientifique rigoureux. L'objectif de la démarche est en effet de construire un modèle qui soit valide, afin de s'assurer que les réponses fournies par celui-ci soient exploitables.

La démarche de M&S suit pour ce faire un processus itératif constitué d'une suite d'étapes clairement définies : la création des modèles conceptuel, formel, exécutable et numérique. Ces étapes qui permettent d'aller de la conceptualisation du système à sa simulation numérique, vont se répéter jusqu'à ce que la démarche ait convergé vers une représentation valide du système.

Si la démarche de modélisation permet de s'assurer de la validité du modèle obtenu, elle présuppose cependant la vérification de celui-ci. Il convient en effet de s'assurer à chaque itération, que les résultats de simulation sont uniquement fonction des choix de modélisation et donc que l'implémentation du modèle n'introduit pas de biais. **Une démarche de modélisation et simulation doit alors être accompagnée soit d'une méthode permettant la vérification systématique du modèle numérique, soit d'un mécanisme permettant la traduction automatique d'un modèle formel sous sa forme numérique. C'est là un défi auquel nous devons répondre dans cette thèse.**

D'autre part, nous avons illustré à travers les exemples représentatifs des paradigmes équationnel, événementiel et multi-agent, la diversité des modèles qui peuvent être produits par la démarche de modélisation et simulation : ils peuvent être issus de paradigmes différents, écrits dans des formalismes différents et avoir des politiques d'exécution différentes (pas de temps cyclique, pas de temps acyclique, et événementielle). Cette diversité traduit la richesse de la M&S, et le grand nombre de choix à disposition du modélisateur. C'est pour nous une force de la M&S car elle peut adapter ses outils aux caractéristiques du système cible, à la question de modélisation et/ou à l'expérience du modélisateur. C'est pourquoi, **nous nous efforçons dans la suite de cette thèse d'englober toute la diversité de la M&S**.

Dans le chapitre suivant, nous identifions les problèmes supplémentaires qui vont se poser lorsque la démarche de modélisation et simulation est confrontée non pas à un système dynamique « classique » mais à un système complexe.

Chapitre 3

Modélisation et simulation de systèmes complexes

Sommaire

3.1 Introduction

Lorsque la démarche de M&S est appliquée à l'étude des systèmes complexes, ses modèles vont se complexifier, et un ensemble de problèmes spécifiques vont alors être rencontrés. Le modèle d'un système complexe va en effet correspondre à un multi-modèle décrivant le système cible comme un système de systèmes : un ensemble de systèmes hétérogènes en interactions. C'est alors l'hétérogénéité qui est le cœur du problème, car elle va être rencontrée à chaque étape de la démarche de M&S : elle concerne à la fois les représentations du système, les formalismes utilisés, et les logiciels implémentant le multi-modèle. La M&S des systèmes complexes nécessite alors la mise en place d'un ensemble de solutions spécifiques permettant d'intégrer cette hétérogénéité dans un tout cohérent.

L'éventail des solutions possibles étant large, un ensemble de choix doit être fait dans la démarche de M&S. Ces choix ne sont pas triviaux car ils vont impacter les propriétés du multi-modèle en le rendant plus ou moins malléable à l'expérimentation. Il est alors important, de

détailler les problèmes et les différentes solutions envisageables afin d'expliquer les choix que nous faisons dans cette thèse.

Dans ce chapitre, nous présentons tout d'abord les caractéristiques propres aux modèles des systèmes complexes et à leurs simulations. Nous présentons ensuite les problèmes rencontrés lors des différentes étapes de la démarche de M&S. Pour chaque problème, nous détaillons les différents choix possibles, leurs avantages et leurs limites au regard des pré-requis de la démarche de M&S.

3.2 Le modèle d'un système complexe : le multi-modèle

Lorsqu'il souhaite représenter et simuler un système complexe, le modélisateur va être confronté à la complexité descriptive du système (voir Chapitre 1). A cause de cette complexité descriptive, le modèle peut rapidement devenir aussi complexe que le système lui même [Yilmaz and Tolk, 2008], et plusieurs thématiciens issus de différents domaines scientifiques peuvent être nécessaires pour définir le modèle [Belem and Müller, 2009].

La complexité descriptive du système peut être abaissée en décrivant le système comme un système de systèmes : un ensemble de systèmes en interaction. Chacun de ces sous-systèmes sera représenté par un modèle et le modèle global sera alors appelé multi-modèle (voir Définition 10).

Définition 10 *Un **multi-modèle** est un modèle composé d'un ensemble de modèles en interaction qui, ensemble, décrivent l'évolution d'un système [Yilmaz and Ören, 2004, Siebert, 2011].*

3.2.1 Intérêt de la multi-modélisation pour représenter un système complexe

La représentation d'un système comme un multi-modèle permet de :

— **Séparer l'expertise des différents thématiciens**. La nature modulaire du multi-modèle permet de concentrer les domaines d'expertise de chaque thématicien dans un ou plusieurs modèles distincts. Le modélisateur peut alors se concentrer sur la validation et la vérification de l'expertise de chaque thématicien prise séparément dans un premier temps, et dans un deuxième temps sur la construction et la validation du multi-modèle [Müller and Diallo, 2012].

— **Réutiliser des modèles préexistants implémentés et validés**. La construction du multi-modèle d'un système complexe peut nécessiter un nombre important de modèles [Taylor et al., 2013]. La construction du multi-modèle est alors un processus extrêmement long et coûteux. Cependant, certains des modèles requis peuvent déjà exister, être validés et être déjà implémentés dans des outils de simulation. Dans cette optique, ces modèles constituent des expertises qu'il est important de pouvoir intégrer dans le multi-modèle [Breunese et al., 1998].

— **Construire un système complexe avec une approche orientée composants.** Avec un multi-modèle, la représentation du système est modulaire, et peut donc être facilement construite et modifiée [Argent, 2004] : il est possible d'enlever, ajouter ou remplacer un ou plusieurs modèles et de changer les connexions entre ceux-ci.

3.2.2 Caractéristiques d'un multi-modèle de système complexe

Le terme multi-modèle a originellement été proposé par Tuncer Őren [Ören, 1987] comme étant un ensemble de modèles décrivant l'évolution d'un système de manière successive : à un

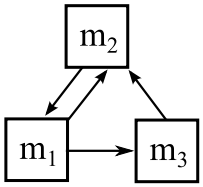


FIGURE 3.1 – Un exemple de structure d'un multi-modèle composé de trois modèles m_1, m_2 et m_3. Les flèches indiquent les interactions entre les modèles.

instant donné, un seul modèle est utilisé pour représenter le système. Grâce à ce type de multi-modèle, il est possible de séparer l'évolution d'un système en plusieurs phases, chacune de ces phases étant représentée par un modèle. Ces multi-modèles sont appelés **multi-modèles séquentiels** et peuvent être opposés à un autre groupe de multi-modèles apparus historiquement plus tard : les **multi-modèles multi-aspects** [Yilmaz et al., 2007]. Dans un multi-modèle multi-aspect, plusieurs modèles sont actifs simultanément pour représenter le système. Les modèles vont alors co-évoluer, et interagir les uns avec les autres selon le graphe de dépendances du multi-modèle (voir Définition 11).

Définition 11 *Le **graphe de dépendances** d'un multi-modèle définit les interactions entre les modèles [Siebert, 2011].*

Un multi-modèle multi-aspect est doté d'une structure (voir Définition 12). Cette structure est classiquement fixe tout au long de l'évolution du multi-modèle. Le multi-modèle est alors appelé un **multi-modèle statique**. Cependant, la structure peut également être dynamique : les interactions entre les modèles peuvent changer, des modèles peuvent s'ajouter, s'enlever ou être remplacés dans le multi-modèle. Le multi-modèle est alors appelé un **multi-modèle dynamique** [Yilmaz et al., 2007]. Nous nous limitons dans cette thèse aux multi-modèles multi-aspects à structure statique.

Définition 12 *La **Structure** d'un multi-modèle correspond à l'ensemble des modèles actifs qui le composent et à son graphe de dépendances (e.g. Figure 3.1) [Yilmaz et al., 2007].*

La conception et la simulation d'un multi-modèle valide va, comme avec n'importe quel modèle, s'inscrire dans une démarche de modélisation et simulation (voir le Chapitre 2). Nous dirons alors que nous sommes dans une démarche de multi-modélisation et simulation. Cette démarche va impliquer les définitions successives des multi-modèles conceptuel, formel, exécutable, et numérique. La définition de chacun de ces quatre multi-modèles va requérir la mise en place de processus spécifiques. Dans la suite, nous détaillons les caractéristiques d'un multi-modèle à ces différents niveaux, les problèmes rencontrés et les différentes solutions qui peuvent être mises en place.

3.3 Le multi-modèle conceptuel

3.3.1 Caractéristiques

Comme nous l'avons vu au Chapitre 1, les systèmes complexes sont caractérisés par différents niveaux de structure et d'organisation évoluant à des échelles temporelles et spatiales distinctes. Lors d'une modélisation d'un système « classique », le thématicien va se positionner à un certain

niveau de détail pour décrire le système. Cependant, pour représenter un système complexe, le thématicien peut devoir se positionner simultanément à plusieurs niveaux (e.g. micro, meso, macro) [Gil-Quijano et al., 2012]. Ce peut être lorsqu'il y a un manque d'expressivité d'un niveau et qu'un autre est requis [El Hmam et al., 2006], quand les données disponibles sur le système sont étalées sur plusieurs niveaux [Gil-Quijano et al., 2012] ou quand la question de modélisation est explicitement d'étudier les influences mutuelles de ces niveaux [Duboz et al., 2003]. Enfin la raison peut être plus pragmatique car une vue macroscopique sur le système va considérer les données agrégées et donc permettre de diminuer le coût de calcul d'une simulation [Gaud et al., 2008, Vo et al., 2012, Xiong et al., 2009].

Le multi-modèle conceptuel va alors être constitué de modèles représentant le système à différentes échelles temporelles et spatiales, et à différents niveaux de résolution. Dans ce cas, le multi-modèle conceptuel est alors composé de représentations hétérogènes du système cible. On parlera alors de modélisation multi-niveau du système. Afin d'obtenir un multi-modèle conceptuel qui ait du sens par rapport au système cible, le thématicien doit alors veiller à assurer la cohérence de ces représentations [Turnitsa and Tolk, 2008, Tolk et al., 2007].

Différents types d'approches multi-niveaux existent. Elles varient selon les interactions entre les niveaux et la politique de changement de niveau choisie. Chacune de ces approches répond à des besoins de modélisation spécifiques. Dans [Davis and Hillestad, 1993], les auteurs donnent une classification de ces différentes approches de modélisation multi-niveau. Dans les sections suivantes, nous présentons ces approches.

3.3.2 Agrégation/Désagrégation

Dans une approche de type Agrégation/Désagrégation [Davis and Hillestad, 1993], deux types d'entités sont considérés : les entités de haute résolution (EHR) qui correspondent à une représentation micro du système et les entités de basse résolution (EBR), qui correspondent à une représentation macro du système.

Une entité n'est décrite ici qu'à un seul niveau à la fois. Cependant, à un instant donné, certaines entités sont décrites à un niveau macro alors que d'autres le sont à un niveau micro. Des opérations d'agrégation (permettant de passer de plusieurs EHRs à une EBR) et de désagrégation (permettant de passer d'une EBR à plusieurs EHRs) sont utilisées pour changer de niveau de résolution.

L'espace modélisé est séparé en différentes zones appelées play-boxes afin de déterminer la politique de changement de niveau. Les entités à l'intérieur d'une play-box sont toutes représentées à un même niveau de résolution. Nous décrivons dans la suite deux exemples représentant le système de cette manière.

Dans [El Hmam et al., 2006], l'approche de modélisation hybride du flux de trafic consiste à utiliser conjointement des modèles macro et micro d'un flux de trafic autoroutier pour profiter des avantages de chaque type de représentation. L'autoroute est divisée en plusieurs zones qui ne sont pas toutes modélisées au même niveau. Un modèle macro décrivant le système d'une manière agrégée est utilisé pour représenter les zones qui ne nécessitent pas un grand niveau de détail (une ligne droite par exemple). Dans ce modèle, qui évolue par rapport à un débit entrant de voitures, le trafic est représenté par un débit, une densité et une vitesse moyens. Le modèle permet de déterminer le débit sortant de voitures en fonction du temps. Les zones où le flux de trafic peut être sujet à des perturbations (e.g les entrées/sorties d'autoroute) sont représentées au niveau micro au moyen d'un modèle individu-centré. Dans ce modèle, les voitures sont représentées individuellement et évoluent le long de l'autoroute selon le comportement de leur conducteur. Le défi est ici d'assurer la transition entre des modèles conceptuels qui ne sont

pas de même nature, entre des individus et des données agrégées. Un agent [1] upward (de micro vers macro) et un agent downward (de macro vers micro) sont définis pour remplir ce rôle. Le premier a pour rôle de convertir les voitures sortant du modèle micro en débit entrant dans le modèle macro, alors que le second doit convertir le débit sortant du modèle macro en voitures entrant dans le modèle micro.

Dans la simulation de l'évacuation de la ville de Nha Trang [Vo et al., 2012], la principale difficulté est la gestion du grand nombre d'agents citoyens se déplaçant dans les rues qui réduit considérablement la vitesse de la simulation. Pour augmenter les performances de la simulation, les rues sont divisées en deux types de patchs : les patchs de carrefour et les patchs de milieu de route. Dans les premiers, les citoyens sont simulés individuellement alors que dans les seconds, un modèle macroscopique de flux de piétons est utilisé pour simuler leurs mouvements. Chacun des patchs est représenté comme un agent-route. Un méta-modèle organisationnel pour modélisation multi-niveau est utilisé pour assurer la transition entre les différents patchs. Les agents-routes utilisent des opérations de capture et de libération sur les agents citoyens. Un agent-route gère l'évolution de tous les agents citoyens qu'il a capturés. Les agents citoyens qui sont capturés perdent donc leurs comportements autonomes. Lorsqu'un citoyen est libéré, il retrouve son comportement autonome. Un agent-route capture les citoyens lorsqu'ils atteignent la bordure d'un patch milieu de route. Le patch détermine alors le mouvement des citoyens capturés en se basant sur le modèle macroscopique, et les relâche lorsqu'ils sortent de la bordure du patch.

3.3.3 Représentations Concurrentes

Dans un modèle à représentations concurrentes, les différents niveaux de résolution sont maintenus simultanément de manière concurrente et peuvent s'influencer l'un l'autre. Une entité est ici représentée simultanément dans les différents niveaux et est appelée une Entité Multi-Résolution (EMR) [Natrajan and Reynolds, 2001]. Une fonction de mappage doit être utilisée pour pouvoir passer d'un niveau à un autre [Reynolds et al., 1997].

L'intérêt d'une telle approche est alors qu'il est possible d'étudier les influences mutuelles entre les niveaux. Dans la suite, nous décrivons deux cas issus de la littérature représentant le système de cette manière et donc pouvant être rapprochés de cette approche.

Dans SIMPOP3 [Gil-Quijano et al., 2012], un modèle micro SIMPOPNano, décrivant l'évolution d'une ville est couplé avec un modèle mésoscopique, SIMPOP2, décrivant l'évolution d'un système de villes dans le but d'étudier les interactions micro/méso dans un système de villes. Le défi ici est d'assurer une traduction entre les niveaux : les concepts de fonction urbaine (de macro vers micro) et d'efficacité spatiale (de micro vers macro) sont utilisés pour assurer la traduction de l'état d'une ville d'un niveau à l'autre. Les deux modèles évoluent ensemble afin de représenter simultanément le système à différents niveaux.

Dans SimulBogota [Gil-Quijano et al., 2012], le défi est de modéliser l'évolution de l'occupation des logements de la ville de Bogotá tout en ayant un manque d'informations sur le comportement des entités de base (les ménages et les logements). Des entités mésoscopiques sont alors définies (des groupes de ménages et des groupes de logements) avec des comportements arbitraires. Les groupes sont détectés dynamiquement et réifiés au niveau mésoscopique à chaque pas de temps de simulation en utilisant un mécanisme de classification automatique des entités microscopiques. Une fois réifiés, les groupes de ménage se partagent les logements au moyen d'un mécanisme d'enchère et la population microscopique évolue en fonction de règles globales (décès, naissances).

1. Ces agents ne doivent pas être confondus avec les agents représentant les voitures dans le modèle micro.

3.3.4 Synthèse

Bien que ces différentes approches de modélisation multi-niveau n'ont pas affaire aux mêmes problèmes, elles doivent toutes répondre à la même question au niveau conceptuel. En effet, tous ces exemples utilisent deux modèles pour décrire le même système mais à différents niveaux de représentation (micro et macro). La difficulté est alors de concilier ces deux types de représentation, c'est à dire étant donné les données de sortie d'un modèle représentant le système à un niveau de représentation, quelles sont les opérations qui doivent être utilisées pour traduire ces données à un autre niveau de représentation ?

3.4 Le multi-modèle formel

3.4.1 Caractéristiques

Lorsqu'un modélisateur va devoir faire le choix d'un formalisme pour construire son multi-modèle formel, il peut se heurter à l'hétérogénéité du système qu'il veut représenter (voir Chapitre 1). A cause de cette hétérogénéité, il se peut que le système présente des caractéristiques discrètes et continues qu'aucun formalisme classique n'est adapté à représenter [Cellier, 1979]. Chaque modèle composant le multi-modèle est alors écrit dans le formalisme le plus adapté à la partie du système qu'il représente.

Par exemple, il se peut que le système évolue à la fois selon des dynamiques discrètes et continues. Dans EPOCH [Hopkinson et al., 2006], les auteurs modélisent un réseau électrique intelligent (smart grid) composé de trois parties distinctes en interactions : la production d'un réseau électrique est monitorée et contrôlée par un système de décisions par l'intermédiaire d'un réseau de communication. Tandis que le système de décision et le réseau de communication évoluent de manière discrète (le premier est modélisé par un agent intelligent, et le deuxième est représenté par un modèle événementiel), le réseau électrique va évoluer de manière continue (représenté par un modèle équationnel).

Dans d'autre cas, la dynamique du système peut être continue sauf à certains points où des discontinuités aparaissent. Dans [Fishwick and Zeigler, 1992], l'évolution de l'état d'un récipient contenant un mélange d'eau et de produit vaisselle posé sur une plaque chauffante munie d'un interrupteur est représentée. Lorsque l'interrupteur est sur ON, le récipient chauffe. L'eau se met à bouillir à une température de 100 degrés Celcius et de la mousse se forme à la surface à cause du liquide vaisselle. Le liquide peut alors déborder du récipient. Lorsque l'interrupteur est sur OFF, le récipient refroidit jusqu'à la température ambiante de la pièce et la mousse diminue. Quatre variables servent à décrire l'état du système : l'état de l'interrupteur de la plaque chauffante (ON ou OFF), la température de l'eau, la hauteur de l'eau dans le récipient, et la hauteur de la mousse dans le récipient. L'espace d'état du système est décomposé en six phases chacune étant décrite par une équation différente. On trouve alors les phases de : chauffe, refroidissement, ébullition, débordement et absence de liquide. Les transitions entre les phases sont définies par un automate à états finis dont chaque état correspond à une phase et les transitions correspondent aux changements de phases.

Pour résumer, un multi-modèle peut être composé d'un ensemble hétérogène de modèles formels qui vont potentiellement décrire le système de manière discrète et continue, et qui vont être exécutés selon des politiques différentes (e.g. pas de temps cyclique et acyclique, événementielle). Afin de construire le multi-model formel, le modélisateur va devoir gérer l'intégration des différents formalismes qui le composent.

3.4.2 Intégration de formalismes hétérogènes : solutions existantes

Trois solutions sont possibles pour intégrér des formalismes hétérogènes [Vangheluwe et al., 2002] :

- **Traduire les modèles formels dans un même formalisme commun.** Dans [Vangheluwe et al., 2002], les auteurs définissent un graphe de transformation de formalisme permettant la traduction d'un ensemble de modèles formels hétérogènes dans un formalisme commun (voir Figure 3.2). L'avantage de telles approches est qu'elles rendent le multi-modèle formel homogène. Celui-ci est alors prêt à être implémenté et simulé grâce au simulateur abstrait dudit formalisme. Le point faible de ces approches est qu'elles obligent à réécrire et à réimplémenter des modèles valides. Elles entachent donc la réutilisabilité des modèles, et peuvent introduire de nouvelles erreurs.
- **Décrire le multi-modèle à l'aide d'un super-formalisme**, c'est à dire un formalisme pouvant intégrer les différents formalismes du multi-modèle. Ce super-formalisme doit permettre de décrire explicitement les interactions entre des modèles discrets et continus. De plus, afin de permettre la simulation du multi-modèle, le super-formalisme doit proposer un simulateur abstrait permettant d'exécuter le multi-modèle en tenant compte des politiques d'exécution potentiellement différentes des différents formalismes. Le formalisme DEV&DESS [Praehofer, 1991] est un exemple de super-formalisme (voir le Chapitre 4). L'avantage de cette approche est qu'elle permet de formaliser explicitement un multi-modèle hétérogène. En revanche, son point faible est que, comme lors de la traduction de modèles dans un formalisme commun, elle oblige à réécrire et à réimplémenter des modèles valides. Cependant, des solutions existent pour réutiliser des modèles pré-existants déjà implémentés (voir le Chapitre 4).
- **Effectuer une co-simulation** où chaque modèle est exécuté par son propre simulateur, et les interactions entre les modèles correspondent aux échanges de données entre les simulateurs. L'intégration des formalismes n'est pas alors faite explicitement au niveau du multi-modèle formel mais directement au moment de la simulation : les résultats de simulation produits par un logiciel de simulation servent à en paramétrer un autre. L'avantage de ces approches est qu'elles permettent de réutiliser des modèles valides déjà implémentés dans des logiciels de simulation différents. En revanche, le point faible de la co-simulation est qu'elle ne définit pas de multi-modèle formel et repousse donc le problème du multi-formalisme au niveau des multi-modèles exécutable et numérique qui doivent être définis de manière ad hoc.

3.5 Le multi-modèle exécutable

Une fois un multi-modèle formalisé, l'informaticien va devoir rendre le multi-modèle exécutable pour pouvoir le simuler. Lorsque les modèles qui composent le multi-modèle doivent co-évoluer, l'informaticien doit mettre en place un simulateur abstrait pour gérer l'exécution du multi-modèle. Un multi-modèle étant modulaire par nature, différents types d'exécution sont possibles [Perumalla, 2007] :

- Dans une **exécution séquentielle**, le multi-modèle est exécuté sur un seul processeur. Ce type d'exécution est le plus simple à mettre en place, les modèles étant exécutés par le simulateur abstrait du multi-modèle. Cependant, les simulations séquentielles ne supportent pas le passage à l'échelle : lorsque la taille du multi-modèle augmente, la durée de la simulation peut rapidement devenir un frein à l'expérimentation.

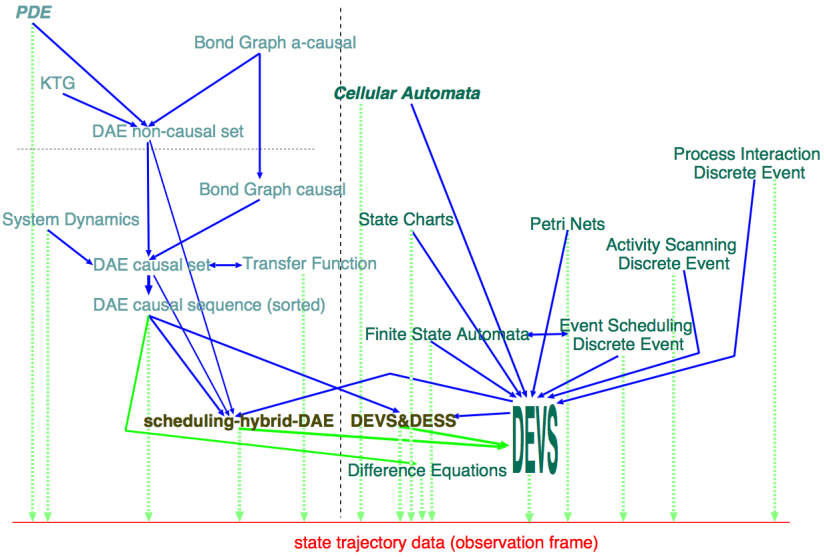


FIGURE 3.2 – Le graphe de transformation des formalismes. Les flèches continues indiquent l'existence d'une transformation d'un formalisme à un autre. Les flèches pointillées verticales indiquent l'existence d'un simulateur permettant la production de résultats de simulation (Source [Vangheluwe et al., 2002]).

— Dans une **exécution parallèle**, le multi-modèle est exécuté sur plusieurs processeurs partageant une même mémoire. Les modèles du multi-modèle sont alors exécutés en parallèle chacun sur un processeur distinct. Ce type d'exécution a l'avantage d'augmenter la rapidité d'exécution de la simulation, et donc de permettre le passage à l'échelle. Cependant, les simulations parallèles sont plus complexes à mettre en place car elles requièrent un algorithme de simulation parallèle. Cet algorithme a pour but de coordonner les exécutions des modèles dans le temps simulé en tenant compte de la contrainte de causalité (voir Définition 13).
— Dans une **exécution distribuée**, le multi-modèle est exécuté sur plusieurs processeurs comme avec une exécution parallèle, mais sans partage de mémoire. En plus de permettre le passage à l'échelle comme avec une simulation parallèle, les simulations distribuées sont plus flexibles, car elle permettent l'exécution du multi-modèle sur des machines différentes qui utilisent des systèmes d'exploitation différents et qui sont potentiellement géographiquement séparées. Cependant, les simulations distribuées requièrent, en plus d'un algorithme de simulation parallèle, l'utilisation d'un intergiciel de communication pour gérer les communications inter-machines.

Définition 13 *La **contrainte de causalité** est respectée lorsque les résultats de la simulation parallèle d'un modèle m sont les mêmes que ceux fournis par l'exécution séquentielle de ce même modèle m [Siebert, 2011] d'après [Fujimoto, 2001].*

3.6 Le multi-modèle numérique

Pour implémenter et simuler un modèle, chaque domaine d'application va disposer d'outils de simulation spécifiques. Ces outils, qui ne sont en général pas faits pour communiquer ensemble, peuvent être écrits dans des langages de programmation différents (Java, C++, Python,

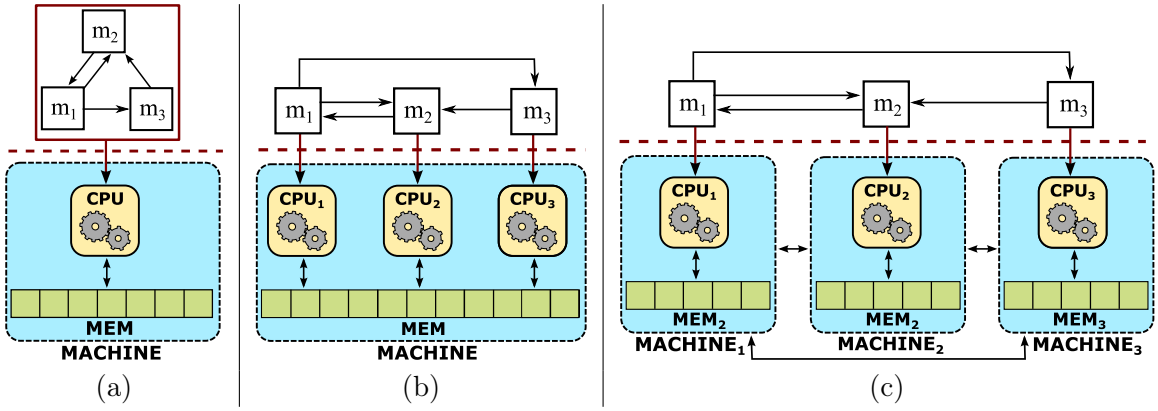


FIGURE 3.3 – Les différents types d'exécution d'un multi-modèle. (a) Une simulation séquentielle. (b) Une simulation parallèle. (c) Une simulation distribuée.

Fortran...), être compatibles avec des systèmes d'exploitation différents (GNU/Linux, Windows, Mac OS, en 32 ou 64bits...), et être disponibles sur un équipement spécifique (e.g. sur une machine disposant d'une licence d'utilisation de l'outil). La Table 3.1, illustre cette hétérogénéité logicielle à travers une liste non exhaustive d'outils de simulation classés par domaine d'application.

Domaines	**Outils de simulation**	**Langages**	**Systèmes d'exploitation**
Déplacements collectifs	NetLogo [Wilensky, 1999]	API Java et Scala	GNU/Linux, Windows, Mac OS
	GAMA [Taillandier et al., 2012]	Java	GNU/Linux, Windows, Mac OS
Réseaux Télécom	NS-3 [Henderson et al., 2006]	C++, API Python	GNU/Linux
	OMNeT++ [Varga and Hornig, 2008]	C++	GNU/Linux, Windows, Mac OS
Robotique	VREP	C/C++, Lua, Python, Java	GNU/Linux, Windows, Mac OS
Physique	Dymola	Code propriétaire	Windows
	Matlab/Simulink	Code propriétaire, API C/C++ et Fortran	GNU/Linux, Windows, Mac OS

TABLE 3.1 – Exemples de domaines d'application et certains de leurs outils de simulation.

Un multi-modèle couvrant plusieurs domaines d'application peut alors être composé de modèles implémentés dans des outils de simulation différents. Le multi-modèle est donc composé d'un ensemble hétérogène de modèles numériques. La simulation du multi-modèle se fera alors

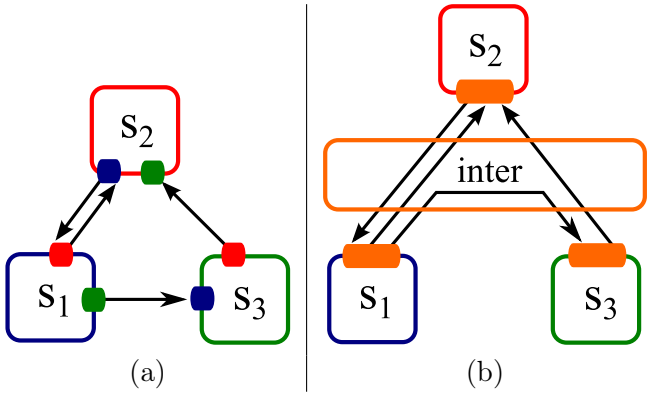


FIGURE 3.4 – Les différentes approches de gestion de l'interopérabilité appliquées à un exemple de multi-modèle numérique composé de trois outils de simulation s_1, s_2 et s_3. Les carrés colorés représentent les modifications apportées aux outils de simulation pour assurer leur interopérabilité (a) une approche ad-hoc (b) une approche avec un intergiciel *inter*.

par une co-simulation, c'est à dire en exécutant conjointement ces différents outils de simulation, et en gérant les échanges de données entre ces simulateurs.

Un programmeur voulant implémenter un multi-modèle exécutable pour le simuler va alors devoir gérer l'interopérabilité de ces outils de simulation afin de simuler le multi-modèle (voir Définition 14). Cette interopérabilité peut être assurée de manière ad-hoc, en modifiant directement les outils de simulation pour les rendre interopérables (Figure 3.4a), ou alors en utilisant un intergiciel. Cette dernière solution a l'avantage d'être plus flexible, car les outils de simulation n'ont pas à être interopérables entre eux, mais doivent simplement être interopérables avec l'intergiciel (Figure 3.4b). Contrairement à une approche ad-hoc, l'approche par intergiciel permet alors l'ajout, la suppression ou le changement d'un outil de simulation au multi-modèle numérique sans impacter les outils de simulation déjà intégrés à l'intergiciel.

Définition 14 ***L'interopérabilité*** *est la capacité de plusieurs systèmes (ou composants) de s'échanger de l'information, et d'utiliser l'information qui a été échangée [Diallo et al., 2011].*

3.7 Conclusion

A partir de ce que nous avons vu dans ce chapitre, nous faisons l'hypothèse dans cette thèse que le modèle d'un système complexe correspond à un multi-modèle. Nous avons alors vu que la conception et la simulation d'un multi-modèle vont poser, à chaque étape, un ensemble de défis spécifiques. Ces défis consistent à assurer :

— **La cohérence des représentations** pour le multi-modèle conceptuel.
— **L'intégration des formalismes** pour le multi-modèle formel.
— **La coordination de la simulation** pour le multi-modèle exécutable.
— **L'interopérabilité des simulateurs** pour le multi-modèle numérique.

La conception et la simulation d'un multi-modèle nécessite alors la mise en place d'un ensemble de solutions spécifiques aux caractéristiques des multi-modèles conceptuel, formel, exécutable et numérique. A ces contraintes, vient s'ajouter le besoin comme dans une démarche de modélisation et simulation classique, de vérifier le multi-modèle numérique de manière systématique (voir Chapitre 2).

Une première approche naïve de construction d'un multi-modèle pourrait être, étant donné l'ensemble des modèles devant interagir, de construire le multi-modèle de manière ad hoc. Cependant, il ne faut pas oublier que cette construction s'inscrit dans une démarche de multi-modélisation et simulation. Dans cette optique, le multi-modèle obtenu initialement n'est potentiellement pas valide, à cause de mauvais choix de modélisation ou d'erreurs de passage de niveau. Le multi-modèle n'est alors le produit que d'une première itération de la démarche et non son produit final.

En conséquence, **le modélisateur peut être amené à modifier le multi-modèle qu'il a implémenté.** Ces modifications peuvent être légères comme un changement des paramètres d'un ou plusieurs modèles, mais elles peuvent également être plus profondes comme un changement de structure du multi-modèle (ajout/suppression de modèles) voire un changement du type du multi-modèle (e.g. passer d'une structure statique à une structure dynamique) [Tisseau, 2001]. De nouvelles contraintes vont alors s'ajouter au multi-modèle (e.g. prendre en compte une nouvelle politique d'exécution ou gérer l'interopérabilité du multi-modèle avec un nouvel outil de simulation). **Si le multi-modèle a été développé initialement de manière ad-hoc, l'ajout d'une ou plusieurs contraintes peut amener non pas à une modification du multi-modèle, mais à sa redéfinition complète.** Le multi-modèle peut alors ne pas s'affiner à chaque itération de la démarche de multi-modélisation et simulation, et cette dernière peut alors ne pas converger.

En conclusion, les méthodes ad hoc ne sont pas suffisantes pour construire un multi-modèle dans une démarche rigoureuse de multi-modélisation et simulation. Il est alors nécessaire de bénéficier d'une approche de conception et simulation de multi-modèle qui apporte un ensemble de solutions génériques aux problèmes de la démarche de multi-modélisation et simulation. Dans le chapitre suivant, nous évaluons la capacité des approches de multi-modélisation existantes à remplir ce rôle.

Chapitre 4

Solutions existantes de multi-modélisation et leurs limites

Sommaire

4.1 Introduction

Il existe dans le domaine de la M&S plusieurs approches offrant des solutions intéressantes aux défis de la démarche de M&S des systèmes complexes (présentés dans le chapitre précédent). Ces approches sont le fruit de plusieurs années (voir décennies) de recherche, et constituent alors des aboutissements scientifiques qu'il est, pour nous, important de réutiliser.

Toutes ces approches permettent d'effectuer une simulation d'un système complexe, et gèrent donc l'exécution d'un multi-modèle. Cependant, en ce qui concerne l'intégration de l'hétérogénéité, chacune de ces approches cible en général un niveau particulier de question, et n'offre donc qu'une solution partielle à notre problème global. Nous pensons toutefois que certaines de ces approches peuvent être complémentaires et qu'elles peuvent être combinées pour offrir des solutions à tous les niveaux, et ainsi servir de support à la démarche de M&S des systèmes complexes.

Dans ce chapitre, nous détaillons et évaluons certaines des approches existantes de la M&S des systèmes complexes. Ce chapitre n'a pas pour vocation de faire une revue exhaustive de ces approches. Nous avons choisi ici de nous focaliser sur les travaux qui font autorité dans le domaine de la M&S, et ceux que nous considérons comme les plus aboutis.

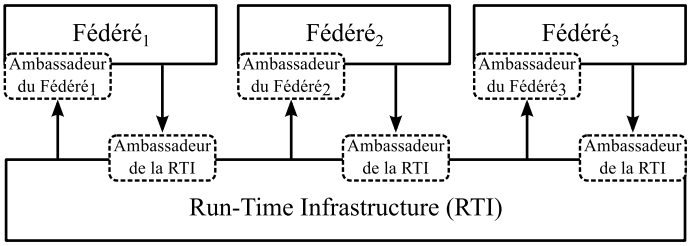


FIGURE 4.1 – Exemple de fédération HLA

Nous faisons également le choix de classifier ces approches selon le niveau de question qu'elles ciblent :

— la Section 4.2 détaille tout d'abord les solutions permettant de gérer l'interopérabilité de plusieurs logiciels de simulation.
— la Section 4.3 présente ensuite les solutions permettant d'intégrer des modèles écrits dans différents formalismes.
— la Section 4.4 présente enfin les solutions permettant d'assurer dans un multi-modèle, la cohérence des différentes représentations du système cible.

Pour chaque solution, nous détaillons son domaine d'application, ses principes généraux, et enfin, ses avantages et limites au regard de la démarche de multi-modélisation et simulation. Nous finissons enfin par une synthèse permettant de comparer les approches évaluées. Cette synthèse nous permet à la fois de sélectionner les approches les plus intéressantes, et d'évaluer leur complémentarité.

4.2 Les solutions d'interopérabilité des logiciels de simulation

Dans cette section, nous détaillons les deux approches offrant les solutions les plus complètes au problème de l'interopérabilité des logiciels de simulation. Nous présentons tout d'abord la High Level Architecture (HLA) qui est, depuis la fin des années 90, une norme incontournable dans ce domaine (Section 4.2.1). Nous présentons ensuite le standard émergent et très prometteur Functional Mockup Interface (FMI) (Section 4.2.2). Ces deux approches diffèrent dans leur manière d'aborder l'interopérabilité : HLA propose une spécification haut niveau d'intergiciel de co-simulation, alors que FMI standardise les interfaces logicielles des simulateurs.

4.2.1 La High Level Architecture

HLA est une spécification d'architecture d'intergiciel de co-simulation événementielle distribuée [Dahmann et al., 1997]. HLA a été initialement développé par le Departement of Defense (DoD) américain et a depuis été normalisé par l'IEEE.

Dans HLA, un ensemble de logiciels de simulation associés pour effectuer une co-simulation est appelé une fédération. Chaque logiciel de simulation est alors appelé un fédéré. Pour simuler le système cible, la fédération va s'appuyer sur les services proposés par un intergiciel de simulation appelé Run-Time Infrastructure (RTI). Les échanges entre un fédéré et la RTI sont assurés par deux interfaces appelées ambassadeurs [Carothers et al., 1997] :

— **L'ambassadeur de la RTI** est une interface fournie par la RTI aux fédérés. Les fédérés peuvent se servir de cet ambassadeur pour utiliser les services de la RTI.
— **L'ambassadeur du fédéré** est une interface qui doit être définie pour chaque fédéré. Cette interface est utilisé par la RTI pour remonter des informations au fédéré.

La spécification HLA est principalement composée de trois parties : les règles [IEEE, 2010b], l'interface [IEEE, 2010a], et l'Object Model Template (OMT) [IEEE, 2010c].

Les règles HLA définissent les principes généraux de HLA. Ces principes sont définis pour assurer l'interopérabilité de plusieurs logiciels de simulation, tout en ayant à modifier au minimum ces derniers, et en s'assurant de pouvoir les réutiliser a posteriori pour d'autres applications [Adelantado, 2012]. Les règles HLA sont divisées en deux groupes : les règles pour les fédérés et les règles pour la RTI. Les règles pour les fédérés stipulent les caractéristiques que ceux-ci doivent avoir pour pouvoir intégrer une fédération HLA. Les règles pour la RTI stipulent le fonctionnement qu'un intergiciel de simulation doit respecter pour être conforme avec HLA.

L'OMT permet de décrire de manière normalisée les échanges de données entre les fédérés. Les communications au sein d'une fédération sont décrites par des ensembles de quatorze tables [Taylor, 2003] composant la SOM (Simulation Object Model) de chaque fédéré, et la FOM (Federation Object Model) de la fédération. La SOM d'un fédéré décrit toutes les données que celui-ci peut mettre à disposition d'une fédération. Quant à elle, la FOM décrit quelles sont les données qui sont échangées au sein d'une fédération. HLA considère que les données partagées par les fédérés sont décrites sous la forme d'objets (e.g. des piétons, des voitures). Ainsi, la FOM et les SOM décrivent les classes des objets qui peuvent être échangés (e.g. les classes piétons et voitures). Il est important de noter que les objets HLA décrivent uniquement les données échangées par les fédérés et non comment ces données sont représentées en interne par les fédérés. En conséquence (1) HLA ne présuppose absolument pas que les fédérés soient conçus avec une méthode orientée objet [Adelantado, 2012], et (2) le concept d'objet dans HLA diffère légèrement de celui utilisé dans une conception orientée objet. En effet, les objets HLA sont uniquement définis par leurs attributs car ils ne sont pas dotés de méthodes.

L'interface HLA définit quels sont les services que la RTI propose aux fédérés et comment ceux-ci vont pouvoir les utiliser. Ces services sont relatifs à six catégories [Dahmann et al., 1997].

- **La gestion de la fédération :** dans HLA, la création et la suppression d'une fédération sont effectuées par les fédérés. De plus, un fédéré peut à n'importe quel moment rejoindre et quitter une fédération existante. La RTI propose alors un ensemble de services pour gérer ces aspects.
- **La déclaration de communication :** les communications dans HLA fonctionnent selon un mode Publisher/Subscriber. A tout moment, un fédéré peut informer la RTI de son intention de publier des mises à jour sur une classe d'objets. De la même manière, à tout moment, un fédéré peut demander à être mis au courant de toute mise à jour faite sur une classe d'objets.
- **La gestion des objets** permet à un fédéré de créer, supprimer ou mettre à jour des instances d'une classe d'objets (e.g. ajouter/supprimer un piéton ou modifier les coordonnées d'une voiture). Ce service permet également de notifier ces modifications aux fédérés abonnés à la classe d'objets mise à jour.
- **La gestion des possessions :** afin d'éviter des modifications concurrentes d'objets, HLA autorise seulement un fédéré à la fois à mettre à jour les objets d'une classe donnée. Ce fédéré est alors le propriétaire de la classe d'objets. La RTI fournit aux fédérés les moyens de prendre possession d'une classe d'objets, de relacher une classe d'objets, et de transférer la propriété d'une classe d'objets d'un fédéré à un autre.

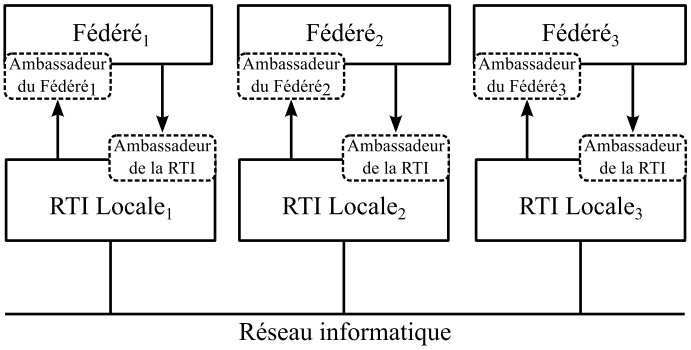


FIGURE 4.2 – Exemple d'une implémentation distribuée de la RTI

— **La gestion du temps** permet de synchroniser les exécutions des fédérés dans le temps simulé au moyen d'un algorithme d'exécution parallèle (voir Chapitre 3), ou d'assurer une exécution temps réel de la fédération.
— **Le routage des données :** permet de distribuer les données aux fédérés pendant la simulation en passant par un réseau informatique. Ce service a pour but d'optimiser les transmissions de données entre un fédéré et la RTI en diminuant les transmissions de données non pertinentes pour le fédéré.

HLA permet de définir un intergiciel qui servira de base logicielle à l'exécution d'un multi-modèle : il va permettre de gérer l'interopérabilité d'outils de simulation différents, et de synchroniser leurs exécutions en se basant sur une exécution événementielle des simulateurs. HLA permet alors la réutilisation de modèles valides déjà implémentés.

Cependant, l'interopérabilité des logiciels de simulation est en fait limitée avec HLA pour deux raisons. Tout d'abord, il faut noter que la norme HLA définit une architecture logicielle de haut niveau au sens où elle spécifie le fonctionnement général d'une RTI et non son implémentation [Dahmann and Morse, 1998]. En conséquence, différentes implémentations de RTI existent. Ces implémentations sont faites dans des langages de programmation différents, sont compatibles avec des systèmes d'exploitation différents et sont monolithiques ou distribuables sur plusieurs machines (Figure 4.2) [Adelantado, 2012]. De plus, ces logiciels implémentent souvent HLA de façon très partielle à cause de la forte complexité de la norme. C'est pourquoi, un logiciel de simulation n'est pas en réalité rendu interopérable avec HLA, mais avec une implémentation particulière de la RTI. La conséquence est que tous les logiciels de simulation devant intégrer une même fédération doivent être compatibles avec la même implémentation de la RTI.

Ensuite, comme HLA se positionne au niveau logiciel, il définit uniquement les interactions dans un multi-modèle comme des échanges syntaxiques de données entre différents logiciels de simulations. HLA ne donne pas alors de sémantique aux interactions entre les modèles eux-mêmes, et ne normalise donc pas ces interactions [Taylor et al., 2006]. Le problème est que, pour envoyer ou recevoir des données dans le cadre d'une même interaction entre deux modèles, deux fédérés peuvent utiliser les nombreux services de la RTI de différentes manières [Taylor, 2007]. Or, pour pouvoir s'échanger des données au travers de la RTI, les fédérés doivent utiliser les services de cette dernière de la même manière, sans quoi ils ne pourront pas communiquer [Taylor, 2003]. La conséquence est que, deux fédérés rendus séparément interopérables avec HLA peuvent en fait ne pas être interopérables entre eux s'il n'ont pas été conçus pour partager et recevoir des données avec la RTI de manière similaire.

Enfin, il faut noter que HLA ne permet pas de représenter formellement un multi-modèle car il ne se base pas sur un formalisme de modélisation, mais uniquement sur le modèle d'exécution

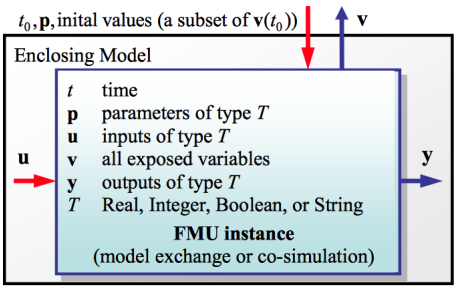


FIGURE 4.3 – Exemple de FMU avec ses entrées (flèches rouges) et sorties (flèches bleues). (Source [Blochwitz et al., 2012])

événementiel. Le problème pour la multi-modélisation est alors que HLA ne fournit pas de réponse quant à l'intégration de formalismes hétérogènes au sein d'un multi-modèle. L'approche prend du sens alors uniquement pour les formalismes événementiels [Vangheluwe et al., 2002].

4.2.2 La Functional Mockup Interface

Le standard Functional Mockup Interface (FMI) propose une interface logicielle générique pour les modèles équationnels et leurs simulateurs [Blochwitz et al., 2011]. Le but de ce standard est double : d'une part permettre de manière simple l'échange de modèles déjà implémentés dans des outils différents, et d'autre part faire en sorte que des modèles puissent être utilisés comme des boîtes noires, en protégeant ainsi leurs contenus [Blochwitz et al., 2011]. Cette dernière motivation est utile dans un contexte industriel où les modèles constituent des données sensibles.

Un modèle implémentant la norme FMI est appelé une Functional Mockup Unit (FMU). Parmi les variables de son modèle, une FMU va définir celles dont les valeurs sont accessibles (les variables de sortie), et celles dont les valeurs peuvent être fixées (les variables d'entrée) (Figure 4.3). L'interface de la FMU est principalement composée d'un ensemble de fonctions en langage C, et d'un fichier de description XML. Les fonctions C vont permettre de contrôler l'exécution de la FMU, d'accéder à la valeur de chaque variable de sortie, et de fixer la valeur de chaque variable d'entrée. Le fichier XML va quant à lui décrire le modèle de la FMU et son interface (e.g. variables d'entrée/sortie).

Pour supporter le standard FMI, les outils de simulation doivent se doter de la capacité (1) d'exporter leurs modèles sous la forme de FMUs et/ou (2) d'importer des FMUs existantes pour les utiliser comme des composants. Le standard FMI prévoit que ces exportations/importations puissent se faire à deux niveaux [Blochwitz et al., 2012] :

— **FMI pour l'échange de modèle** permet d'exporter un modèle sans son solveur dans une FMU. Le modèle correspond alors à une bibliothèque partagée C décrite de manière suffisamment précise par un fichier XML pour permettre son exécution par un solveur externe. De cette manière, le modèle pourra être partagé et simulé dans d'autres outils de simulation en utilisant n'importe quel solveur adapté.

— **FMI pour la co-simulation** permet au contraire d'encapsuler dans une bibliothèque partagée C un modèle avec son solveur. De cette manière, le modèle pourra être utilisé dans une co-simulation. Les FMUs seront alors considérées comme des esclaves dont les exécutions seront coordonnées par un maître de co-simulation. Dans une co-simulation de FMUs, celles-ci vont échanger des données uniquement à une suite de points de communication discrets dans le temps simulé. Ces points de communication, qui peuvent être réguliers ou non, sont donnés par le maître de co-simulation. Entre ces points de commu-

nication, chaque FMU va évoluer dans le temps simulé grâce à son solveur. Il est à noter que le standard FMU ne donne pas de spécification pour le maître de co-simulation.

FMI est un standard très prometteur pour la gestion de l'interopérabilité de différents outils de simulation. Cependant, il comporte plusieurs limites. Notamment, la mise en place d'un maître de co-simulation est souvent très complexe car FMI laisse beaucoup de libertés aux comportements des FMUs. Par exemple, lorsqu'un master de co-simulation donne un nouveau point de communication à plusieurs FMUs, chacune d'elles est libre de (1) accepter le pas de communication et s'exécuter, ou (2) ne pas s'exécuter et rejeter le pas de communication si celui-ci est trop grand pour donner des résultats valides (voir Chapitre 2), ou (3) s'exécuter d'un pas de temps plus petit que le pas de communication [Broman et al., 2013]. L'algorithme de simulation du master de co-simulation doit alors prendre en compte cette hétérogénéité de comportements des FMUs.

Si le standard FMI est intéressant pour gérer l'interopérabilité des logiciels de simulation dans un multi-modèle, il présente plusieurs lacunes par rapport à la M&S des systèmes complexes. En effet, le positionnement du standard est uniquement logiciel et se limite aux formalismes des équations différentielles. Les interactions entre les modèles sont alors uniquement vues comme un échange purement syntaxique de variables sans tenir compte de leur sémantique. FMI ne fournit pas alors de solution à l'intégration de formalismes et de représentations hétérogènes au sein d'un multi-modèle.

4.3 Solution d'intégration des formalismes hétérogènes

Dans cette section, nous détaillons les approches faisant autorité en matière d'intégration de modèles formels hétérogènes. Nous présentons tout d'abord l'approche $AToM^3$ qui permet de traduire les différents modèles formels dans un formalisme commun (Section 4.3.1). Nous présentons ensuite l'approche Ptolemy II qui permet d'intégrer différents modèles formels dans une hiérarchie hétérogène en unifiant leurs exécutions grâce à un langage abstrait à base d'acteurs (Section 4.3.2). Enfin, nous détaillons le formalisme de modélisation Discrete EVent System specification (DEVS) qui a la particularité d'être suffisamment générique pour pouvoir intégrer tous les autres formalismes de modélisation (Section 4.3.3).

4.3.1 AToM3

$AToM^3$ [Lara and Vangheluwe, 2002] est un logiciel de modélisation multi-paradigme, c'est à dire permettant de décrire un modèle selon plusieurs formalismes. Une fois un modèle formalisé, $AToM^3$ permet de le traduire automatiquement dans un autre formalisme. Il est alors possible de combiner plusieurs modèles décrits dans des formalismes différents en les traduisant dans un formalisme commun. Le formalisme commun le plus proche de plusieurs modèles peut être trouvé en se basant sur le graphe de transformation de formalisme (comme vu dans la Figure 3.2 du Chapitre 3). Une fois ce multi-modèle formel homogénéisé, celui-ci pourra être simulé par un simulateur du formalisme commun choisi.

Les transformations de formalisme sont rendues possibles dans $AToM^3$ par la modélisation des formalismes eux-mêmes comme des graphes syntaxiques. $AToM^3$ va alors permettre de décrire la traduction d'un modèle dans un autre formalisme comme une transformation de graphe. Ces transformations de graphe sont spécifiées en utilisant le formalisme de grammaire de graphe [Mosterman and Vangheluwe, 2002]. La grammaire de graphe permet de définir des règles reliant un graphe cible à un graphe résultat. La transformation d'un modèle d'un formalisme vers un autre va alors être spécifiée par un ensemble de règles. Pour appliquer une transformation sur

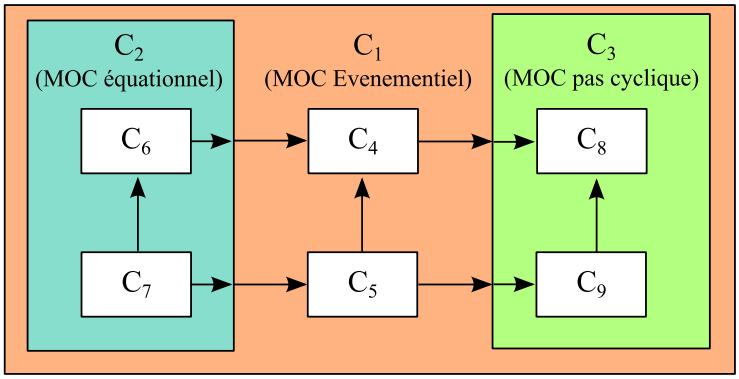


FIGURE 4.4 – Exemple de hiérarchie hétérogène composée de trois MOC.

un graphe donné (i.e pour traduire un modèle dans un autre formalisme), $AToM^3$ va identifier les parties de ce graphe qui correspondent au graphe cible d'une des règles. La partie du graphe identifiée sera alors remplacée par le graphe résultat de la règle [De Lara and Vangheluwe, 2002]. La transformation sera terminée quand aucune règle ne pourra plus être appliquée.

$AToM^3$ offre une solution intéressante à l'intégration de plusieurs formalismes dans un multi-modèle. De plus, il permet de passer automatiquement d'un multi-modèle à sa simulation. Cependant, le logiciel limite la réutilisation de modèles pré-existants car ceux-ci doivent être réécrits dans le logiciel. De plus $AToM^3$ ne permet pas de réutiliser des modèles déjà implémentés dans des outils de simulation différents.

4.3.2 Ptolemy II

Ptolemy II [Eker et al., 2003] permet de décrire et simuler un système comme une hiérarchie hétérogène. Dans cette optique, un système est décrit comme un ensemble de composants en interaction. La manière dont ces composants vont s'exécuter et communiquer va être définie par un Modèle de Calcul (MOC). Un MOC peut correspondre par exemple à une exécution événementielle, à pas de temps cyclique, au comportement d'un automate à états finis, à des interactions de composants équationnels, ou encore à une exécution de flux de données [Brooks et al., 2008b]. La vision hiérarchique tient au fait que chacun de ces composants peut être soit atomique, soit composite, c'est à dire composé lui même d'un ensemble de composants qui peuvent être à leur tour composites, et ainsi de suite. L'hétérogénéité vient du fait que chaque composant composite dans la hiérarchie peut utiliser son propre MOC pour définir comment ses composants s'exécutent et interagissent (Figure 4.4).

Ptolemy II se base sur une conception orientée acteur : l'exécution du programme est effectuée par un ensemble de processus concurrents (les acteurs) interagissant par échanges de messages. Chaque composant de la hiérarchie est alors implémenté par un acteur (composite ou atomique selon le type du composant). Dans Ptolemy, tous les acteurs sont décrits par une syntaxe abstraite commune : un acteur est doté d'une interface composée de ports d'entrée et de sortie pour les messages. Ainsi, la hiérarchie dans Ptolemy est décrite syntaxiquement de manière homogène : le système est décrit par un ensemble d'acteurs imbriqués et reliés par leurs ports d'entrée/sortie [Brooks et al., 2008a].

Cependant, la sémantique de cette description est hétérogène car elle change avec le MOC de chaque acteur composite de la hiérarchie. Un MOC dans Ptolemy II est implémenté par un domaine. Un domaine est composé d'un réalisateur et de plusieurs récepteurs. Le réalisateur est

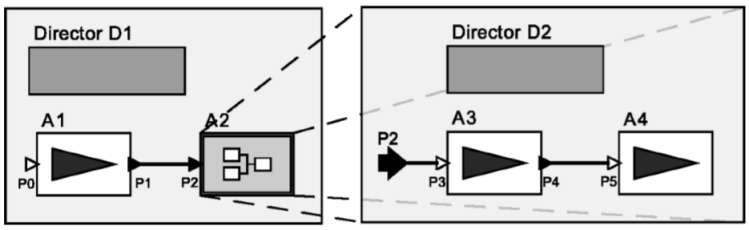


FIGURE 4.5 – Exemple de modèle hiérarchique dans Ptolemy II. $A1$, $A3$ et $A4$ sont des acteurs atomiques alors que $A2$ est un acteur composite. $D1$ et $D2$ sont des réalisateurs.

chargé de gérer l'exécution des acteurs de son acteur composite en accord avec la sémantique du MOC. Les récepteurs gèrent les communications entre les acteurs (au niveau de leurs ports d'entrée) de l'acteur composite en accord avec la sémantique du MOC [Lee, 2010]. La Figure 4.5 montre un exemple de hiérarchie hétérogène dans Ptolemy II.

Un réalisateur va contrôler tous ses acteurs de la même manière, qu'ils soient atomiques ou composites grâce à une sémantique abstraite [Lee, 2010]. Cette sémantique abstraite définit un ensemble d'actions que tout acteur doit implémenter :

— **preinitialize()** initialise l'acteur.
— **intialize()** initialise l'état de l'acteur.
— **prefire()** détermine si l'acteur peut être exécuté.
— **fire()** l'acteur produit des sorties en fonction de ses entrées, mais ne change pas d'état.
— **postfire()** l'acteur change d'état en fonction de ses entrées.
— **wrapup()** est appelé à la fin de la simulation.

Pour un acteur composite, la réalisation de ces actions va consister à déclencher les actions de certains de ses acteurs en fonction de son MOC [Eker et al., 2003]. Si un acteur implémente toutes ces actions en accord avec la sémantique abstraite, il est dit polymorphique car il peut être dirigé par n'importe quel réalisateur (i.e. implémentant n'importe quel MOC). Cependant, les acteurs polymorphiques sont en fait rares car les MOC ne se conforment pas tous parfaitement à la sémantique abstraite des acteurs [Goderis et al., 2009]. La conséquence est qu'il n'est pas en fait possible de combiner n'importe comment les domaines pré-existants au sein d'une hiérarchie.

L'intérêt de Ptolemy II pour la multi-modélisation est qu'il permet d'aborder le problème de l'intégration des formalismes en les décrivant dans une syntaxe et une sémantique abstraites communes : celles des acteurs.

De plus, Ptolemy II autorise les acteurs atomiques à réutiliser des modèles déjà existants. Il est par exemple possible d'intégrer dans Ptolemy II des FMUs (voir Section 4.2.2) équationnelles dans une exécution événementielle [Muller and Widl, 2013]. Cependant, ces intégrations ne sont pas génériques et ne peuvent pas être combinées librement : par exemple les acteurs de FMU ne sont pas combinables avec un autre MOC que l'événementiel.

Enfin, Ptolemy II fournit une interface graphique permettant de générer directement le code du multi-modèle numérique à partir du modèle formel et donc de limiter les erreurs de traduction entre les multi-modèles formel et numérique [Eker et al., 2003].

Pour résumer, l'intéret de Ptolemy II pour la démarche de multi-modélisation et simulation est qu'il fournit un cadre de travail pour intégrer des modèles formels hétérogènes. Cependant, son point faible est que les solutions d'intégration développées manquent de flexibilité car elles ne permettent pas de combiner tous les formalismes. L'expérimentation sur le multi-modèle obtenu est alors limitée. De plus, Ptolemy II ne permet pas de gérer l'intégration de représentations hétérogènes dans un multi-modèle car il se situe uniquement à un niveau formel.

4.3.3 Le formalisme DEVS

Généralités

Discrete EVent System specification (DEVS) est un formalisme de modélisation créé par Bernard P. Zeigler dans les années 1970 [Zeigler et al., 2000]. S'il a au départ été conçu comme un formalisme événementiel, DEVS revêt en fait un caractère universel. En effet, outre le fait qu'il soit compatible avec les trois types de stratégies événementielles (voir Chapitre 2) [Zeigler et al., 2000], DEVS est également à la base d'un long travail d'intégration de formalismes. Ainsi, au fil des années, il a été montré que DEVS pouvait intégrer les formalismes des Réseaux de Petri [Jacques and Wainer, 2002], des statecharts [Borland and Vangheluwe, 2003], des automates à états finis temporels [Dacharry and Giambiasi, 2005], des automates cellulaires [Wainer and Giambiasi, 2001], et des équations différentielles [Zeigler et al., 2000, Vangheluwe, 2000, Quesnel et al., 2005]. De plus, plusieurs formalisations du paradigme multi-agent en DEVS ont été proposées [Duboz et al., 2006,Bisgambiglia and Franceschini, 2013,Uhrmacher and Schattenberg, 1998]. Enfin, il a été montré que DEVS pouvait également intégrer le super-formalisme DEV&DESS qui permet de décrire les interactions entre les formalismes discrets et continus [Zeigler, 2006].

Grâce à ces intégrations formelles, DEVS occupe une place centrale dans le graphe de transformation de formalismes (voir Figure 3.2) : il permet alors d'intégrer directement ou indirectement tous ces formalismes. Comme nous le montrons dans la suite, le formalisme est alors à même de décrire et de simuler un multi-modèle formel hétérogène [Vangheluwe, 2000].

Cette propriété est possible car DEVS est basé sur le même paradigme de modélisation qu'un multi-modèle : il décrit un système complexe comme un système de systèmes [Zeigler, 2013]. DEVS définit pour cela deux types de modèles : les modèles atomiques, et les modèles couplés que nous présentons dans les parties suivantes. Nous détaillons ensuite comment un modèle DEVS est simulé, et comment il permet de décrire un multi-modèle formel hétérogène.

Le modèle atomique DEVS

Le modèle atomique DEVS permet de décrire un système jusqu'au niveau de spécification 3 (voir Chapitre 2). Un modèle atomique DEVS va alors définir le comportement d'un système, c'est à dire qu'il va représenter (i) l'état du système ; (ii) comment celui-ci évolue de manière discrète dans le temps continu ; (iii) comment le système est affecté par des stimuli extérieurs (les entrées), et (iv) comment le système va générer des sorties. Un modèle atomique DEVS est muni d'une interface composée de ports d'entrée et de ports de sortie lui permettant de communiquer avec l'extérieur du système (i.e. de recevoir des stimuli extérieurs et d'envoyer les sorties générées). D'un point de vue extérieur, un modèle atomique DEVS peut être donc vu comme une boîte noire munie de ports d'entrée et de sortie.

DEVS distingue deux types d'événements : les événements internes et externes. Les premiers correspondent aux changements d'état planifiés et exécutés par la dynamique interne du modèle. Ils servent alors à décrire comment la dynamique interne du système fait évoluer ce dernier dans le temps. Les événements externes vont quant à eux passer par l'interface du modèle. Ils représentent alors les entrées et sorties du système. Les événements externes d'entrée vont arriver par les ports d'entrée du modèle et vont alors représenter des stimuli extérieurs affectant le fonctionnement du système. Les événements externes de sortie vont être émis par les ports de sortie du modèle, et vont représenter les sorties générées par le système. Lorsqu'un modèle reçoit un événement externe, il l'exécute immédiatement. Cette action va changer l'état du modèle et peut affecter sa dynamique interne (i.e. changer le temps du prochain événement). Les événements externes de sortie d'un modèle atomique sont générés par les exécutions de ses événements internes. Un

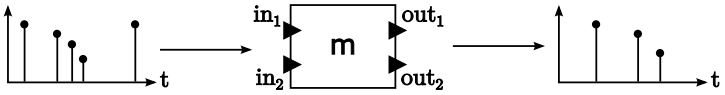


FIGURE 4.6 – Un exemple de modèle atomique DEVS et ses séquences d'événements externes d'entrée et de sortie.

modèle DEVS va alors recevoir en entrée et émettre en sortie des séquences d'événements externes (voir Figure 4.6).

Formellement, un modèle atomique DEVS est équivalent à la structure suivante [Zeigler et al., 2000] :

$$DEVS = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta) \tag{4.1}$$

où :

$X = \{(p, v)|p \in InPorts, v \in X\}$ est l'ensemble des ports d'entrée et leurs valeurs,

$Y = \{(p, v)|p \in OutPorts, v \in Y\}$ est l'ensemble des ports de sortie et leurs valeurs,

S correspond à l'ensemble des états successifs du modèle,

$\delta_{ext} : Q \times X \to S$ est la fonction de transition externe où

$Q = \{(s, e)|s \in S, 0 \leq e \leq ta(s)\}$ correspond à l'état total du modèle,

e est le temps écoulé depuis la dernière transition,

$\delta_{int} : S \to S$ est la fonction de transition interne,

$\lambda : S \to Y$ est la fonction de sortie,

$ta : S \to \mathbb{R}^{+}_{0,\infty}$ est la fonction d'avancement du temps.

Le fonctionnement du modèle atomique DEVS est le suivant (voir Figure 4.7). Chaque état $s \in S$ du modèle est associé à une durée de séjour par la fonction d'avancement du temps $ta(s)$. Cette durée de séjour correspond à la durée pendant laquelle le modèle va rester dans cet état en l'absence d'événement externe. On peut noter que cette durée de séjour peut être infinie. Dans ce cas, le modèle est dit passif, au sens où il va rester dans cet état tant qu'il ne recevra pas d'événement externe.

Lorsque la durée de séjour de l'état courant $s \in S$ du modèle est atteinte, un événement interne se produit : la fonction de sortie $\lambda(s)$ et la fonction de transition interne $\delta_{int}(s)$ sont appelées. $\lambda(s)$ va déterminer l'événement externe $u \in Y$ généré par l'événement interne, et $\delta_{int}(s)$ va déterminer le nouvel état $s' \in S$ du modèle.

Lorsque le modèle reçoit un événement externe $x \in X$, la fonction de transition externe $\delta_{ext}(s, e, x)$ est appelée. Cette fonction détermine le nouvel état $s' \in S$ du modèle en fonction de l'événement externe reçu, de l'état courant du modèle $s \in S$, et de la durée $e(0 \leq e \leq ta(s))$ pendant laquelle le système est resté dans cet état.

Le modèle couplé DEVS

Si le modèle atomique DEVS permet de décrire le comportement d'un système, le modèle couplé DEVS permet quant à lui de décrire la structure du système. Il permet donc de définir un système jusqu'au niveau de spécification 4 (voir Chapitre 2).

Un modèle couplé DEVS va décrire un système comme un ensemble de modèles atomiques interconnectés. Une connexion d'un modèle atomique m_1 vers un autre modèle m_2 va consister à relier un port de sortie *out* de m_1 à un port d'entrée *in* de m_2. Ainsi, tous les événements

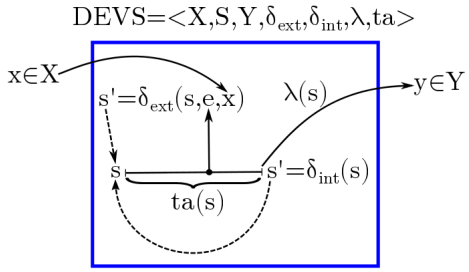


FIGURE 4.7 – Comportement d'un modèle DEVS. Inspirée de [Zeigler et al., 2000]

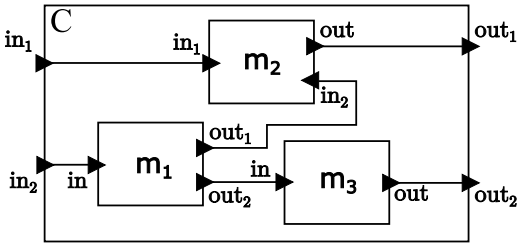


FIGURE 4.8 – Un exemple de modèle couplé DEVS

externes émis par m_1 à travers *out* seront immédiatement transmis à m_2 par *in*. Les connexions entre les modèles atomiques sont appelés **couplages internes**.

Comme un modèle atomique, un modèle couplé DEVS est doté d'une interface lui permettant de recevoir des événements externes par des ports d'entrée, et d'émettre des événements externes par des ports de sortie. Chaque port d'entrée du modèle couplé est connecté à un port d'entrée d'un de ses modèles atomiques. Ainsi, lorsqu'un événement externe est reçu par le port d'entrée d'un modèle couplé, il est immédiatement transmis au port d'entrée du modèle atomique correspondant. Les connexions entre les ports d'entrée d'un modèle couplé et ceux de ses modèles atomiques sont appelées **couplages externes d'entrée**.

De la même manière chaque port de sortie du modèle couplé est connecté à un port de sortie d'un modèle atomique. Ainsi, les ports de sortie d'un modèle couplé émettent immédiatement tous les événements externes sortant des ports de sortie des modèles atomiques auxquels il sont connectés. Les connexions entre les ports de sortie d'un modèle couplé et ceux de ses modèles atomiques sont appelées **couplages externes de sortie**.

Description	**Notation**
Composants	$D = \{m_1, m_2, m_3\}$
Interface	$InPorts = \{in_1, in_2\}$ $OutPorts = \{out_1, out_2\}$
Couplages	$IC = \{((m_1, out_1), (m_2, in_2)), ((m_1, out_2), (m_3, in))\}$ $EIC = \{((C, in_1), (m_2, in_1)), ((C, in_2), (m_1, in))\}$ $EOC = \{((m_2, out), (C, out_1)), ((m_3, out), (C, out_2))\}$

TABLE 4.1 – Formalisation du modèle couplé de la Figure 4.8

Formellement, un modèle couplé correspond à la structure [Zeigler et al., 2000](Figure 4.8 et Table 4.1) :

$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC) \tag{4.2}$$

où :

$X = \{(p, v) | p \in InPorts, v \in X\}$ est l'ensemble des ports d'entrée et leurs valeurs,

$Y = \{(p, v) | p \in OutPorts, v \in Y\}$ est l'ensemble des ports de sortie et leurs valeurs,

D est l'ensemble des identifiants des modèles atomiques,

pour chaque $d \in D$, M_d est un modèle atomique $M_d = (X_d, Y_d, S, \delta_{ext}, \delta_{int}, \lambda, ta)$ avec,

$X_d = \{(p, v) | p \in InPorts_d, v \in X_d\}$

$Y_d = \{(p, v) | p \in OutPorts_d, v \in Y_d\}$

$EIC = \{((N, ip_N), (d, ip_d)) | ip_N \in InPorts, d \in D, ip_d \in InPorts_d\}$ est l'ensemble des couplages externes d'entrée,

$EOC = \{((d, op_d), (N, op_N)) | op_N \in OutPorts, d \in D, op_d \in OutPorts_d\}$ est l'ensemble des couplages externes de sortie,

$IC = \{((a, op_a), (b, ip_b)) | a, b \in D, op_a \in OutPorts_a, ip_b \in InPorts_b\}$ est l'ensemble des couplages internes.

Fermeture par composition

Nous avons vu précédemment que le formalisme DEVS permet de décrire un système jusqu'au niveau 4 de spécification grâce aux concepts de modèle atomique et de modèle couplé. Une caractéristique importante de DEVS est que ces deux concepts sont prouvés comme étant formellement équivalents [Zeigler et al., 2000] : un modèle couplé est équivalent à un modèle atomique. Cette propriété est appelée la fermeture par composition.

La fermeture par composition de DEVS offre plusieurs avantages :

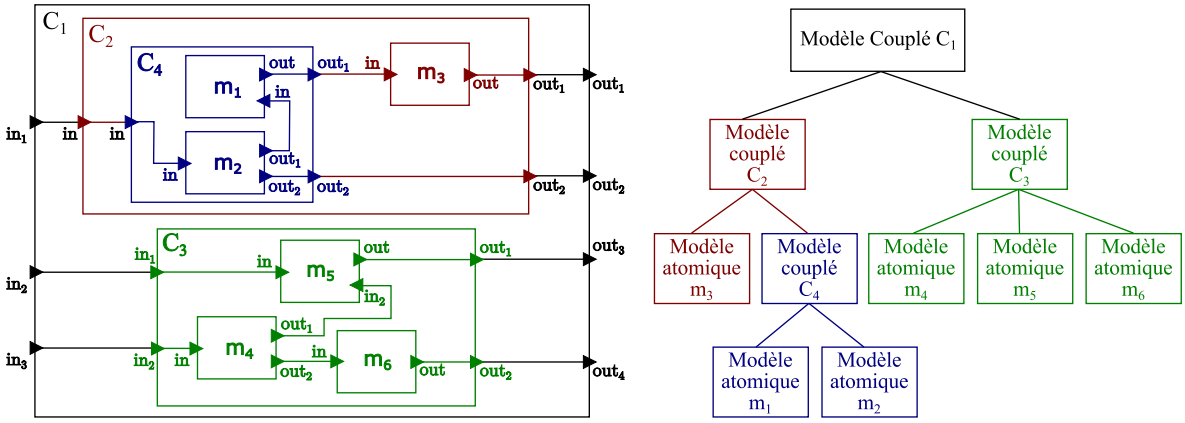
— Puisqu'un modèle couplé est équivalent à un modèle atomique, alors un modèle couplé peut ne pas être composé seulement de modèles atomiques, mais également de modèles couplés. **Il est alors possible, uniquement à partir des concepts de modèles atomiques et couplés, de décrire et simuler le système comme une hiérarchie à N niveaux** (Figure 4.9). On parlera alors de modèle DEVS hiérarchique. Cette description hiérarchique va alors permettre de diminuer la complexité représentative du système (voir Chapitre 1).

— La fermeture par composition implique qu'il existe pour chaque modèle atomique, un modèle couplé équivalent et vice versa. Autrement dit, le choix du modélisateur de décrire un système cible comme un modèle couplé ou comme un modèle atomique n'a en soit aucun impact sur la validité du modèle formel. **C'est alors uniquement la manière dont le modélisateur va décrire le système avec la structure choisie, c'est à dire ses choix de modélisation, qui va compter** [Duboz et al., 2012].

— **La fermeture par composition donne une sémantique au couplage entre les modèles atomiques** : coupler des modèles équivaut à décrire un modèle atomique DEVS.

— Comme les deux modèles sont équivalents, **il est possible d'établir un protocole de simulation commun aux deux types de modèles** : contrôler la simulation d'un modèle atomique ou d'un modèle couplé s'effectue de la même manière. Le modèle DEVS hiérarchique peut alors être simulé avec un protocole de simulation unifié.

Dans la suite nous détaillons comment un modèle DEVS hiérarchique est simulé.



(a) Modèle couplé (b) Vue hiérarchique

FIGURE 4.9 – Exemple de modélisation hiérachique DEVS

Simulation d'un modèle DEVS

La simulation d'un modèle DEVS (i.e. un modèle atomique ou couplé) repose sur deux variables tl et tn représentant respectivement le temps du dernier événement (externe ou interne) exécuté, et le temps du prochain événement interne planifié. A chaque fois que le modèle DEVS exécute un événement (interne ou externe) au temps t, alors $tl = t$ et tn est mis à jour.

Dans le cas d'un modèle atomique dans un état $s \in S$, alors $tn = tl + ta(s)$. Lorsque le modèle reçoit un événement externe au temps t, alors le temps écoulé depuis le dernier changement d'état est : $e = t - tl$.

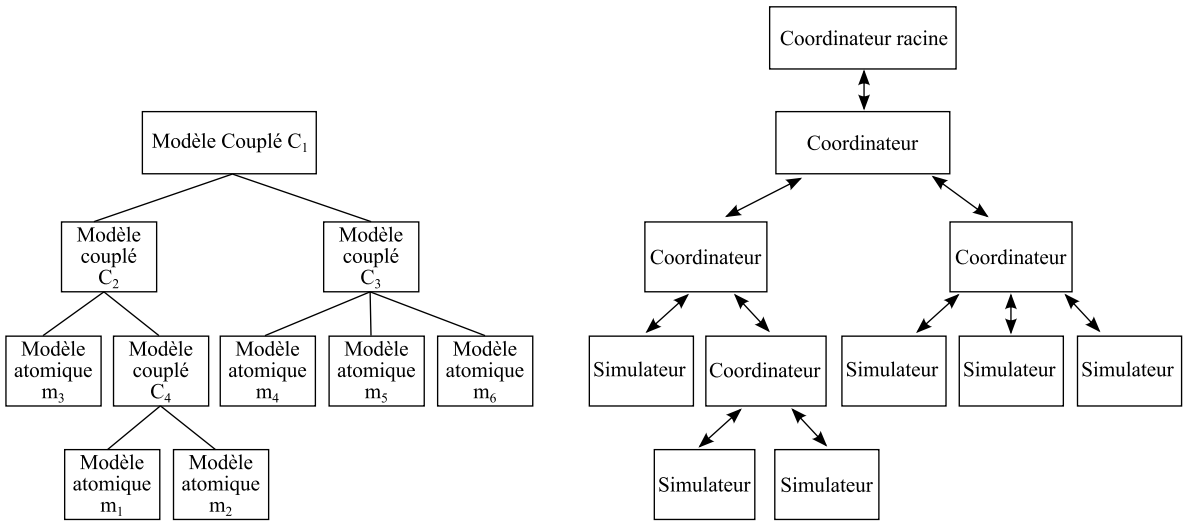
Dans le cas d'un modèle couplé DEVS, le calcul du prochain événement interne tn consiste à maintenir dans une liste ordonnée le tn_d de chacun de ses composants $d \in D$. Alors, $tn = min\{tn_d | d \in D\}$. La simulation d'un modèle couplé DEVS consiste alors à coordonner les exécutions de ses composants (i.e. des modèles atomiques et/ou des modèles couplés), et de propager les événements externes à l'intérieur de son modèle couplé. La coordination des composants est basée sur le principe de causalité événementielle (voir Définition 15). Respecter la contrainte de causalité lors d'une simulation parallèle d'un modèle couplé DEVS revient donc à respecter le principe de causalité événementielle.

Définition 15 *Le **principe de causalité événementielle** est respecté lorsque chaque modèle exécute ses événements (internes et externes) suivant un ordre temporel croissant [Fujimoto, 2001, Zeigler et al., 2000]*

L'architecture de simulation d'un modèle DEVS hiérarchique repose en général sur trois entités de simulation (Figure 4.10) :

- **Les simulateurs** sont associés aux modèles atomiques DEVS. Chaque simulateur a pour rôle d'exécuter un modèle atomique.
- **Les coordinateurs** sont associés aux modèles couplés DEVS. Chaque coordinateur a pour rôle de gérer l'exécution de son modèle couplé.
- **Le coordinateur racine** est associé au coordinateur du modèle couplé racine. Il sert à initialiser et coordonner la simulation de tous les modèles de la hiérarchie.

Une architecture de simulation d'un modèle DEVS correspond alors à une hiérarchie d'entités de simulation. Dans cette hiérarchie, les communications vont avoir lieu de manière locale : chaque entité de simulation va communiquer uniquement avec son parent et ses enfants directs.



(a) Exemple de hiérarchie de modèles DEVS (b) Architecture de simulation correspondante

FIGURE 4.10 – Architecture de simulation pour une hiérarchie DEVS

Ces communications sont alors effectuées en utilisant le protocole de simulation DEVS. Comme mentionné à la Section 4.3.3, ce protocole de simulation est identique pour les coordinateurs et les simulateurs.

Dans ce protocole, chaque entité de simulation va communiquer à son parent :

— Ses événements externes de sortie (y, t) émis au temps de simulation t.

— Le temps tn de son prochain événement interne.

A l'inverse, chaque entité de simulation va transmettre trois types de messages à ses enfants :

— L'ordre (i, t) d'initialiser leurs modèles au temps t.

— Les messages (x, t) signalant l'arrivée d'un événement externe d'entrée au temps t.

— L'ordre $(*, t)$ d'exécuter leurs événements internes au temps t.

Pour simuler un modèle DEVS hiérarchique, le formalisme propose un ensemble de simulateurs et coordinateurs abstraits séquentiels et parallèles [Zeigler et al., 2000]. La simulation d'un modèle DEVS hiérarchique peut également être distribuée sur plusieurs machines [Praehofer and Reisinger, 1994, Kim et al., 1996]

Intégration de formalismes avec DEVS

Comme nous l'avons vu précédemment, DEVS est doté d'un caractère universel. N'importe quel modèle formel peut alors être intégré sous la forme de modèle atomique DEVS. Un multi-modèle hétérogène peut alors être décrit formellement par un modèle couplé DEVS.

Dans [Quesnel et al., 2005], les auteurs différencient deux manières d'intégrer un modèle avec DEVS :

— **Le mapping** consiste à établir une équivalence entre le formalisme du modèle et DEVS. Cette technique permet de traduire un modèle formel en DEVS, mais limite la réutilisabilité du modèle car celui-ci doit être réécrit et réimplémenté en DEVS.

— **Le wrapping** consiste à faire le pont entre le comportement du simulateur abstrait du modèle et celui du simulateur abstrait DEVS. Cette technique a l'avantage de permettre de réutiliser des modèles exécutables existants qui peuvent alors être contrôlés avec le protocole de simulation DEVS [Kim and Kim, 1998].

Approche de modélisation et simulation basée DEVS

Le formalisme DEVS a donné lieu à de nombreuses implémentations allant de simples bibliothèques de simulation permettant d'implémenter un multi-modèle [Sarjoughian and Zeigler, 1998], à de véritables laboratoires virtuels permettant de spécifier un multi-modèle formel et de passer directement à sa simulation [Kim et al., 2009, Quesnel et al., 2009, Zeigler, 2013]. Ces dernières approches sont alors importantes pour la démarche de multi-modélisation et simulation car elles permettent d'automatiser le passage entre le modèle formel et le modèle numérique, et donc d'assurer la vérification du modèle.

De plus, dans DEVS-HLA [Zeigler et al., 1998], la technique du wrapping DEVS a été utilisée conjointement avec un intergiciel de simulation basé sur la norme HLA (voir Section 4.2.1). Cette approche rend alors possible la réutilisation de modèles hétérogènes déjà implémentés et validés en gérant l'intégration de leurs modèles formels et l'interopérabilité de leurs outils de simulation [Zeigler et al., 1999].

Limite de DEVS pour la multi-modélisation

Comme nous l'avons vu dans les sections précédentes, les avantages de DEVS sont nombreux pour la multi-modélisation : il permet de définir les multi-modèles formel, exécutable et numérique, en intégrant des formalismes et des modèles exécutables hétérogènes. De plus, l'approche peut être combinée avec un intergiciel de simulation afin de réutiliser des modèles déjà implémentés.

Cependant, DEVS comporte une lacune pour la multi-modélisation : il ne fournit pas de solution explicite pour faire le pont entre des modèles conceptuels hétérogènes [Seck and Honig, 2012].

4.4 Solutions d'intégration des représentations hétérogènes

Dans cette section nous détaillons les approches permettant d'assurer la cohérence des différentes représentations du système cible dans un multi-modèle. La plateforme MIMOSA propose d'homogénéiser le multi-modèle conceptuel dès sa conception en unifiant, grâce aux ontologies, les modèles conceptuels des différents thématiciens (Section 4.4.1). A l'opposé, l'approche AA4MM permet, grâce à son paradigme Agents & Artifacts, d'effectuer des traductions entre différents modèles conceptuels déjà implémentés dans des logiciels de simulation (Section 4.4.2).

4.4.1 L'approche MIMOSA

MIMOSA [Müller, 2007] est une plateforme de M&S des systèmes complexes permettant la création successive des modèles conceptuel, formel, exécutable et numérique.

La plateforme est structurée en deux composants :

— **Le composant déclaratif** va permettre aux thématiciens de représenter le(s) modèle(s) conceptuel(s) du système sous la forme d'une ontologie.

— **Le composant exécutif** permet de définir les modèles formel, exécutable et numérique en se basant sur le formalisme DEVS (détaillé précédemment en Section 4.3.3).

La méthodologie de MIMOSA est la suivante [Müller, 2010].

Tout d'abord, les thématiciens vont spécifier dans le composant déclaratif le modèle conceptuel, c'est à dire les concepts de base qui composent le système et leurs relations (e.g. les concepts

de voiture et de piéton). Il faut noter ici que MIMOSA étend la notion d'ontologie en permettant de spécifier une dynamique aux concepts représentés.

A ce niveau, MIMOSA va permettre à différents thématiciens de travailler ensemble pour articuler leurs différentes représentations du système [Müller and Diallo, 2012]. MIMOSA établit alors une méthodologie pour coupler les ontologies de différents thématiciens. Le modèle conceptuel devient alors un ensemble articulé de points de vue [Müller and Aubert, 2011]. La méthode consiste à établir une ontologie par thématicien, puis à tenter de fusionner les concepts en commun par des liens de subsumation ou d'équivalence. Lorsque les concepts ont des sémantiques incompatibles, il faut alors chercher un compromis dans leurs définitions afin de pouvoir les lier.

Une fois le modèle conceptuel établi, MIMOSA va permettre de créer le modèle concret, c'est à dire de définir l'état initial du système en instanciant les différents concepts du modèle conceptuel (e.g. créer cent voitures et dix piétons).

Enfin, MIMOSA va déduire un modèle couplé DEVS du modèle concret de la manière suivante : chaque instance d'un concept du modèle conceptuel va correspondre à un modèle atomique DEVS. Le modèle couplé ainsi obtenu pourra être directement exécuté grâce aux algorithmes de simulation de DEVS.

MIMOSA présente un intérêt évident pour la démarche de multi-modélisation et simulation, au sens où il permet, grâce aux ontologies, d'intégrer des représentations hétérogènes d'un même système complexe. De plus, comme il se repose sur le formalisme DEVS, MIMOSA permet alors d'intégrer des formalismes hétérogènes. Enfin, MIMOSA systématise le passage d'une étape de la démarche à une autre et limite ainsi les erreurs pouvant intervenir. Cependant, MIMOSA est incompatible avec la réutilisation de modèles car sa méthodologie présuppose que les thématiciens unifient leurs représentations avant que leurs modèles soient formalisés et implémentés. L'approche ne supporte donc pas la gestion de l'interopérabilité des outils de simulation.

4.4.2 L'approche Agents & Artifacts for Multi-Modeling

Généralité

L'approche Agents & Artifacts for Multi-Modeling (AA4MM) [Siebert, 2011] a pour but de construire et simuler le multi-modèle hétérogène d'un système complexe à partir d'un ensemble de modèles pré-existants déjà implémentés dans des logiciels de simulation. AA4MM repose sur le paradigme multi-agent Agents & Artifacts (voir Section 2.6.3) pour représenter un multi-modèle : le multi-modèle correspond à un ensemble d'agents autonomes interagissant à travers un ensemble d'artefacts. La simulation du multi-modèle correspond alors à la dynamique du système multi-agent. L'originalité de l'approche est de considérer les interactions entre les modèles de manière indirecte à travers un ensemble d'artefacts.

AA4MM a été développé dans le cadre de l'étude des réseaux mobiles ad-hoc [Leclerc et al., 2010]. Ces réseaux sont par exemple composés d'un ensemble de smartphones servant également d'antennes relais. Ces smartphones vont être mobiles au gré des déplacements de leurs utilisateurs : des piétons évoluant dans un espace urbain.

Les concepts de AA4MM sont associés à des spécifications opérationnelles permettant d'effectuer un couplage structurel de modèles [Siebert et al., 2010b]. Ce couplage structurel (voir Figure 4.11) considère l'évolution d'un nombre constant d'entités (e.g. les piétons/smartphones) simultanément impliquées dans plusieurs dynamiques (e.g. le déplacement des piétons et l'évolution de la connectivité du réseau), chaque dynamique étant représentée par un modèle différent (e.g. un modèle des piétons et un modèle du réseau).

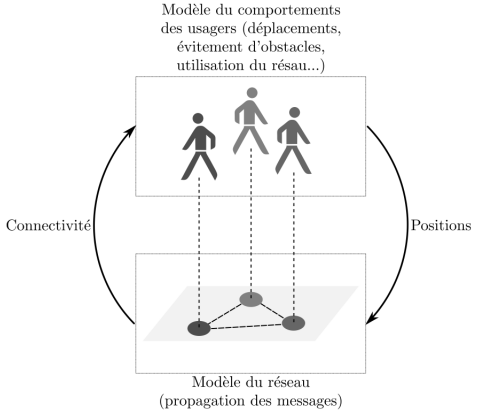


FIGURE 4.11 – Couplage structurel de modèles dans le cadre de l'étude des réseaux mobiles ad-hoc. Source [Siebert, 2011]

(a) modèle (b) m-agent (c) artefact de couplage (d) artefact de modèle

FIGURE 4.12 – Symboles des concepts de AA4MM

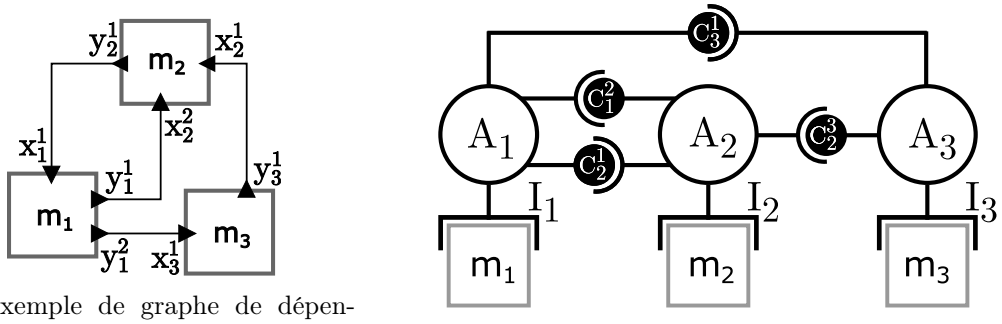
Représentation d'un multi-modèle avec AA4MM

Pour décrire un multi-modèle, AA4MM repose sur quatre concepts, chacun d'eux étant associé à une notation graphique (voir Figure 4.13) :

- Un **modèle** m_i (symbole à la Figure 4.12a) déjà implémenté dans un logiciel de simulation. AA4MM considère que ce modèle est doté d'ensembles X_i de ports d'entrée et Y_i de ports de sortie
- Un **m-agent** $\mathcal{A}_i$ (symbole à la Figure 4.12b) est associé à chaque modèle m_i. $\mathcal{A}_i$ a pour rôle de gérer l'exécution de m_i, et les interactions de celui-ci avec les autres modèles.
- Chaque interaction (i.e. échange de données de simulation) d'un modèle m_i vers un modèle m_j est instanciée par un **artefact de couplage** $\mathcal{C}_j^i$ (symbole à la Figure 4.12c) gérant les interactions de $\mathcal{A}_i$ vers $\mathcal{A}_j$. Un artefact de couplage $\mathcal{C}_j^i$ est orienté : il gère les interactions de $\mathcal{A}_i$ vers $\mathcal{A}_j$. $\mathcal{C}_j^i$ est alors pour $\mathcal{A}_i$ un artefact de couplage de sortie, et un artefact de couplage d'entrée pour $\mathcal{A}_j$.
- Enfin, chaque **artefact de modèle** $\mathcal{I}_i$ instancie les interactions entre un m-agent $\mathcal{A}_i$ et son modèle m_i (symbole à la Figure 4.12d).

En accord avec le paradigme multi-agent sur lequel il est basé (voir Section 2.6.3), chaque m-agent n'a qu'une connaissance locale du graphe de dépendance du multi-modèle. Cette caractéristique permet à AA4MM d'avoir une représentation modulaire et décentralisée d'un multi-modèle. AA4MM représente le graphe de dépendance de manière éclatée (voir Table 4.2) de façon à ce que chaque m-agent $\mathcal{A}_i$ sache uniquement :

- A quel artefact de couplage de sortie il doit envoyer des données de simulation produites par chaque port de sortie de m_i. $\mathcal{A}_i$ connaît alors l'ensemble des liens d'entrée IN_i com-



(a) Exemple de graphe de dépendance

(b) Multi-modèle avec AA4MM

FIGURE 4.13 – Construction d'un multi-modèle avec AA4MM.

prenant les couples (j, k) reliant l'artefact de couplage d'entrée $\mathcal{C}_i^j$ avec le port d'entrée $x_i^k \in X_i$.

— Auprès de quel artefact de couplage d'entrée il doit récupérer les données de simulation pour chacun des ports d'entrée de m_i. $\mathcal{A}_i$ connaît alors l'ensemble des liens de sortie OUT_i comprenant les couples (n, j) reliant le port de sortie $y_i^n \in Y_i$ avec l'artefact de couplage de sortie $\mathcal{C}_j^i$.

Les connexions entre les ports de sortie d'un modèle m_i et les ports d'entrée d'un modèle m_j sont effectuées par l'artefact de couplage $\mathcal{C}_j^i$. Cet artefact contient l'ensemble L_j^i contenant les couples (n, k) reliant le port de sortie $y_i^n \in Y_i$ avec le port d'entrée $x_j^k \in X_j$.

Descriptions	**Notations**
Liens de sortie de m_1	$\mathsf{OUT}_1 = \{(1,2),(2,3)\}$
Liens d'entrée de m_1	$\mathsf{IN}_1 = \{(2,1)\}$
Liens de sortie de m_2	$\mathsf{OUT}_2 = \{(1,1)\}$
Liens d'entrée de m_2	$\mathsf{IN}_2 = \{(1,2),(3,1)\}$
Liens de sortie de m_3	$\mathsf{OUT}_3 = \{(1,2)\}$
Liens d'entrée de m_3	$\mathsf{IN}_3 = \{(1,1)\}$
Liens de m_1 à m_2	$\mathsf{L}_2^1 = \{(1,2)\}$
Liens de m_1 à m_3	$\mathsf{L}_3^1 = \{(2,1)\}$
Liens de m_2 à m_1	$\mathsf{L}_1^2 = \{(1,1)\}$
Liens de m_3 à m_2	$\mathsf{L}_2^3 = \{(1,1)\}$

TABLE 4.2 – Formalisation du graphe de dépendance du multi-modèle de la figure 4.13b avec AA4MM.

Spécifications opérationnelles

Chacun des concepts de AA4MM est associé à des spécifications opérationnelles permettant d'implémenter un intergiciel de co-simulation. Ces spécifications permettent d'implémenter et de simuler un multi-modèle en ayant uniquement un ensemble restreint de fonctions spécifiques à l'application à définir.

Un artefact de couplage $\mathcal{C}_j^i$ fonctionne comme une boîte aux lettres. L'artefact a un tampon de données dans lequel $\mathcal{A}_i$ peut déposer les données de simulation produites par m_i. $\mathcal{A}_j$ peut ensuite récupérer ces données afin de paramétrer m_j pendant la simulation. $\mathcal{C}_j^i$ peut également appliquer

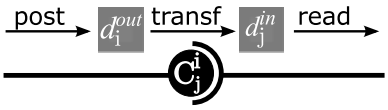


FIGURE 4.14 – Fonctionnement d'un artefact de couplage AA4MM

des opérations de transformation qui vont modifier les données échangées. Ces opérations permettent de traduire les communications entre des modèles ayant des représentations différentes du système cible (e.g. passer de coordonnées en kilomètres à des coordonnées en mètres) [Siebert et al., 2010a]. Les artefacts de couplage permettent ainsi de gérer l'intégration de modèles représentant le système à différentes échelles temporelles et spatiales [Siebert et al., 2010a], et à différents niveaux de résolution [Camus et al., 2015b, Camus et al., 2013].

$\mathcal{C}_j^i$ propose trois fonctions à $\mathcal{A}_i$ et $\mathcal{A}_j$ (voir Figure 4.14) :

— **post**($d_{out_i}(t_i)$) peut être utilisé par $\mathcal{A}_i$ pour déposer une donnée de simulation $d_{out_i}(t_i)$ produite par m_i au temps t_i.
— **transf**(d_{out_i}) transforme la donnée de simulation d_{out_i} en donnée transformée $d^*_{out_i}$.
— **read**(t_j) peut être utilisé par $\mathcal{A}_j$ pour récupérer des données de simulation pour le temps de simulation t_j.

Un artefact de modèle $\mathcal{I}_i$ fonctionne comme une interface logicielle pour le simulateur de m_i, c'est à dire qu'il contient les primitives logicielles permettant de manipuler ce simulateur. Ces primitives correspondent à :

— **init()** démarre le logiciel de simulation, et instancie l'état initial de m_i.
— **setInputData**(x_i, d_{out_j}) insère la donnée de simulation d_{out_j} dans le port d'entrée $x_i \in X_i$.
— **getOutputData**(y_i) récupère la donnée de simulation présente dans le port de sortie $y_i \in Y_i$.
— **run**(qt) exécute le modèle pour la durée qt.
— **getCurrentTime()** retourne le temps courrant de m_i.
— **stop()** termine la simulation (e.g. arrêt du logiciel).

Pour synchroniser l'exécution de ces modèles, AA4MM est basé sur les hypothèses suivantes :

— les modèles ont une politique d'exécution par pas de temps (cycliques ou non),
— chaque modèle a besoin avant toute exécution, d'une et une seule donnée valide pour chacun de ses liens d'entrée IN_i.

Basé sur ces hypothèses, le comportement des m-agents (Figure 4.15) correspond à un algorithme de simulation parallèle prouvé comme respectant la contrainte de causalité [Siebert, 2011, Siebert et al., 2009]. Le comportement d'un m-agent correspond au cycle :

— **Lire** des données de simulation depuis ses artefacts de couplages d'entrée. Cette lecture est bloquante : $\mathcal{A}_i$ attend tant qu'il n'a pas reçu de données. Une fois les donnée reçues, celles-ci sont utilisées pour paramétrer m_i via $\mathcal{I}_i$.
— **Exécuter** m_i pour un pas de simulation via $\mathcal{I}_i$.
— **Poster** les données de simulation produites par m_i dans ses artefacts de couplage de sortie (ou une donnée *null* si l'exécution du modèle n'a pas produit de données).

La synchronisation des exécutions des m-agents s'effectue grâce aux artefacts de couplage : lorsqu'un m-agent $\mathcal{A}_j$ fait appel à la méthode $read(t_j)$ d'un de ses artefacts de couplage d'entrée $\mathcal{C}_j^i$, ce dernier ne lui fournit que des données valides (i.e. des données pouvant être utilisées par $\mathcal{A}_j$ sans violer la contrainte de causalité) pour le temps t_j. $\mathcal{A}_j$ est alors obligé d'attendre des données valides de $\mathcal{C}_j^i$ pour continuer son exécution.

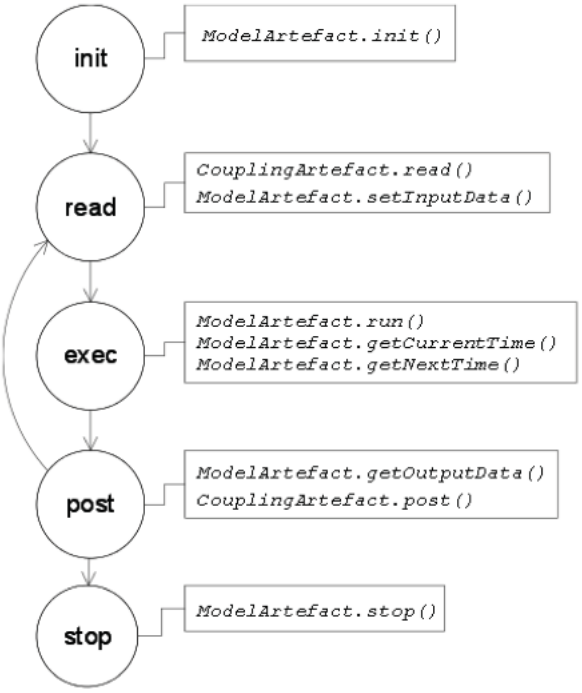


FIGURE 4.15 – Algorithme de simulation de AA4MM. Source [Siebert et al., 2010a]

Afin que les artefacts de couplage puissent déterminer si une donnée est valide ou non, tous les m-agents associent chaque donnée de simulation produite à un intervalle de validité. Plus précisément, une donnée de simulation produite par l'exécution d'un modèle passant du temps de simulation t à $t+qt$ est associé à l'intervalle $]t, t+qt]$. Lorsqu'un m-agent demande une donnée pour un modèle ayant un temps courant de simulation t, un artefact de couplage considère alors comme valide toute donnée associée à un intervalle de validité Γ avec $t \in \Gamma$. Ce dispositif permet de garantir le respect de la contrainte de causalité lors de la simulation parallèle du multi-modèle.

Pour implémenter un multi-modèle avec AA4MM, l'informaticien dispose des implémentations pré-existantes des concepts de m-agents et d'artefacts de couplage. L'informaticien a alors uniquement à définir un artefact de modèle pour chaque logiciel de simulation utilisé dans le muti-modèle, ainsi que les opérations de transformation permettant de faire le pont entre les différentes représentations du système.

Avantages et limites de AA4MM pour la multi-modélisation

L'avantage de AA4MM pour la multi-modélisation est qu'il permet, grâce au concept d'artefact de couplage, de décrire les interactions entre les modèles comme des processus. Il est alors possible de décrire les processus d'intégration de l'hétérogénéité de différentes représentations du système (voir Chapitre 3) [Siebert et al., 2010a]. De plus, grâce au concept d'artefact de modèle, AA4MM permet de gérer l'interopérabilité des logiciels de simulation avec l'intergiciel de simulation. Ensuite, grâce au paradigme A&A, AA4MM propose une vision modulaire et décentralisée d'un multi-modèle qui favorise l'expérimentation dans le cadre de la démarche de multi-modélisation et simulation [Siebert et al., 2010b].

Cependant, AA4MM comporte également plusieurs limites pour la multi-modélisation. Tout d'abord, l'approche est limitée par son domaine d'application, celui de l'étude des réseaux mo-

	HLA	FMI	AToM^3	Ptolemy	DEVS	MIMOSA	AA4MM
Représentation	non	non	non	non	non	oui	oui
Formalisme	non	non	oui	oui	oui	oui	non
Simulateur	oui	oui	non	non	non	non	oui
Systématisation	non	non	oui	oui	oui	oui	non

TABLE 4.3 – Comparaison de la capacité des solutions de multi-modélisation et simulation existantes à remplir les pré-requis de la démarche de multi-modélisation et simulation. oui signifie que l'approche fournit une solution, alors que oui signifie une solution incomplète. non signifie une absence de solution, alors que non signifie que l'approche est incompatible avec le pré-requis.

biles ad-hoc : AA4MM considère uniquement des modèles couplés structurellement ayant une représentation discrète du temps. Les spécifications opérationnelles de l'approche, c'est à dire son algorithme et son protocole de simulation parallèle, ne sont donc adaptés que dans ce cadre. **L'approche est alors limitée dans sa capacité d'intégration de formalismes hétérogènes et ne peut notamment intégrer des formalismes événementiels.**

Ensuite, le passage d'un multi-modèle formel à un multi-modèle numérique n'est pas systématique dans AA4MM : une fois le multi-modèle formalisé, celui-ci doit être implémenté manuellement. Comme nous l'avons vu au Chapitre 2, cette phase peut alors comporter des erreurs qui vont compromettre la vérification du modèle dans le cadre d'une démarche de M&S.

4.5 Synthèse

Dans ce chapitre, nous avons détaillé plusieurs approches de M&S. La Table 4.3 synthétise la capacité de ces approches à répondre aux pré-requis de la démarche de M&S des systèmes complexes (voir Chapitre 3). De cette synthèse, nous pouvons tirer plusieurs conclusions.

Tout d'abord, nous pouvons constater qu'**aucune approche n'apporte de solution à tous les niveaux**. Nous pouvons toutefois voir que la plupart des approches n'ont pas d'incompatibilité forte avec les pré-requis qu'elles ne couvrent pas. Ces approches peuvent alors sûrement être étendues pour pouvoir couvrir tous les défis de la démarche de M&S.

Il faut noter ensuite que l'approche AA4MM est très intéressante pour nous, car elle a la particularité d'être la seule parmi celles étudiées à apporter explicitement des solutions pour l'intégration de représentations hétérogènes, tout en gérant l'interopérabilité de différents logiciels de simulation. L'approche ne permet toutefois pas l'intégration de formalismes hétérogènes au sein d'un multi-modèle.

Nous pouvons alors constater que **AA4MM et DEVS apparaissent comme complémentaires**, car DEVS apporte la solution d'intégration formelle rigoureuse qui fait défaut à AA4MM. Les deux approches recouvrent donc ensemble potentiellement tous les pré-requis portant sur l'hétérogénéité du multi-modèle.

Aucune approche ne répondant simultanément à tous les défis de la démarche de M&S des systèmes complexes, nous pourrions faire le choix dans cette thèse de définir entièrement une nouvelle approche. Cependant, compte tenu de l'apport scientifique considérable que constituent DEVS et AA4MM, nous pensons qu'il est important de se servir ici de ces travaux. **Nous faisons alors le choix dans cette thèse de combiner AA4MM et DEVS pour répondre aux défis de la démarche de M&S des systèmes complexes.** Nous détaillons ce point dans le Chapitre suivant.

Chapitre 5

Positionnement

Sommaire

5.1 Introduction

L'objectif de ce chapitre est double : (1) détailler plus précisément notre problématique de départ en synthétisant notre travail bibliographique des chapitres précédents, et (2) poser les hypothèses et contraintes sur lesquelles se base notre contribution.

Rappelons notre questionnement de départ : dans le cadre de la démarche de M&S, comment décrire sans ambiguïté et simuler rigoureusement le modèle d'un système complexe ?

A partir de ce que nous avons vu dans les Chapitres 2 et 3, nous faisons l'hypothèse que le modèle d'un système complexe correspond à un multi-modèle : un ensemble hétérogène de modèles en interaction. Nous faisons alors les hypothèses suivantes sur les caractéristiques d'un multi-modèle :

— Il est composé de modèles qui peuvent avoir des représentations hétérogènes du système cible, et qui vont donc décrire ce système à différentes échelles (temporelles et spatiales) et différents niveaux de résolution.
— Il est composé de modèles qui peuvent être écrits dans des formalismes hétérogènes, et qui décrivent donc les changements d'état du système, le temps ou l'espace de manières différentes (i.e. discrète ou continue).
— Il est composé de modèles implémentés dans des logiciels de simulation hétérogènes qui vont donc être écrits dans des langages de programmation différents, et fonctionner sur des systèmes d'exploitation différents.
— Le nombre de modèles en interaction qui le compose peut être très grand.

Nous posons également deux contraintes sur les propriétés qu'un multi-modèle doit avoir pour être utilisable dans la démarche de M&S :

— Comme la démarche de M&S est un processus d'expérimentation qui va progresser par une suite d'essais et d'erreurs, nous souhaitons un multi-modèle qui soit **modulaire** pour être rapidement et facilement modifié (i.e. ajout, suppression, remplacement de modèle ou d'interaction entre les modèles).

— Comme le multi-modèle d'un système complexe peut être composé d'un grand nombre de modèles, nous souhaitons que notre solution gère le passage à l'échelle. Comme nous l'avons vu au Chapitre 3, pour satisfaire cette contrainte, il faut permettre la **simulation parallèle** d'un multi-modèle.

5.2 Problématique

Pour décrire et simuler un multi-modèle en restant dans le cadre scientifique rigoureux de la démarche de M&S, les pré-requis suivants doivent alors être remplis :

1. Permettre l'intégration rigoureuse de l'hétérogénéité d'un multi-modèle au niveau des représentations, des formalismes et des logiciels de simulation.
2. Assurer la vérification de l'implémentation d'un multi-modèle : soit a posteriori, soit a priori en assurant le passage automatique de la description d'un multi-modèle à sa simulation.
3. Gérer l'exécution parallèle d'un multi-modèle en respectant la contrainte de causalité.

Pour répondre à ces pré-requis, la démarche de M&S des systèmes complexes doit donc se doter d'une approche apportant des solutions à la fois aux niveaux des modèles numérique, exécutable, formel et conceptuel.

Dans le Chapitre 4, nous avons vu qu'aucune des approches existantes de M&S ne permettait de répondre simultanément à tous ces pré-requis. Nous avons cependant observé que DEVS et AA4MM utilisés conjointement permettent théoriquement de répondre aux pré-requis 1 et 3 tout en étant compatibles avec nos contraintes. Nous en avons conclu, compte tenu de l'apport conséquent de ces deux approches, qu'il était scientifiquement intéressant de les combiner.

Deux questions sont alors soulevées : AA4MM et DEVS sont-elles combinables ? Si oui, comment ?

Compte tenu de ce que nous avons vu dans le Chapitre précédent, nous pensons que AA4MM et DEVS sont compatibles et donc combinables, car leurs apports ne sont pas situés au même niveau d'abstraction :

— C'est grâce à son paradigme A&A que AA4MM permet la description et la simulation parallèle de manière modulaire et décentralisée d'un multi-modèle composé de représentations et de logiciels de simulation hétérogènes.

— C'est grâce à son formalisme et ses spécifications opérationnelles que DEVS permet la description et la simulation parallèle d'un multi-modèle composé de formalismes hétérogènes.

Pour répondre aux pré-requis 1 et 3, nous avons alors besoin d'une approche qui associe aux concepts A&A du paradigme de AA4MM, les spécifications opérationnelles du formalisme DEVS.

La question qui reste encore en suspens est alors : Comment répondre au pré-requis 2 ? Pour y répondre, nous pensons qu'il est plus intéressant d'assurer le passage automatique de la description d'un multi-modèle à sa simulation, que de vérifier systématiquement à posteriori l'implémentation d'un multi-modèle. En effet, systématiser l'implémentation d'un multi-modèle permet de s'affranchir de toute manipulation de code. Cette solution permet alors d'ouvrir la démarche de M&S des systèmes complexes aux non-informaticiens, notamment aux thématiciens et modélisateurs issus d'autres domaines scientifiques.

Pour répondre au pré-requis 2 nous avons alors besoin d'une approche qui permette de décrire numériquement et de manière non ambiguë un multi-modèle, puis

par une suite de transformations systématiques, de passer de la description d'un multi-modèle à un programme exécutable. Comme décrit au Chapitre 2, nous avons alors besoin d'emprunter une démarche d'Ingénierie Dirigée par les Modèles (IDM).

5.3 Propositions

Notre contribution consiste à proposer une approche d'IDM constituée :

— Des spécifications opérationnelles de l'intergiciel de co-simulation Multi-agent Environment for Complex-SYstem CO-simulation (MECSYCO) permettant de simuler un multi-modèle hétérogène en combinant les concepts multi-agents de AA4MM avec le formalisme DEVS. En implémentant ces spécifications dans le langage Java, nous avons produit l'intergiciel de co-simulation MECSYCO-Java.

— Du méta-modèle Méta-AA4MM formalisant le langage de multi-modélisation de AA4MM, et permettant de décrire de manière rigoureuse des multi-modèles hétérogènes. En implémentant ce méta-modèle, nous avons produit un environnement graphique de multi-modélisation pour AA4MM. Nous appellerons des multi-modèles décrits avec Méta-AA4MM des multi-modèles AA4MM.

— Des règles de transformation permettant de passer de manière systématique d'un multi-modèle AA4MM à un programme implémentant les spécifications de MECSYCO. En implémentant ces règles en langage XSLT, nous avons produit un processus permettant de passer d'un multi-modèle AA4MM à une simulation MECSYCO-Java de manière automatique.

Nous affirmons que cette approche offre les solutions suffisantes pour décrire un multi-modèle sans ambiguïté et le simuler rigoureusement dans le cadre de la démarche de M&S des systèmes complexes. Nous développons dans les chapitres 6 et 7 cette approche. Par la suite, nous évaluons dans les Chapitres 8 et 9 ses avantages et ses limites à travers un ensemble de preuves de concept, et un cas d'application pratique.

Chapitre 6

Spécification de l'intergiciel MECSYCO

Sommaire

6.1 Introduction

Dans ce chapitre, nous proposons les spécifications opérationnelles de l'intergiciel de co-simulation MECSYCO (Multi-agent Environment for Complex System CO-simulation) qui a pour vocation de fournir une base opérationnelle pour simuler le modèle d'un système complexe dans le cadre rigoureux de la démarche de M&S. Comme nous l'avons vu au chapitre précédent, pour remplir ce rôle, MECSYCO doit permettre d'intégrer des représentations, des formalismes et des simulateurs différents au sein d'un multi-modèle, puis de simuler ce dernier de manière parallèle. En suivant nos choix du chapitre précédent, l'intergiciel repose pour ce faire sur une combinaison de DEVS et AA4MM.

Nous avons vu dans les chapitres précédents que, dans le cadre de la démarche de M&S des systèmes complexes, **la force de AA4MM est au niveau de son paradigme A&A, alors que ses limites sont au niveau de ses spécifications opérationnelles.**

Nous avons également vu que le formalisme DEVS apparaît comme complémentaire de l'approche AA4MM. En effet, ce formalisme a un caractère universel qui implique que n'importe quel modèle formel peut être décrit sous la forme d'un modèle atomique DEVS. **Un multi-modèle formel peut alors être homogénéisé sous la forme d'un modèle couplé DEVS**.

Nous proposons ici d'exploiter l'universalité de DEVS dans AA4MM, en nous en servant comme d'un formalisme pivot pour gérer l'intégration de formalismes hétérogènes. La démarche que nous adoptons consiste alors à (1) wrapper chaque modèle formel sous la forme d'un modèle atomique DEVS, et ensuite (2) simuler le multi-modèle de façon parallèle dans AA4MM comme un modèle couplé DEVS.

Les problèmes soulevés par cette proposition sont alors : Comment wrapper un modèle sous la forme d'un modèle atomique DEVS avec les concepts de AA4MM ? Cette étape est-elle suffisante pour considérer un multi-modèle comme un modèle couplé DEVS dans AA4MM ? Quel est l'impact sur les interactions entre les m-agents dans AA4MM ? Comment simuler en parallèle un modèle atomique DEVS avec les concepts de AA4MM ?

Ces questions nous amènent à définir de nouvelles spécifications opérationnelles pour les concepts Agent A&A de AA4MM en nous basant sur le formalisme DEVS. Pour conserver les propriétés de AA4MM tout en ajoutant la capacité d'intégration formelle de DEVS, ces extensions se font dans le respect du paradigme A&A de AA4MM. Ces spécifications incluent (1) un nouvel algorithme de simulation parallèle (i.e. un nouveau comportement des m-agents), et (2) un nouveau protocole de simulation (i.e. de nouvelles fonctions et opérations pour les artefacts).

Plus précisément, notre contribution consiste à étendre les spécifications originelles de AA4MM comme suit :

- **les artefacts de modèle** servent de wrappeur DEVS et non plus uniquement d'interfaces logicielles. Nous définissons alors de nouvelles primitives logicielles pour les artefacts de modèles. Nous montrons alors que, grâce à cette modification, le multi-modèle peut être simulé comme un modèle couplé DEVS (Section 6.3).
- **les artefacts de couplage** permettent l'échange d'événements externes entre les m-agents, et non plus uniquement de données de simulation. Nous définissons alors un nouveau fonctionnement (Section 6.4) et de nouvelles opérations de transformation (Section 6.6) pour les artefacts de couplage.
- **les m-agents** simulent un modèle couplé DEVS et non plus uniquement un multi-modèle à pas de temps discrets. Nous définissons alors un nouvel algorithme de simulation parallèle pour le comportement des m-agents. Cet algorithme inclut une procédure d'initialisation du multi-modèle (Section 6.5).

6.2 Notions préliminaires

Pour définir MECSYCO, nous nous appuyons sur les notions de AA4MM suivantes établies par [Siebert, 2011] :

- Chaque modèle m_i du multi-modèle est doté d'un ensemble de ports d'entrée $X_i = \{x_i^1, x_i^2, ..., x_i^p\}$, et d'un ensemble de ports de sortie $Y_i = \{y_i^1, y_i^2, ..., y_i^q\}$.
- Chaque m-agent $\mathcal{A}_i$ connaît l'ensemble OUT_i des liens de sortie de m_i. Chaque lien de sortie de m_i correspond à un couple (n, j) associant un port de sortie y_i^n avec un artefact de couplage de sortie $\mathcal{C}_j^i$.
- Chaque m-agent $\mathcal{A}_i$ connaît l'ensemble IN_i des liens d'entrée de m_i. Chaque lien d'entrée de m_i correspond à un couple (j, k) associant un artefact de couplage d'entrée $\mathcal{C}_i^j$ avec le port d'entrée x_i^k.
- Chaque artefact de couplage $\mathcal{C}_j^i$ connaît l'ensemble L_j^i des liens de m_i vers m_j. Chaque lien de m_i vers m_j correspond à un couple (n, k) associant le port de sortie y_i^n au port d'entrée x_j^k.

Niveau	Description	Notation
Modèle	M-agent	$\mathcal{A}_i$
	Artefact de modèle	$\mathcal{I}_i$
	Artefact de couplage de $\mathcal{A}_i$ vers $\mathcal{A}_j$	$\mathcal{C}_j^i$
	Modèle	m_i
	Ensemble des ports d'entrée de m_i	X_i
	Ensemble des ports de sortie de m_i	Y_i
	k^{eme} port d'entrée de m_i	$x_i^k (0 < k \leq card(X_i))$
	n^{eme} port de sortie de m_i	$y_i^n (0 < n \leq card(Y_i))$
	Les opérations temporelles de m_i vers m_j	TO_j^i
	Les opérations événementielles de y_i^n vers x_j^k	$O_{j,k}^{i,n}$
	Ensemble des liens d'entrée de m_i	IN_i
	Ensemble des liens de sortie de m_i	OUT_i
	Ensemble des liens de m_i vers m_j	L_j^i
Simulation	Temps maximum de simulation	$Z \in \mathbb{R}_{0,\infty}^+$
	Temps de simulation de m_i	$t_i \in \mathbb{R}_{0,\infty}^+$
	Date du prochain événement interne de m_i	$nt_i \in \mathbb{R}_{0,\infty}^+$
	Earliest estimated input time de m_i	$\mathsf{EIT}_i \in \mathbb{R}$
	Earliest estimated output time de m_i	$\mathsf{EOT}_i \in \mathbb{R}$
	Link time de $\mathcal{C}_j^i$	$\mathsf{LT}_j^i \in \mathbb{R}$
	Temps de l'événement externe le plus imminent m_i	$t_{in_i} \in \mathbb{R}$
	Événement externe le plus imminent de m_i	ein_i
	Événement externe de sortie du port y_i^n	$eout_i^n$

Table 6.1 – Notations de MECSYCO

Toute les notations que nous définissons dans ce chapitre sont résumées par la Table 6.1.

6.3 Représentation d'un modèle couplé DEVS dans MECSYCO

Pour permettre, dans un multi-modèle AA4MM, la traduction de chaque modèle sous la forme d'un modèle atomique DEVS, nous étendons le concept d'artefact de modèle pour qu'il ne serve plus uniquement d'interface logicielle pour les simulateurs, mais également de wrappeur DEVS (voir Section 4.3.3). Ainsi, les artefacts de modèle ne gèrent plus uniquement l'interopérabilité de leur logiciel de simulation avec AA4MM, mais également l'intégration dans DEVS des modèles formels auxquels ils sont associés. **L'utilisation d'un wrappeur nous permet d'intégrer dans DEVS des modèles déjà implémentés, et donc de favoriser la réutilisation de modèles.**

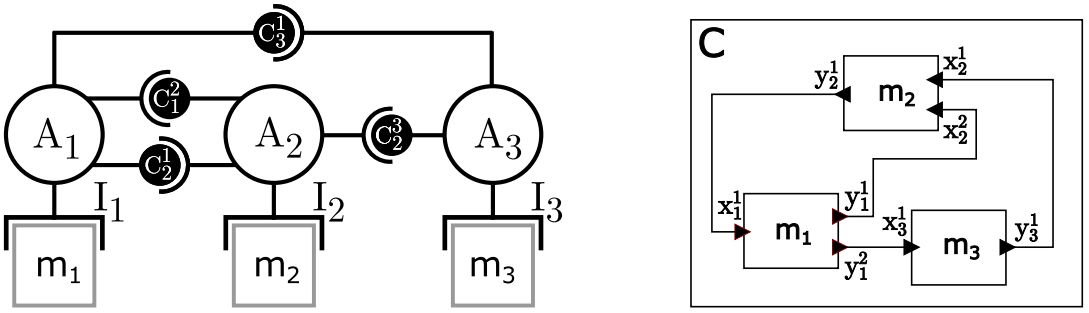
En accord avec le principe de wrappeur, un artefact de modèle $\mathcal{I}_i$ va émuler le comportement d'un simulateur DEVS à partir du comportement du simulateur abstrait de m_i. Grâce à leurs artefacts de modèle, les m-agents pourront alors contrôler leurs modèles comme des modèles DEVS, en se servant du protocole de simulation DEVS (voir Section 4.3.3).

En conséquence, un artefact de modèle $\mathcal{I}_i$ contient maintenant les primitives logicielles suivantes :

— $init()$ initialise le modèle m_i, c'est à dire démarre le logiciel de simulation du modèle, fixe les paramètres de ce dernier et son état initial.

— $processExternalEvent(ein_i, t_i, x_i^k)$ exécute l'événement externe d'entrée ein_i arrivant au temps de simulation t_i dans x_i^k, le k^{eme} port d'entrée de m_i.

— $processInternalEvent(t_i)$ exécute l'événement interne du modèle m_i planifié au temps



(a) Exemple de multi-modèle AA4MM (b) Modèle couplé DEVS équivalent.

FIGURE 6.1 – Equivalence formelle entre un multi-modèle AA4MM et un modèle couplé DEVS

de simulation t_i.

— $getOutputEvent(y_k^i)$ retourne $eout_i^k$, l'événement externe de sortie présent dans le k^{eme} port de sortie de m_i, y_k^i (ou *null* si aucun événement de sortie n'est présent dans ce port).

— $getNextInternalEventTime()$ retourne le temps du prochain événement interne m_i.

Grâce à ces nouvelles primitives, les artefacts de modèle permettent de considérer un multi-modèle AA4MM comme un modèle couplé DEVS. Ceci est possible car, comme nous le montrons ci-dessous, il y a une équivalence formelle entre le graphe de dépendance AA4MM et les couplages internes du modèle couplé DEVS : le premier est une version décentralisée du second.

En effet, chaque couplage interne $((m_a, y_a^i), (m_b, x_b^j)) \in IC$ du modèle couplé DEVS faisant le lien entre le port de sortie y_a^i du composant m_a et le port d'entrée x_b^j du composant m_b, correspond à un artefact de couplage $\mathcal{C}_b^a$ ainsi que les couples $(i, b) \in \mathsf{OUT}_a$, $(a, j) \in \mathsf{IN}_b$, et $(i, j) \in \mathsf{L}_b^a$. Afin d'illustrer notre propos, nous montrons dans la Figure 6.1 et la Table 6.2 un exemple d'équivalence formelle entre un graphe de dépendance AA4MM et les couplages internes d'un modèle couplé DEVS.

Couplages internes DEVS	**Liens entre les modèles AA4MM**
$((m_1, y_1^1), (m_2, x_2^2)) \in IC$	$(1,2) \in \mathsf{OUT}_1$ $(1,2) \in \mathsf{IN}_2$ $(1,2) \in \mathsf{L}_2^1$
$((m_1, y_1^2), (m_3, x_3^1)) \in IC$	$(2,3) \in \mathsf{OUT}_1$ $(1,1) \in \mathsf{IN}_3$ $(2,1) \in \mathsf{L}_3^1$
$((m_2, y_2^1), (m_1, x_1^1)) \in IC$	$(1,1) \in \mathsf{OUT}_2$ $(2,1) \in \mathsf{IN}_1$ $(1,1) \in \mathsf{L}_1^2$
$((m_3, y_3^1), (m_2, x_2^1)) \in IC$	$(1,2) \in \mathsf{OUT}_3$ $(3,1) \in \mathsf{IN}_2$ $(1,1) \in \mathsf{L}_2^3$

TABLE 6.2 – Équivalence entre les couplages internes du modèle couplé DEVS et le graphe de dépendance du multi-modèle AA4MM de la Figure 6.1

6.4 Echange d'événements avec MECSYCO

Un multi-modèle AA4MM correspondant maintenant à un modèle couplé DEVS, les interactions entre modèles vont correspondre à des échanges d'événements externes (Définition 16). **Nous changeons alors les spécifications originelles de AA4MM en associant chaque donnée de simulation avec une estampille temporelle, et non plus avec un intervalle de validité** (comme expliqué précédemment en Section 4.4.2).

Définition 16 *Un **événement externe** correspond à une donnée de simulation associée à une estampille temporelle. Cette estampille temporelle indique la date (dans le temps simulé) d'émission de l'événement.*

Comme le multi-modèle est simulé en parallèle dans AA4MM, les échanges d'événements vont avoir lieu de manière asynchrone. **Le problème est alors qu'il faut permettre le stockage temporaire des événements externes échangés entre les simulateurs**.

La solution adoptée pour les simulations parallèles DEVS consiste à équiper les simulateurs parallèles de tampon à chaque port d'entrée. Ces tampons, qui correspondent à des files FIFO (premier entré, premier sorti), permettent de stocker les événements externes attendant d'être exécutés.

Dans AA4MM, nous situons ces tampons au niveau des artefacts de couplage car ceux-ci sont les supports des interactions entre les modèles. **Nous définissons alors de nouvelles spécifications opérationnelles pour les artefacts de couplage afin que ceux-ci (1) contiennent une file FIFO à événements, et (2) proposent aux m-agents des méthodes permettant de manipuler cette file**.

Nous définissons alors les opérations suivantes pour que les m-agents puissent s'échanger des événements externes en utilisant les artefacts de couplage. Chaque artefact de couplage $\mathcal{C}_j^i$ propose l'opération $post(e_{out}^k)$ au m-agent $\mathcal{A}_i$. Cette opération stocke dans le tampon, l'événement externe e_{out}^k provenant du port de sortie y_i^k. $\mathcal{C}_j^i$ propose également trois opérations à $\mathcal{A}_j$:

- $getEarliestEvent(k)$ retourne l'événement externe le plus imminent à destination du k^{eme} port d'entrée de m_j, x_j^k (ou *null* s'il n'y a pas d'événement externe).
- $getEarliestEventTime(k)$ retourne l'estampille temporelle de l'événement externe le plus imminent à destination du k^{eme} port d'entrée de m_j, x_j^k (ou $+\infty$ s'il n'y a pas d'événement externe).
- $removeEarliestEvent(k)$ supprime du tampon l'événement externe le plus imminent à destination du k^{eme} port d'entrée de m_j, x_j^k.

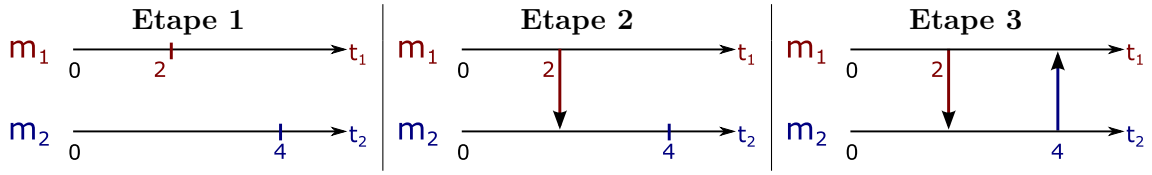
6.5 Simulation parallèle d'un multi-modèle avec MECSYCO

6.5.1 Positionnement

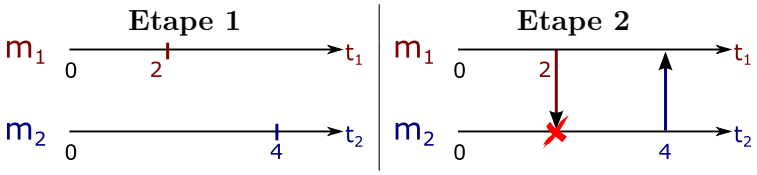
Dans cette section, nous détaillons comment les m-agents vont effectuer la simulation parallèle du multi-modèle comme un modèle couplé DEVS dans MECSYCO. Comme nous l'avons dit en Section 3.5, toute simulation parallèle doit s'assurer de respecter la contrainte de causalité. En effet, comme montré par la Figure 6.2, si les modèles atomiques d'un modèle couplé sont exécutés en parallèle sans coordination, il y a un risque de violer le principe de causalité événementielle (voir Definition 15).

Deux types d'approches existent pour résoudre ce problème [Perumalla, 2007] :

- **Les approches conservatives** consistent à s'assurer que la contrainte de causalité ne sera jamais violée pendant la simulation.



(a) Exécution DEVS classique respectant la contrainte de causalité.



(b) Exécution parallèle violant la contrainte de causalité. A l'étape 2, les modèles m_1 et m_2 exécutent en parallèle leurs événements internes respectivement au temps 2 et 4. m_2 reçoit alors au temps 2 l'événement externe de m_1 alors qu'il a déjà calculé son événement interne au temps 4.

FIGURE 6.2 – Exemple de violation de la contrainte de causalité lors de l'exécution parallèle d'un multi-modèle composé de deux modèles m_1 et m_2 s'envoyant mutuellement des événements externes. Les prochains événements internes de chaque modèle sont disposés sur les axes horizontaux représentant le temps simulé. Chaque flèche verticale indique l'envoi d'un événement externe.

— **Les approches optimistes** consistent à exécuter les modèles le plus rapidement possible, tout en vérifiant a posteriori si la contrainte de causalité a été violée. Lorsque c'est le cas, il faut retourner en arrière dans la simulation au moment où la contrainte de causalité a été violée. Ce retour arrière est appelé *roll-back*.

Les approches optimistes requièrent que tous les simulateurs du multi-modèle aient une capacité de *roll-back* leur permettant de se réinitialiser à un état passé. Cette capacité peut être implémentée de deux manières différentes. La première consiste à sauvegarder l'état du modèle à chaque étape de la simulation. Pour revenir à un instant donné, il suffit alors de charger l'état correspondant dans le modèle. La deuxième consiste à définir la dynamique interne inverse du modèle (i.e permettant de faire régresser le modèle), et à générer des anti-événements externes (i.e. permettant d'annuler les effets des événements externes envoyés initialement).

Utiliser une approche optimiste dans MECSYCO permettrait uniquement d'intégrer des simulateurs ayant une capacité de *roll-back*. Comme cette caractéristique est peu commune dans les outils de simulation existants, **nous optons ici pour une approche de simulation conservative**.

Dans une approche conservative DEVS, chaque simulateur doit s'assurer avant d'exécuter n'importe quel événement (interne ou externe) de respecter la contrainte de causalité. Il faut également s'assurer que la simulation conservative ne va pas se retrouver en situation d'attente infinie (ou *deadlock*). Cette situation peut arriver s'il est impossible à un moment de la simulation, de déterminer quels sont les simulateurs qui peuvent s'exécuter sans violer la contrainte de causalité.

Nous proposons alors l'algorithme de simulation parallèle conservatif de MECSYCO. Pour ce faire, nous adaptons aux concepts A&A de AA4MM l'algorithme conservatif de DEVS [Zeigler et al., 2000] lui-même inspiré de l'algorithme de Chandy-Misra-Bryant (CMB) [Chandy and Misra, 1979, Bryant, 1979]. Sur ce dernier point, l'algorithme CMB a l'avantage de permettre une exécution complètement décentralisée d'un

multi-modèle et est donc compatible avec le paradigme de AA4MM. Nous rappelons que la simulation du multi-modèle dans AA4MM est effectuée par les m-agents. **L'algorithme de simulation parallèle que nous proposons ici correspond alors au comportement des m-agents.** Les preuves que toute simulation parallèle effectuée avec l'algorithme CMB respectera la contrainte de causalité et n'aura pas de deadlock peuvent être trouvées dans [Zeigler et al., 2000, p. 269-270].

Dans la suite nous détaillons le fonctionnement de l'algorithme. La Section 6.5.2 pose le principe général de l'algorithme. La Section 6.5.3 détaille le fonctionnement de l'algorithme. Nous montrons ensuite en Section 6.5.4 l'algorithme du comportement des m-agents. Enfin nous présentons en Section 6.5.5 comment le multi-modèle est initialisé, et comment les m-agents détectent le moment où ils atteignent la fin de la simulation.

6.5.2 Principe général

La coordination des m-agents avec l'algorithme CMB est basée sur le principe suivant : chaque m-agent va déterminer son lookahead c'est à dire la durée dans le futur (dans le temps simulé) pendant laquelle il est sûr que son modèle ne générera pas d'événements externes de sortie. Cette prévision est basée à la fois sur le lookahead des autres m-agents et sur l'état de son modèle (i.e. chaque m-agent tient le raisonnement : *« si je sais que personne ne m'enverra d'événement avant une date D, alors je peux être sûr, compte tenu de l'état de mon modèle, que je n'enverrai pas moi même d'événement avant une date D' »*). En se basant sur le lookahead des autres m-agents, chaque m-agent va être en mesure de savoir quand il pourra exécuter son modèle en étant sûr de respecter la contrainte de causalité.

La mise à jour du lookahead d'un m-agent a lieu (1) lorsque l'état de son modèle change, et (2) lorsqu'un autre m-agent met lui-même à jour son lookahead. Les prévisions des m-agents vont donc s'alimenter les unes les autres, et les m-agents exécuteront leurs modèles dès qu'ils le pourront.

Le fonctionnement de l'algorithme de simulation parallèle étant complexe, la section suivante qui le détaille est relativement aride et technique. Il est néanmoins nécessaire de détailler l'algorithme précisément pour permettre son implémentation non-ambiguë et ainsi assurer la reproductibilité de nos travaux de thèse. C'est pourquoi nous décrivons dans la suite, le fonctionnement de l'algorithme de manière formelle et rigoureuse.

6.5.3 Fonctionnement

Coordination des m-agents via les artefacts de couplage

En accord avec le paradigme A&A de AA4MM, les m-agents vont coordonner leurs tâches (i.e. l'exécution de leurs modèles) via leurs artefacts de couplage. **Le principe fondamental sur lequel repose cette coordination dans MECSYCO est celui du *Link Time*** (voir Définition 17).

Définition 17 *Le Link Time* LT_j^i *(*$\mathsf{LT}_j^i \in \mathbb{R}$*) correspond à la date (dans le temps simulé) jusqu'à laquelle tous les liens* L_j^i *(d'un modèle* m_i *vers un modèles* m_j*) ont été simulés (i.e. la date jusqu'à laquelle tous les événements externes devant transiter sur les liens* L_j^i *ont été envoyés).*

Chaque LT_j^i est maintenu dans un artefact de couplage $\mathcal{C}_j^i$. **Nous étendons alors les artefacts de couplage afin qu'ils soient dotés non plus uniquement d'un tampon d'événements externes mais également d'un *Link Time*.**

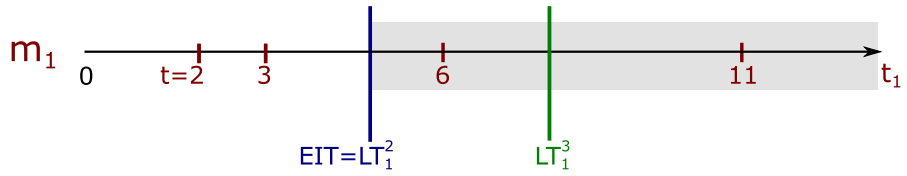


FIGURE 6.3 – Exemple de calcul de l'EIT pour un modèle m_1 recevant les événements externes de deux modèles m_2 et m_3. m_1 se base sur une pile dont les événements internes sont disposés sur un axe horizontal représentant le temps simulé. Les *Link Times* des artefacts de couplage d'entrée de $\mathcal{A}_1$ sont positionnés sur l'axe. Les événements en dehors de la zone grise correspondent aux événements sécurisés.

La coordination des m-agents avec les *Link Times* se fait en deux temps. D'un coté, chaque m-agent a la responsabilité de mettre à jour les *Link Times* de ses artefacts de couplage de sortie. De l'autre, les m-agents vont se baser sur les *Link Times* de leurs artefacts de couplage d'entrée pour coordonner leurs exécutions. L'artefact propose alors deux opérations supplémentaires permettant aux m-agents de manipuler LT^i_j :

— $setLinkTime(t)$ permet de fixer la valeur de LT^i_j au temps de simulation t.

— $getLinkTime()$ retourne la valeur de LT^i_j.

Dans les sections suivantes nous détaillons :

1. Comment les m-agents utilisent le *Link Time* de leurs artefacts de couplage d'entrée pour exécuter leurs modèles en respectant la contrainte de causalité.
2. Comment les m-agents calculent le *Link Time* de leurs artefacts de couplage de sortie.

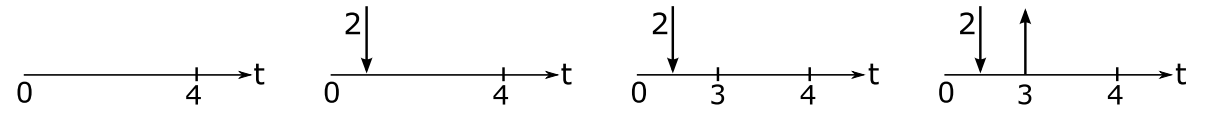
Utilisation des Link Times pour respecter la contrainte de causalité

Grâce aux *Link Times* de ses artefacts de couplage d'entrée, un m-agent $\mathcal{A}_i$ va pouvoir calculer son Estimated Earliest Input Time (EIT_i) (voir Définition 18). En effet, on déduit de la Définition 17 qu'un artefact de couplage $\mathcal{C}^i_j$ ne recevra pas d'événement externe ayant d'estampille temporelle inférieure à LT^i_j. On déduit alors de la Définition 18 que EIT_i est égal au minimum des *Link Times* de tous les artefacts de couplage d'entrée de $\mathcal{A}_i$ (voir Figure 6.3), autrement dit :

$$\mathsf{EIT}_i = \min_{(j,k)\in \mathsf{IN}_i} \{\mathsf{LT}^j_i\} \tag{6.1}$$

Définition 18 *L'*EIT_i *d'un m-agent* $\mathcal{A}_i$ *correspond à la date (dans le temps simulé) en dessous de laquelle* $\mathcal{A}_i$ *est sûr qu'aucun nouvel événement externe ne sera déposé dans ses artefacts de couplage d'entrée. (*$\mathsf{EIT}_i \in \mathbb{R}$*)*

Comme nous le montrons ci-dessous, EIT_i permet de déterminer la date jusqu'à laquelle $\mathcal{A}_i$ peut simuler m_i en étant sûr de respecter la contrainte de causalité. En effet, afin de satisfaire la contrainte de causalité, avant d'exécuter dans son modèle n'importe quel événement (interne ou externe) ayant une estampille temporelle t, un m-agent doit être sûr qu'il ne recevra jamais aucun événement externe d'entrée ayant une estampille temporelle inférieure à t. Si cette condition est remplie, l'événement à exécuter est dit sécurisé. On déduit alors de la Définition 18 que **tous les événements (internes et externes) ayant une estampille temporelle inférieure ou égale à** EIT_i **sont sécurisés, et peuvent donc être exécutés suivant un ordre temporel croissant.**



(a) Etat initial de la pile

(b) Réception d'un événement externe au temps 2

(c) Planification résultante d'un événement interne au temps 3

(d) Exécution de l'événement interne au temps 3 et génération de l'événement externe resultant.

FIGURE 6.4 – Exemple de causalité indirecte d'un événement externe de sortie par un événement externe d'entrée. Le modèle fonctionne ici avec une pile d'événements. Les prochains événements internes planifiés sont disposés sur l'axe horizontal représentant le temps simulé. Les flèches verticales indiquent l'envoi ou la réception d'un événement externe.

Le calcul des Link Times grâce au principe du lookahead

Pour un m-agent $\mathcal{A}_i$, déterminer les *Link Times* de ses artefacts couplage de sortie revient à calculer son Estimated Earliest Output Time (EOT) (voir Définition 19), autrement dit :

$$\forall j, \mathsf{T}_j^i = \mathsf{EOT}_i \tag{6.2}$$

Définition 19 *L'*EOT_i *d'un m-agent* $\mathcal{A}_i$ *correspond à la date (dans le temps simulé) en dessous de laquelle* $\mathcal{A}_i$ *garantit qu'il n'enverra pas d'événements externes de sortie. (*$\mathsf{EOT_i} \in \mathbb{R}$*)*

Pour calculer EOT_i, $\mathcal{A}_i$ va devoir prévoir le comportement de son modèle m_i dans le temps simulé. Plus précisément $\mathcal{A}_i$ va devoir prédire quand m_i va générer un nouvel événement externe de sortie, autrement dit, quand m_i va effectuer sa prochaine transition interne. Le problème est que m_i peut éventuellement replanifier ses événements internes s'il reçoit des événements externes d'entrée. **Les événements externes d'entrée peuvent alors causer (de manière indirecte) des événements externes de sortie** (voir Figure 6.4). En conséquence, ni la date nt_i du prochain événement interne planifié, ni la date lt_i du dernier événement exécuté ne fournissent d'information fiable pour prédire le comportement de m_i.

Pour résoudre ce problème, **l'algorithme CMB se repose sur le principe du lookahead qui consiste à utiliser le délai minimum de propagation** Dt_i **de chaque modèle** m_i **(voir Définition 20)**. Dt_i est fixe tout au long de la simulation et doit être déterminé pour chaque modèle m_i du multi-modèle. Dans le cas particulier des modèles où les événements externes d'entrée n'ont pas d'impact sur la planification des événements internes $Dt_i = +\infty$.

Définition 20 *Le* ***délai minimum de propagation*** Dt_i *d'un modèle* m_i *correspond au délai minimum possible dans* m_i *entre l'arrivée d'un événement externe d'entrée et la génération de l'événement externe de sortie résultant (dans le temps simulé).*

A n'importe quelle étape de la simulation de m_1, le prochain événement externe qui sera généré par le modèle peut être causé (directement ou indirectement) par (1) son prochain événement interne planifié, (2) un des événements externes en attente dans les artefacts de couplage d'entrée de $\mathcal{A}_i$, ou (3) un événement externe non encore déposé dans les artefacts de couplage d'entrée de $\mathcal{A}_i$. Grâce à Dt_i, $\mathcal{A}_i$ est capable de déterminer dans chacun de ces trois cas de figure, la date (dans le temps simulé) en dessous de laquelle il est sûr que le prochain événement externe ne sera pas généré :

— Le prochain événement interne de m_i planifié au temps nt_i ($nt_i \in \mathbb{R}$) causera un événement externe à nt_i.
— Il découle de la définition de Dt_i, que les événements externes qui sont en attente dans les artefacts de couplage d'entrée de $\mathcal{A}_i$ ne peuvent pas causer d'événement externe de sortie avant $t_{in_i} + Dt_i$, t_{in_i} correspondant à l'estampille temporelle du plus imminent de ces événements externes ($t_{in_i} \in \mathbb{R}$).
— Il découle de la définition de EIT_i que les événements externes d'entrée non encore déposés dans les artefacts de couplage d'entrée de $\mathcal{A}_i$ ne pourront avoir d'estampille temporelle inférieure à EIT_i. Il découle alors de la définition de Dt_i que ces événements externes ne pourront pas causer d'événement externe de sortie avant $EIT_i + Dt_i$.

En combinant ces trois prévisions, $\mathcal{A}_i$ est alors en mesure de déterminer EOT_i. En effet, $\mathcal{A}_i$ peut être sûr que m_i ne produira pas d'événement externe de sortie avant la plus imminente des trois dates nt_i, $t_{in_i} + Dt_i$ et $EIT_i + Dt_i$. Nous définissons alors la fonction *Lookahead* permettant de calculer EOT_i :

$$\mathsf{EOT}_i = Lookahead_i(nt_i, EIT_i, t_{in_i}) = min(nt_i, EIT_i + Dt_i, t_{in_i} + Dt_i) \tag{6.3}$$

6.5.4 Le comportement des m-agents

En se basant sur le principe de fonction énoncé dans les sections précédentes, nous proposons l'algorithme 1 correspondant au comportement des m-agents dans MECSYCO. Cet algorithme permet aux m-agents d'exécuter leurs modèles dans le respect de la contrainte de causalité. Les opérations proposées respectivement par les artefacts de modèles et les artefacts de couplage sont présentées dans les Tables 6.3 et 6.4. Pour un m-agent $\mathcal{A}_i$, l'algorithme peut être résumé par le cycle suivant :

1. Lire le *Link Times* de chacun de ces artefacts de couplage d'entrée et calculer EIT_i.
2. Récupérer le temps nt_i du prochain événement interne ($nt_i \in \mathbb{R}$).
3. Récupérer l'estampille temporelle t_{in_i} de ein_i, l'événement externe le plus imminent parmi tous les événements externes en attente dans les artefacts de couplage d'entrée ($t_{in_i} \in \mathbb{R}$).
4. Calculer EOT_i et mettre à jour le *Link Time* des artefacts de couplage de sortie.
5. Exécuter les événements sécurisés dans un ordre temporel croissant, c'est à dire :
   (a) Déterminer quel est l'événement le plus imminent entre le prochain événement interne et ein_i.
   (b) Déterminer si cet événement est sécurisé.
   (c) Si oui, calculer cet événement.
   (d) Si cet événement était un événement interne sécurisé alors récupérer et propager les événements externes de sortie résultants.

Afin d'illustrer le fonctionnement global de l'algorithme, nous présentons avec la Figure 6.5 et la Table 6.6 un exemple de déroulement d'une simulation parallèle du multi-modèle de la figure 6.1a. Dans cet exemple, on considère que les modèles sont dotés d'une pile d'événements. tn_i correspond alors à la date du premier événement de la pile du modèle m_i. Pour des raisons de simplicité, nous considérons dans cet exemple que les événements externes n'ont pas replanifié d'événements internes dans les modèles. Chaque étape correspond à une photographie à un instant t (dans le temps réel) de l'état de la simulation. On considère ici, pour des raisons de simplicité, que ces photographies ont été prises après que les événements externes envoyés ont été reçus par les m-agents et intégrés dans les modèles. Les valeurs des paramètres de cette simulation sont détaillées dans la Table 6.5.

Opérations	**Description**
$init()$	Initialise le logiciel de simulation et fixe les paramètres de m_i
$getNextInternalEventTime()$	Retourne la date du prochain événement interne de m_i
$getOutputEvent(y_i^n)$	Retourne l'événement externe présent dans le port de sortie y_i^n (ou *null* s'il n'y a pas d'événement externe)
$processExternalEvent(ein_i, t, x_i^k)$	Exécute l'événement externe ein_i avec l'estampille temporelle t arrivant dans le port d'entrée x_i^k
$processInternalEvent(t)$	Exécute le prochain événement interne planifié au temps t

TABLE 6.3 – Opérations proposées par un artefact de modèle $\mathcal{I}_i$

Opérations	**Description**
$post(eout_i^n, t_i)$	Envoie l'événement externe $eout_i^n$ et son estampille temporelle t_i
$getEarliestEvent(k)$	Retourne l'événement externe à destination du port x_j^k le plus imminent (ou *null* s'il n'y a pas d'événement externe)
$getEarliestEventTime(k)$	Retourne l'estampille temporelle de l'événement externe à destination du port x_j^k le plus imminent (ou $+\infty$ s'il n'y a pas d'événement externe)
$removeEarliestEvent(k)$	Supprime du tampon l'événement externe le plus imminent à destination du port x_j^k
$setLinkTime(t_i)$	Fixe LT_j^i à t_i
$getLinkTime()$	Retourne LT_j^i

TABLE 6.4 – Operations proposées par un artefact de couplage $\mathcal{C}_j^i$

Paramètres	**Valeurs**
Dt_1	3
Dt_2	2
Dt_3	1
Z	12

TABLE 6.5 – Paramètres de la simulation de la Figure 6.5.

Etapes	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
EIT_1	0	2	3	4	6.5	6.5	9	10	12.5	13	EOS	EOS
EIT_2	0	1	3	6	6	7	9.5	10.5	12	12	16,5	EOS
EIT_3	0	2	5	6	6	9.5	9.5	11	11	15.5	EOS	EOS
EOT_1	2*	5	6*	6*	9.5	9.5	11*	11*	15.5	16.5	EOS	EOS
EOT_2	2	3	4*	6.5*	6.5*	9	10*	12.5	13	14	14	EOS
EOT_3	1*	3*	6	7	7	10.5	10.5	12	12	16.5	EOS	EOS

TABLE 6.6 – Valeurs des EOTs et EITs des m-agents aux différentes étapes de la simulation distribuée de la Figure 6.5. Les astérisques indiquent que $\mathsf{EOT}_i = tn_i$. Dans les autres cas $\mathsf{EOT}_i = \mathsf{EIT}_i + Dt_i$.

Algorithme 1 Comportement d'un m-agent $\mathcal{A}_i$.

ENTREE : IN_i, OUT_i, Dt_i
SORTIE :

$nt_i \leftarrow \mathcal{I}_i.getNextEventTime()$
$t_{in_i} \leftarrow +\infty$
$\mathsf{EOT}_i \leftarrow 0$
$\mathsf{EIT}_i \leftarrow 0$
▷ Tant que la fin de la simulation n'est pas atteinte (voir Section 6.5.5)
while $(\neg endOfSimulation)$ **do**
  $\mathsf{EIT}_i \leftarrow +\infty$
  $t_{in_i} \leftarrow +\infty$
  for all $(j,k) \in \mathsf{IN}_i$ **do**
    if $\mathcal{C}_i^j.getLinkTime() < \mathsf{EIT}_i$ **then** ▷ Calcul de EIT_i
      $\mathsf{EIT}_i \leftarrow \mathcal{C}_i^j.getLinkTime()$
    end if
    if $\mathcal{C}_i^j.getEarliestEventTime(k) < t_{in_i}$ **then** ▷ Prendre le prochain événement externe
      $t_{in_i} \leftarrow \mathcal{C}_i^j.getEarliestEventTime(k)$
      $ein_i \leftarrow \mathcal{C}_i^j.getEarliestEvent(k)$
      $p \leftarrow k$ ▷ Sauvegarde du port devant recevoir le prochain événement
      $c \leftarrow j$ ▷ Sauvegarde de l'artefact contenant le prochain événement.
    end if
  end for
  ▷ Calcul de EOT_i et mise à jour des artefacts de couplage de sortie
  if $\mathsf{EOT}_i \neq Lookahead_i(nt_i, EIT_i, t_{in_i})$ **then** ▷ MAJ des artefacts de couplage de sortie
    $\mathsf{EOT}_i \leftarrow Lookahead_i(nt_i, EIT_i, t_{in_i})$
    $\forall (k,j) \in \mathsf{OUT}_i : \mathcal{C}_j^i.setLinkTime(\mathsf{EOT}_i)$
  end if
  ▷ Calcul de l'événement sécurisé (interne ou externe) le plus imminent
  if $(nt_i \leq t_{in_i})$ **and** $(nt_i \leq \mathsf{EIT}_i)$ **and** $(nt_i \leq \mathsf{Z})$ **then** ▷ Si c'est un événement interne
    $\mathcal{I}_i.processInternalEvent(nt_i)$ ▷ Exécution de l'événement
    for all $(k,j) \in \mathsf{OUT}_i$ **do** ▷ Envoi des événements externes résultants
      $eout_i^k \leftarrow \mathcal{I}_i.getOutputEvent(y_i^k)$
      if $eout_i^k \neq \varnothing$ **then**
        $\mathcal{C}_j^i.post(eout_i^k, nt_i)$
      end if
    end for
    $nt_i \leftarrow \mathcal{I}_i.getNextInternalEventTime()$
  else if $(t_{in_i} < nt_i)$ **and** $(t_{in_i} \leq \mathsf{EIT}_i)$ **and** $(t_{in_i} \leq \mathsf{Z})$ **then** ▷ Si c'est un événement externe
    $\mathcal{I}_i.processExternalEvent(ein_i, t_{in_i}, x_i^p)$ ▷ Exécution de l'événement externe
    $\mathcal{C}_i^c.removeEarliestEvent(p)$
    $nt_i \leftarrow \mathcal{I}_i.getNextInternalEventTime()$
  end if
end while

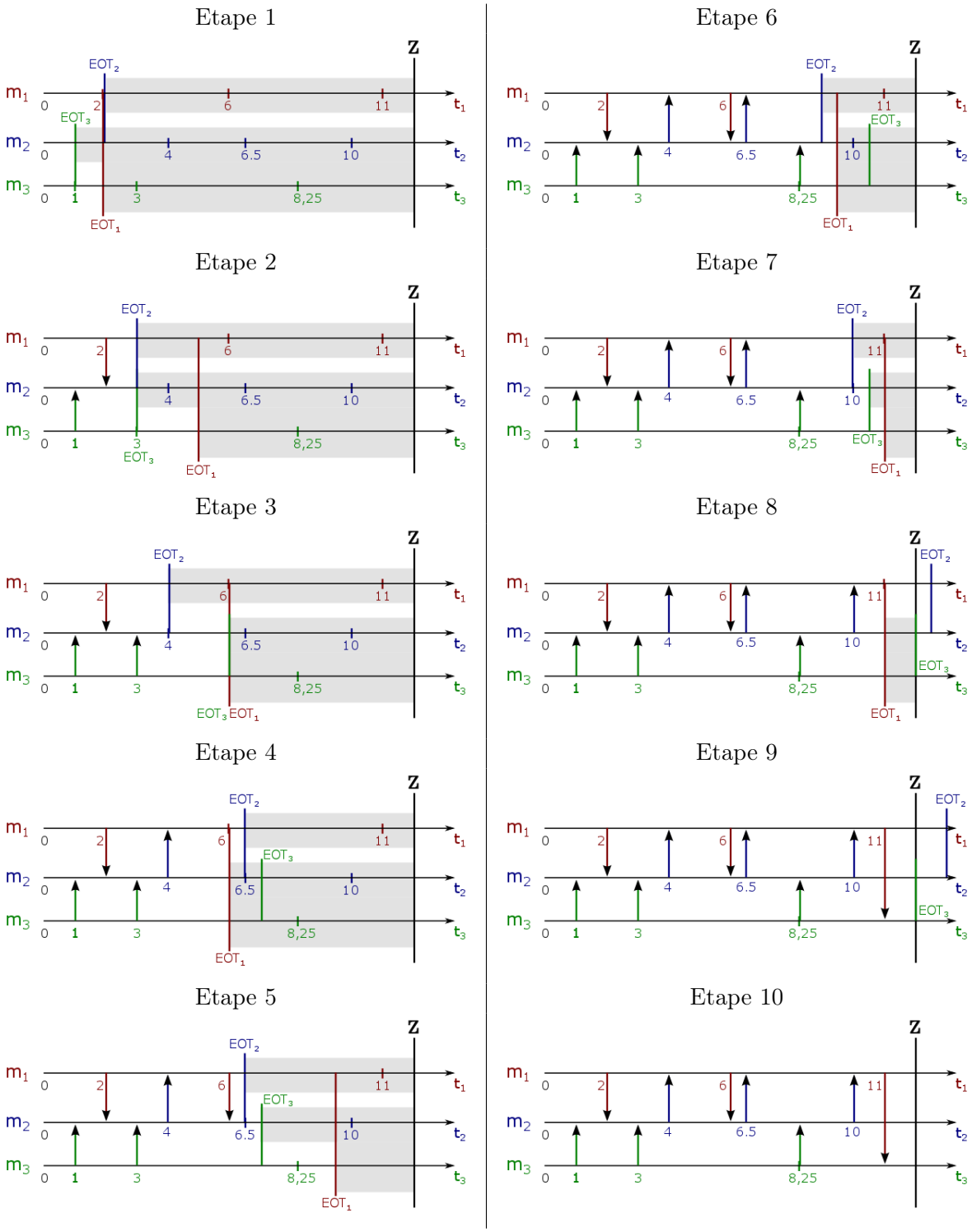


FIGURE 6.5 – Exemple de déroulement de la simulation parallèle du multi-modèle de la Figure 6.1a. Les événements internes planifiés sont disposés sur les axes horizontaux représentant le temps simulé. Chaque flèche verticale indique l'envoi d'un événement externe.Les événements en dehors des zones grises correspondent aux événements sécurisés.

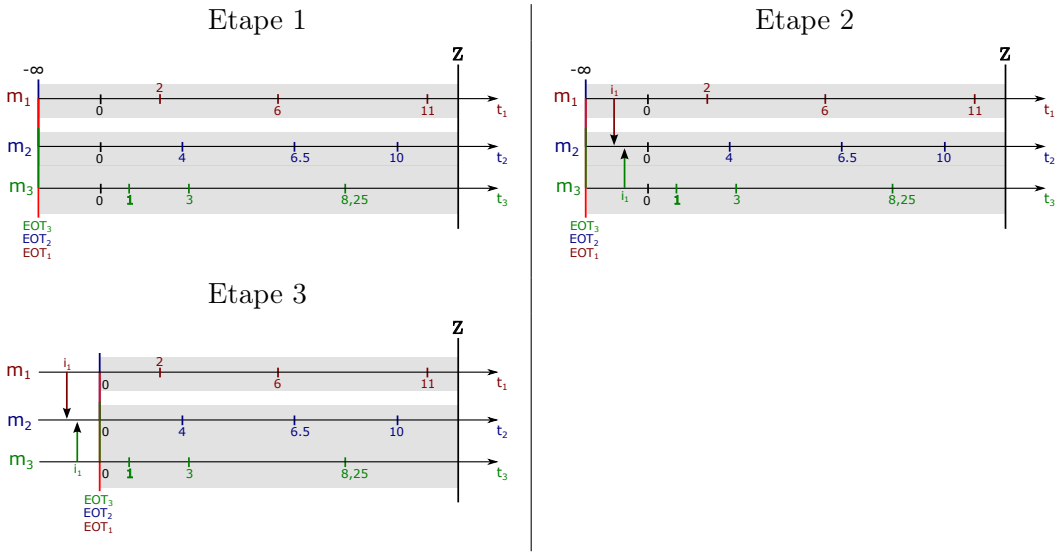


FIGURE 6.6 – Exemple de phase d'initialisation du multi-modèle de la Figure 6.1a. Les événements internes planifiés sont disposés sur les axes horizontaux représentant le temps simulé. Chaque flèche verticale indique l'envoi d'un événement externe. Les événements en dehors des zones grises correspondent aux événements sécurisés. A l'étape 1, tous les *Link Times* sont fixés à $-\infty$. Les m-agents ne peuvent pas alors exécuter les événements internes de leurs modèles. A l'étape 2, les m-agents qui le doivent (ici $\mathcal{A}_1$ et $\mathcal{A}_2$) envoient leurs événements d'initialisation (notés i_1 et i_2). A l'étape 3, les agents ont fini d'envoyer leurs événements d'initialisation et fixent les *Link Times* de leurs artefacts de couplage de sortie à 0 (ici $\Psi = 0$).

6.5.5 Début et fin de la co-simulation

Initialisation de la simulation

Avant de démarrer la simulation, les modèles composant le multi-modèle peuvent avoir besoin d'être initialisés avec des informations issues des autres modèles [Siebert, 2011]. On peut par exemple vouloir initialiser une variable d'état X d'un modèle m_i à la valeur de la variable d'état Y d'un modèle m_j. Les modèles doivent alors s'échanger des événements externes avant le début de la simulation. Dans cette optique, **nous définissons un comportement spécifique aux m-agents devant partager les données initiales de leurs modèles.**

Comme les événements d'initialisation n'ont pas lieu pendant la simulation à proprement parler, il doivent avoir lieu avant (dans le temps simulé) le début de la simulation. Autrement dit, pour une simulation devant commencer à la date Ψ, l'estampille temporelle des événements externes d'initialisation doit être strictement inférieure à Ψ.

Pour empêcher les m-agents de débuter la simulation sans avoir reçu d'événement d'initialisation, nous initialisons le *Link Time* de chaque artefact de couplage d'entrée à $-\infty$. Chaque m-agent changera le *Link Time* de ses artefacts de couplage de sortie à Ψ après avoir déposé ses événements externes d'initialisation (ou directement s'il n'a pas à envoyer d'événement d'initialisation). Cette procédure garantit que chaque m-agent intégrera avant le début de la simulation les événements externes d'initialisation de son modèle. La Figure 6.6 récapitule le déroulement de la phase d'initialisation.

Condition d'arrêt de la simulation

Nous considérons ici que nous voulons simuler un multi-modèle jusqu'à une date Z ($Z \in \mathbb{R}$). Un m-agent $\mathcal{A}_i$ peut arrêter la simulation quand ces trois conditions sont remplies :

— m_i n'a plus d'événement interne planifié avant la date Z. Autrement dit $\mathsf{EOT}_i > Z$.

— $\mathcal{A}_i$ n'a plus d'événement externe ayant d'estampille temporelle inférieure ou égale à Z en attente dans ses artefacts de couplage d'entrée. Autrement dit $t_{in_i} > Z$.

— Aucun nouvel événement externe d'entrée ayant une estampille temporelle inférieure ou égale à Z ne sera déposé dans aucun des artefacts de couplage d'entrée de $\mathcal{A}_i$. Autrement dit $EIT_i > Z$.

Autrement dit, la fin de la simulation correspond au prédicat :

$$endOfSimulation = (\mathsf{EOT}_i > Z) \wedge (\mathsf{EIT}_i > Z) \wedge (t_{in_i} > \mathsf{Z}) \tag{6.4}$$

6.5.6 Synthèse

Nous avons défini dans cette section l'algorithme de co-simulation parallèle et conservatif de MECSYCO. Cet algorithme qui correspond au comportement des m-agents et qui permet d'exécuter un multi-modèle de manière décentralisée, est en accord avec le paradigme A&A de AA4MM où chaque m-agent est une entité autonome évoluant selon sa propre dynamique.

Dans la section suivante, nous montrons comment, en étendant le concept d'opération de transformation, nous conservons avec cet algorithme la capacité des artefacts de couplage à intégrer des représentations hétérogènes.

6.6 Opérations de transformation

Comme nous l'avons vu en Section 4.4.2, les opérations de transformation des artefacts de couplage vont permettre d'effectuer des traductions entre des modèles ayant des représentations différentes du système cible. A partir des informations échangées entre modèles, ces opérations vont effectuer des changements (1) d'échelle spatiale (e.g. passer de km à m), (2) de niveau de résolution (e.g. passer de voitures individuelles à un débit moyen de voitures par heure), et (3) d'échelle temporelle (e.g. passer de minutes à heures). Cependant, comme nous le montrons ci-dessous, ces opérations ne s'appliquent pas sur les mêmes informations avec les nouvelles spécifications de MECSYCO.

En effet, d'un côté, les opérations spatiales ou de changement de niveau de résolution vont s'appliquer sur les événements externes échangés entre deux modèles m_i et m_j (e.g. une opération changeant des coordonnées de voitures de km à m). Chacune de ces opérations, que nous appellerons opérations événementielles, va s'appliquer sur un des liens $l \in \mathsf{L}^i_j$.

A l'opposé, les opérations temporelles vont effectuer des transformations sur toutes les informations temporelles échangées entre deux modèles m_i et m_j. Ces informations concernent le *Link Time* de l'artefact de couplage (autrement dit EOT_i), et les estampilles temporelles des événements externes.

Nous étendons alors l'ensemble des liens L^i_j d'un artefact de couplage $\mathcal{C}^i_j$. Chaque couple $(n, k) \in \mathsf{L}^i_j$ faisant le lien entre le n^{eme} port de sortie de m_i et le k^{eme} port d'entrée de m_j, est maintenant associé à l'ensemble ordonné $O^{i,n}_{j,k}$ des opérations événementielles s'appliquant séquentiellement sur les événements externes échangés entre les ports y^n_i et x^k_j. L^i_j contient maintenant les triplets $(n, k, O^{i,n}_{j,k})$.

Nous étendons également l'artefact de couplage $\mathcal{C}_j^i$ pour qu'il contienne maintenant l'ensemble ordonné TO_j^i des opérations temporelles s'appliquant séquentiellement sur les estampilles temporelles de tous les événements externes échangés entre m_i et m_j, et tous les nouveaux *Link Times* de $\mathcal{C}_j^i$.

6.7 Conclusion

Nous avons proposé dans ce chapitre les spécifications opérationnelles de l'intergiciel de co-simulation MECSYCO. Cet intergiciel permet d'intégrer l'hétérogénéité du muti-modèle d'un système complexe. **MECSYCO offre alors une base logicielle pour simuler de manière rigoureuse le multi-modèle d'un système complexe.**

L'ensemble des spécifications opérationnelles que nous avons définies ici sont données par les Tables 6.1, 6.3 et 6.4 ainsi que par l'algorithme 1.

L'intergiciel se repose d'une part sur le formalisme DEVS pour intégrer des modèles formels hétérogènes, et d'autres part sur les concepts A&A de l'approche AA4MM pour gérer l'interopérabilité des logiciels de simulation et l'intégration des représentations. Pour ce faire, nous avons étendu les spécifications opérationnelles des concepts de AA4MM afin que ces derniers puissent représenter et simuler un modèle couplé DEVS. Cette extension inclut (1) un nouveau protocole de simulation, et (2) un nouvel algorithme conservatif de co-simulation parallèle basé sur l'algorithme CMB pour DEVS.

Il est important de noter que les modifications apportées à AA4MM ont été faites dans le respect de son paradigme A&A. En effet, comme dans AA4MM, une co-simulation MECSYCO est effectuée par des m-agents autonomes qui interagissent et se coordonnent par l'intermédiaire d'artefacts passifs [Siebert, 2011]. MECSYCO reprend alors les propriétés logicielles de AA4MM, à savoir une architecture modulaire permettant une exécution parallèle décentralisée (et distribuable) d'un multi-modèle.

Il faut également noter que l'algorithme de simulation parallèle de MECSYCO respecte le principe de fonctionnement de l'algorithme CMB pour DEVS. En effet, la coordination de la simulation est effectuée grâce aux communications locales (selon le graphe de dépendance du multi-modèle) des EOT, et aux calculs des EIT par les simulateurs parallèles. Les preuves de CMB concernant le respect de la contrainte de causalité et l'absence d'interblocage établies dans [Zeigler et al., 2000, p. 269-270], sont alors transposables à MECSYCO.

Toutefois, MECSYCO ne répond que partiellement à notre objectif de décrire et simuler un multi-modèle dans le cadre rigoureux de la démarche de M&S. En effet, si l'intergiciel permet de simuler rigoureusement un multi-modèle hétérogène, les spécifications opérationnelles que nous avons proposées ne permettent pas de vérifier l'implémentation des multi-modèles de manière systématique : on ne peut pas passer automatiquement d'un schéma AA4MM à un programme exécutable. Pour répondre à ce problème, nous montrons dans le chapitre suivant comment nous systématisons l'implémentation des multi-modèles, en suivant une démarche d'ingénierie dirigée par les modèles.

Chapitre 7

Approche d'Ingénierie Dirigée par les Modèles pour AA4MM

Sommaire

7.1 Introduction

L'intergiciel de co-simulation MECSYCO que nous avons défini dans le chapitre précédent se repose sur le paradigme de l'approche AA4MM, c'est à dire sur un ensemble de concepts Agents & Artifacts (A&A) permettant de décrire le multi-modèle d'un système complexe comme un système multi-agent. Ces concepts sont associés à des spécifications opérationnelles basées sur le formalisme DEVS. **Un système multi-agent AA4MM est alors déclinable sous une forme exécutable** (i.e. simulable).

Cependant, **le problème est que le passage d'un multi-modèle AA4MM à son implémentation n'est pas systématique**, c'est à dire que le code MECSYCO doit être écrit manuellement à partir de la description d'un multi-modèle AA4MM. Comme expliqué au chapitre 2, l'implémentation d'un multi-modèle peut alors comporter des erreurs qui vont affecter les résultats de simulation.

Afin de garantir la vérification d'un multi-modèle AA4MM à chaque itération de la démarche de multi-modélisation et simulation, **nous proposons dans ce chapitre d'automatiser l'implémentation d'un multi-modèle, c'est à dire que nous allons définir une suite de transformations, qui vont permettre de passer de la description d'un multi-modèle à un programme exécutable (i.e. une simulation)**.

Cela correspond à une démarche d'Ingénierie Dirigée par les Modèles (IDM). En suivant cette démarche nous créons un environnement graphique de multi-modélisation pour

AA4MM permettant (1) de décrire des multi-modèles AA4MM, et (2) de générer automatiquement le code de simulation MECSYCO correspondant.

Ce chapitre est articulé comme suit. Dans un premier temps, nous présentons en Section 7.2 un aperçu bibliographique de l'IDM. Nous proposons ensuite en Section 7.3 une approche d'IDM pour AA4MM. Enfin, nous détaillons en Section 7.4 MECSYCO-Java, l'implémentation de cette proposition en langage Java.

7.2 L'ingénierie dirigée par les modèles

Dans l'Ingénierie Dirigée par les Modèles (IDM), un modèle n'est pas considéré uniquement comme une représentation d'un système cible, mais également comme la spécification d'un logiciel [Seidewitz, 2003]. Dans cette optique, l'IDM va chercher à automatiser l'implémentation d'un logiciel à partir de la définition d'un modèle. Le principe de l'IDM consiste alors à appliquer un ensemble de transformations sur un modèle pour obtenir un logiciel [Durak, 2015]. L'intérêt de l'IDM dans le contexte de la modélisation et simulation, est alors qu'elle permet de systématiser le passage d'un modèle à sa simulation et garantit donc la vérification du modèle à chaque itération de la démarche de modélisation et simulation.

L'IDM repose sur la notion de méta-modèle. On peut trouver plusieurs définitions de ce terme dans la littérature. La Définition 21 est la plus littérale, elle nous renseigne sur le positionnement d'un méta-modèle, mais non sur sa fonction et sa nature : un méta-modèle se place à un niveau d'abstraction supplémentaire par rapport aux modèles. La fonction d'un méta-modèle est donnée par la Définition 22 : un méta-modèle définit un langage de modélisation. On peut ainsi définir des méta-modèles pour les différents formalismes de modélisation (e.g. statechart, réseau de Pétri...) ou définir des Domain Specific Langage (DSL), c'est à dire des langages de modélisation métier [Lédeczi et al., 2001]. Enfin, la Définition 23 nous renseigne sur la nature d'un méta-modèle : un méta-modèle est une spécification, c'est à dire qu'il va définir les concepts de base d'un langage et comment ceux-ci peuvent être instanciés et combinés pour former des modèles de manière valide. [Seidewitz, 2003].

Définition 21 *Un **méta-modèle** est un modèle d'un modèle [Sprinkle et al., 2010]*

Définition 22 *Un **méta-modèle** est un modèle qui définit un langage permettant d'exprimer des modèles [OMG, 2003]*

Définition 23 *Un **méta-modèle** est une spécification d'une classe de systèmes étudiés, où chaque système étudié de la classe est lui-même un modèle valide exprimé dans un certain langage de modélisation [Seidewitz, 2003]*

Un méta-modèle est lui-même spécifié à l'aide d'un langage spécifique : un méta-méta-modèle [Bézivin and Gerbé, 2001]. Ce méta-méta-modèle correspond en général à la Meta Object Facility (MOF) de l'Object Management Group (OMG) [OMG, 2014]. Afin d'éviter que la MOF soit elle-même définie par un autre langage, et alors qu'une potentielle infinité de niveaux d'abstraction puisse exister, la MOF est méta-réflexive : elle contient sa propre définition. La MOF est basée sur une simplification de l'Unified Modeling Language (UML) et sur l'Object Constraint Language (OCL) [Nordstrom et al., 1999]. UML permet de définir la syntaxe du langage, c'est à dire quels sont les concepts qui le composent et comment ceux-ci peuvent être associés. OCL permet quant à lui de définir des contraintes sémantiques non exprimables en UML. Ces contraintes vont prendre la forme d'un ensemble de prédicats devant être vrais pour n'importe quel modèle du

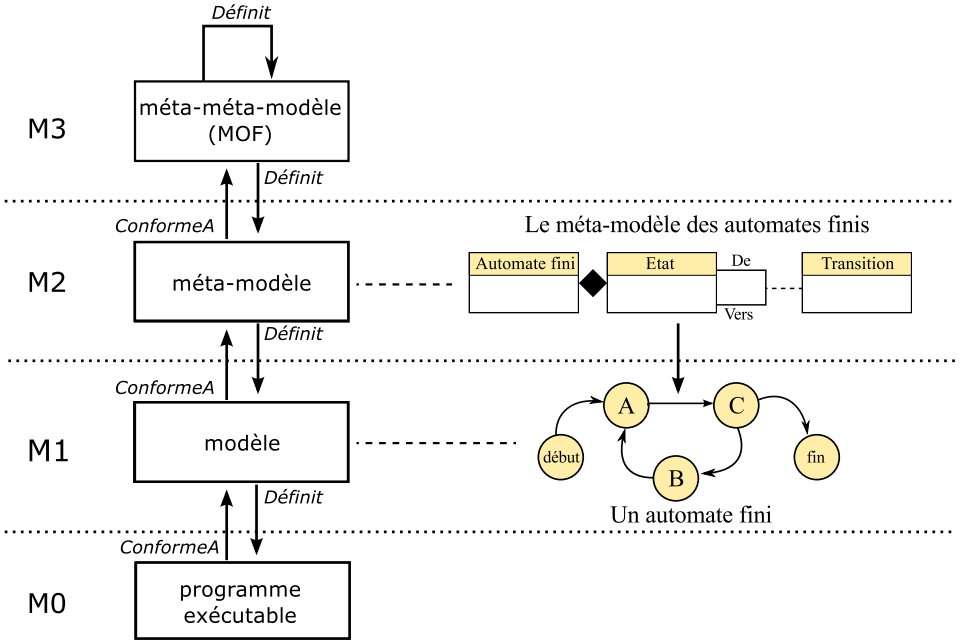


FIGURE 7.1 – Architecture de l'ingénierie dirigée par les modèles et un exemple d'application pour les automates finis. (Inspirée de [Karsai et al., 2000, Lédeczi et al., 2001])

méta-modèle. La MOF impose également que chaque méta-modèle (et donc chaque modèle décrit avec ce méta-modèle) puisse être traduit dans le format XML Metadata Interchange (XMI) basé sur XML. Grâce à cette caractéristique, les méta-modèles peuvent être exportés au format XML et être ainsi facilement analysés et échangés entre différentes implémentations de la MOF. Les différents niveaux d'abstraction de l'ingénierie dirigée par les modèles sont illustrés par la Figure 7.1.

Il existe des outils de méta-modélisation comme le Generic Modeling Environment (GME) [Ledeczi et al., 2001] qui permettent de définir un méta-modèle et, à partir de ce méta-modèle, de générer un environnement graphique de modélisation métier. Cet environnement va permettre de construire des modèles, c'est à dire d'instancier et relier les concepts du langage de modélisation en accord avec les contraintes syntaxiques et sémantiques de ce dernier. L'environnement ainsi généré va alors permettre la construction de modèles de manière rigoureuse, c'est à dire en s'assurant que le modèle obtenu est valide par rapport au langage défini.

Grâce à la notion de méta-modèle, il est possible de définir des transformations s'appliquant sur les modèles. En effet, une fois un méta-modèle défini, il est également possible de lui associer un ensemble de transformations. Ces transformations vont être définies au niveau des méta-modèles et vont s'appliquer au niveau de leurs modèles [Durak, 2015]. Elles peuvent être de deux types :

— **Les transformations de modèle à modèle** relient deux méta-modèles entre eux. Elles vont alors permettre de traduire un modèle écrit dans un langage de modélisation dans un autre langage (voir Figure 7.2).

— **Les transformations de modèle à texte** vont permettre de générer du code de programmation à partir d'un modèle. Grâce à ces transformations, il est possible d'automatiser l'implémentation d'un modèle, et donc sa simulation [Cetinkaya et al., 2010b, Cetinkaya et al., 2010a].

Les transformations de méta-modèle peuvent être écrites dans un langage standardisé. Ainsi,

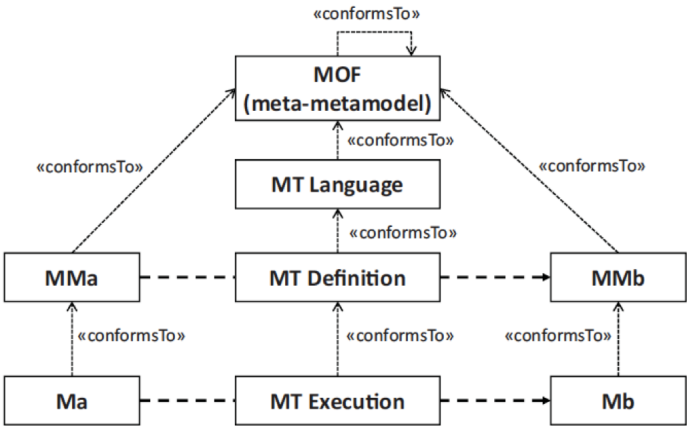


FIGURE 7.2 – Transformation de modèle avec l'ingénierie dirigée par les modèles. Une transformation MT est définie entre deux méta-modèles MMa et MMb, et appliquée sur deux modèles Ma et Mb. (Source [Durak, 2015])

les langages Query/View/Transformation (QVT) [OMG, 2015a] et Atlas Transformation Language (ATL) [Jouault and Kurtev, 2006] permettent d'effectuer des transformations de modèle à modèle. Le langage MOF Model to Text Transformation Language (MOFM2T) [OMG, 2015b] permet quant à lui d'effectuer des transformations de modèles en texte. Une autre solution est d'effectuer la transformation au niveau des fichiers XMI en utilisant le langage de transformation de fichiers XML eXtensible Stylesheet Language Transformations (XSLT) [Bézivin and Gerbé, 2001].

L'intérêt des approches IDM pour la démarche de M&S des systèmes complexes est qu'elles systématisent le passage du modèle formel au modèle exécutable, et donc assurent la vérification du modèle. De plus, la méta-modélisation permet de contraindre le modélisateur dans la conception de modèle et ainsi d'éviter la construction de modèle ayant une structure connue a priori comme étant non valide [Sprinkle et al., 2010], ce qui permet de limiter les itérations de la démarche de modélisation et simulation.

Nous choisissons dans cette thèse d'exploiter ces avantages en définissant une approche d'IDM pour AA4MM. Cette approche, que nous définissons dans la section suivante, (1) formalise un méta-modèle pour AA4MM afin de décrire des multi-modèles de manière non-ambiguë, et (2) définit des transformations de méta-modèle à texte permettant de passer automatiquement d'un schéma AA4MM à un programme exécutable MECSYCO.

7.3 Mise en oeuvre

7.3.1 Principe

La Figure 7.3 détaille l'approche d'IDM que nous proposons pour AA4MM. Cette approche suit l'architecture présentée précédemment en Figure 7.1,

Comme nous l'avons vu en Chapitre 3, un système complexe cible est représenté de manière partielle par plusieurs modèles déjà implémentés dans des logiciels de simulation. Nous avons donc trois niveaux d'abstraction ici : (1) le niveau du système réel, (2) le niveau des simulateurs qui, une fois intégrés dans une co-simulation, vont simuler l'évolution du système, et (3) le niveau des modèles qui, une fois interconnectés dans un multi-modèle, vont représenter le système cible.

L'intergiciel MECSYCO que nous avons défini est situé au niveau simulateur car il va permettre d'effectuer la co-simulation du système cible. **Pour systématiser l'implémentation d'un multi-modèle nous devons donc assurer le passage automatique du niveau modèle au niveau simulateur.**

Pour ce faire, nous définissons un quatrième niveau d'abstraction, celui du méta-modèle. Ce méta-modèle va définir le langage de multi-modélisation permettant de construire des multi-modèles (situés au niveau des modèles). Ce langage est ici celui du paradigme de AA4MM. **Nous formalisons alors le méta-modèle de AA4MM que nous appellerons Méta-AA4MM.**

Comme nous l'avons vu en Section 7.2, nous devons nous doter d'un langage particulier pour définir Méta-AA4MM : un méta-méta-modèle situé à un niveau d'abstraction supplémentaire. Nous choisissons ici UML comme méta-méta-modèle car il constitue un langage de référence standardisé par l'OMG. Nous détaillons Méta-AA4MM en Section 7.3.2.

En nous appuyant sur Méta-AA4MM, nous définissons les règles de transformation d'un multi-modèle AA4MM en un programme exécutable MECSYCO. Une fois implémentées, ces règles permettent la génération automatique du code de co-simulation MECSYCO des multi-modèles AA4MM. Nous détaillons le passage d'un multi-modèle AA4MM à une simulation MECSYCO dans la Section 7.3.3. Nous finissons enfin en Section 7.3.4 par un résumé du processus de multi-modélisation proposé.

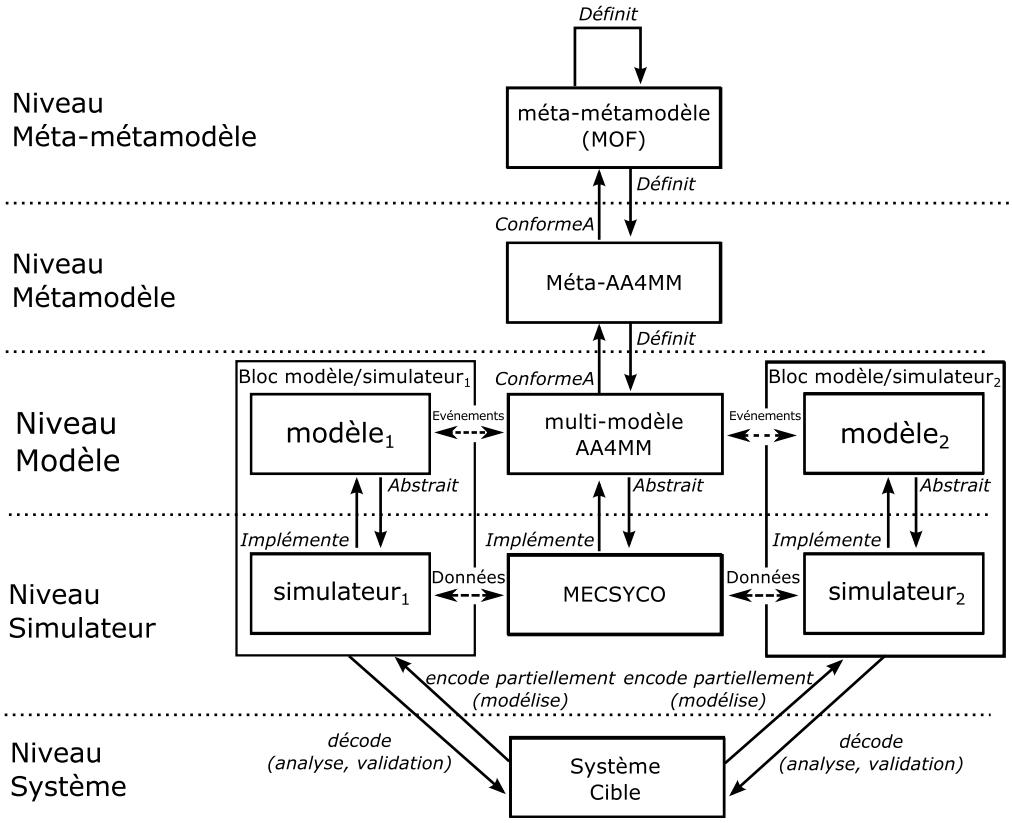
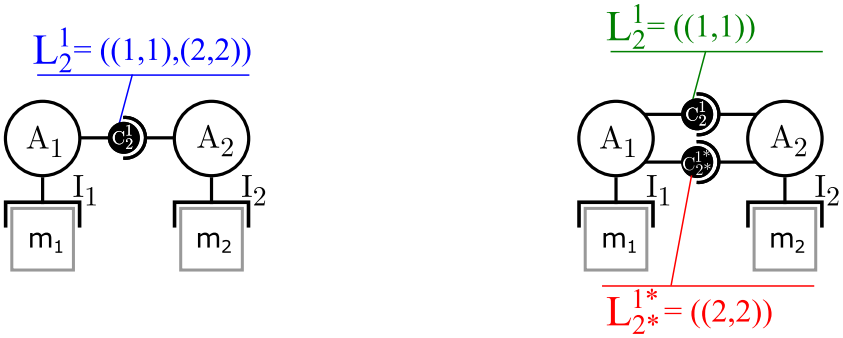


FIGURE 7.3 – L'approche d'ingénierie dirigée par les modèles pour AA4MM (les relations entre le système cible et les blocs modèle/simulateur sont inspirées de [Seck and Honig, 2012]).



(a) Exemple de multi-modèle AA4MM. (b) multi-modèle AA4MM développé.

FIGURE 7.4 – Développement de la notation graphique de AA4MM

7.3.2 Le méta-modèle AA4MM

Remarque préliminaire : développement de la notation graphique AA4MM

Avec les spécifications de MECSYCO que nous avons présentées au Chapitre 6, un artefact de couplage $\mathcal{C}_j^i$ peut gérer l'ensemble de liens L_j^i entre deux modèles m_i et m_j.

Lorsque l'on se place dans un cadre de multi-modélisation graphique comme ici, cette caractéristique a l'inconvénient de ne pas expliciter graphiquement les liens contenus dans L_j^i comme montré en Figure 7.4a. Pour résoudre ce problème, nous restreignons ici les capacités des artefacts de couplage : un artefact de couplage est utilisé pour chaque lien $(n, k) \in \mathsf{L}_j^i$. De cette manière, chaque lien entre les modèles est représenté graphiquement par un artefact de couplage spécifique (Figure 7.4b). Il est à noter que ce développement de la notation graphique de AA4MM n'a pas d'impact sur les spécifications opérationnelles proposées dans le chapitre précédent.

Les concepts de Méta-AA4MM

La première étape pour définir Méta-AA4MM consiste à identifier les concepts qui vont servir à décrire un multi-modèle.

Nous pouvons noter que AA4MM fait intervenir dans la description d'un multi-modèle des concepts qui n'ont pas tous le même niveau de généricité. Par exemple, les m-agents et les artefacts de couplage sont génériques et peuvent être utilisés pour n'importe quel multi-modèle. En revanche, les artefacts de modèles et les opérations sont spécifiques à un domaine d'application particulier. Ces concepts vont donc changer en fonction du multi-modèle.

Afin de séparer les concepts de AA4MM du plus générique au plus spécifique, nous proposons une structuration en niveaux ontologiques [Gašević et al., 2007] du méta-modèle AA4MM. Ces niveaux ontologiques sont orthogonaux aux niveaux d'abstraction présentés en Section 7.3.1.

Un niveau ontologique est noté $\mathcal{O}_i$. Les concepts d'un niveau ontologique $\mathcal{O}_i$ sont plus spécifiques que les concepts du niveau $\mathcal{O}_{i-1}$, (mis à part $\mathcal{O}_1$ qui correspond au niveau de base). Les niveaux ontologiques, que nous présentons dans les sections suivantes, sont les suivants :

— $\mathcal{O}_1$ correspond au **paradigme A&A**.
— $\mathcal{O}_2$ correspond aux **concepts génériques proposés par AA4MM** pour décrire un multi-modèle comme un système A&A.
— $\mathcal{O}_3$ correspond **au niveau domaine**. Il contient les concepts spécifiques à un domaine d'application particulier.

Afin de définir les associations autorisées entre ces concepts, nous ajoutons des contraintes

syntaxiques au méta-modèle. Le méta-modèle Méta-AA4MM que nous proposons et détaillons dans les sections suivantes, correspond alors à l'ensemble des concepts avec leurs contraintes syntaxiques étalées sur les trois niveaux ontologiques. Ce méta-modèle est donné dans la Figure 7.5.

Concepts du niveau $\mathcal{O}_1$

Le niveau paradigmatique définit les trois concepts fondamentaux de A&A sur lesquels AA4MM se base.

— **Les m-agents** sont les entités autonomes du système.
— **L'environnement** est un espace de travail pour les m-agents.
— **Les artefacts** sont les entités passives composant l'environnement.

Concepts du niveau $\mathcal{O}_2$

Nous trouvons à ce niveau les concepts utilisés par AA4MM pour décrire un multi-modèle.

— **Les modèles** sont les représentations d'un système cible déjà implémentées dans les logiciels de simulation. Nous faisons ici le choix de considérer les modèles du multi-modèle comme des artefacts au sens où ils vont constituer des outils que les m-agents vont pouvoir utiliser pour simuler l'évolution du système cible.
— Les **artefacts de modèle** sont des artefacts permettant de contrôler un modèle.
— Les **artefacts de couplage** sont des artefacts de communication permettant de stocker et distribuer des événements externes.
— Les **opérations** sont utilisées par les artefacts de couplage pour transformer les informations qu'ils stockent. Comme expliqué en Section 6.6, nous distinguons les **opérations temporelles** et les **opérations événementielles**.
— Les **m-agents de simulation** sont les entités autonomes en charge de la simulation d'un modèle. Le concept de *M-Agent de Simulation* a trois attributs. **Z** correspond à la date maximum jusqu'à laquelle le modèle du m-agent doit être simulé (voir Section 6.5.5). **Dt** correspond au délai minimum de propagation du modèle du m-agent (voir Section 6.5.3). **Init** indique si le m-agent doit générer les données initiales de son modèle et les envoyer aux autres m-agents (voir Section 6.5.5).

Concepts du niveau $\mathcal{O}_3$

A ce niveau se trouvent les concepts qui sont spécifiques à un domaine d'application :

— Les **modèles** sont les parties pré-existantes du domaine.
— Chaque **artefact de modèle** est spécifique au logiciel de simulation d'un modèle.
— Chaque **opération** des artefacts de couplage est spécifique aux interactions entre deux modèles.

Contraintes Syntaxiques de Méta-AA4MM

Afin d'interdire la construction de multi-modèle ne respectant pas les spécifications de MECSYCO, nous définissons les contraintes syntaxiques de Méta-AA4MM. Ces contraintes garantissent que les multi-modèles construits en suivant Méta-AA4MM pourront être simulés grâce à MECSYCO.

Pour représenter les connexions autorisées entre les concepts de Méta-AA4MM décrits précédemment, nous définissons les contraintes syntaxiques (i.e. associations UML) suivantes :

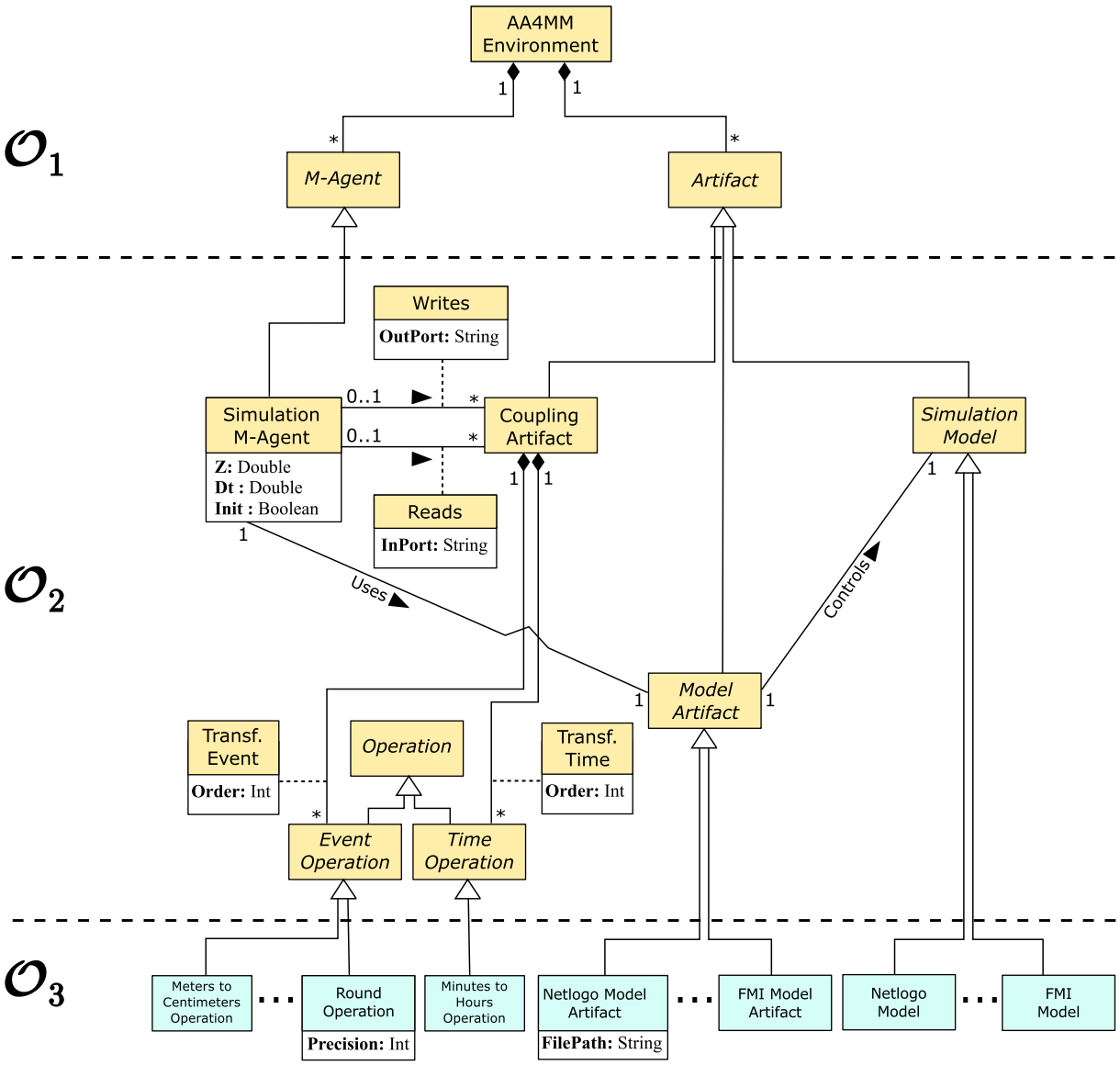


FIGURE 7.5 – Méta-AA4MM formalisé en UML. Les concepts en italique sont abstraits (i.e. ils ne peuvent être instanciés). Les concepts du niveau $\mathcal{O}_3$ (en bleu) sont spécifiques aux domaines d'application et sont ici donnés à titre d'exemples.

— L'environnement de AA4MM est composé d'un ou plusieurs m-agents et d'un ou plusieurs artefacts.
— Un artefact de couplage peut utiliser séquentiellement des opérations événementielles pour transformer les événements externes échangés (cf. **Transf.Event**). L'ordre d'application des opérations est défini par l'attribut **Ordre** de *Transf.Event*.
— Un artefact de couplage peut utiliser séquentiellement des opérations temporelles pour transformer les informations temporelles échangées (cf. **Transf.Time**). L'ordre d'application des opérations est défini par l'attribut **Ordre** de *Transf.Time*.
— Un m-agent peut avoir plusieurs artefacts de couplage de sortie associés aux ports de sortie de son modèle.
— Un m-agent peut avoir plusieurs artefacts de couplage d'entrée associés aux ports d'entrée de son modèle.
— Un m-agent est associé à un et un seul artefact de modèle.
— Un artefact de modèle est associé à un et un seul modèle.

Méta-AA4MM ne nécessite pas de contraintes sémantiques OCL. Ces contraintes peuvent toutefois être rajoutées si le domaine d'application le nécessite. Par exemple, dans un cadre particulier de modélisation de réseaux électriques, il est possible d'interdire la connexion d'un port de sortie *Tension* au port d'entrée *Intensité* d'un autre modèle. Il suffit pour cela d'interdire qu'un artefact de couplage ait à la fois (1) un lien **writes** avec un attribut **OutPort** égal à "$Tension$" et (2) un lien **reads** avec un attribut **InPort** égal à "$Intensite$".

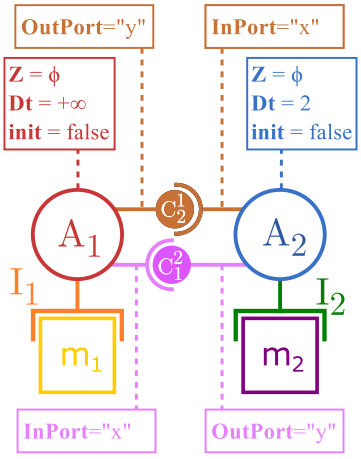
7.3.3 Passage d'un multi-modèle AA4MM à un code MECSYCO

Le dispositif que nous proposons pour passer d'un multi-modèle à du code est composé de deux parties. D'une part, nous définissons une bibliothèque partagée implémentant les spécifications opérationnelles de MECSYCO suivant une conception orientée objet. D'autre part, nous proposons des règles de transformation qui vont générer le code de la simulation MECSYCO, en instanciant les classes de notre bibliothèque. Nous détaillons maintenant ces deux parties.

Bibliothèque partagée implémentant MECSYCO

La bibliothèque partagée que nous proposons implémente chaque concept de Méta-AA4MM comme une classe, et chaque contrainte syntaxique comme une méthode. Les méthodes suivantes permettent de concrétiser les associations du méta-modèle AA4MM dans les objets de la bibliothèque :

— L'association **writes** est concrétisée par la méthode $\mathcal{A}_i.addOutputCouplingArtifact(\mathcal{C}_j^i, OutPort)$ qui ajoute l'artefact de couplage de sortie $\mathcal{C}_j^i$ au m-agent $\mathcal{A}_i$, et ajoute le couple $(OutPort, j)$ dans OUT_i.
— L'association **reads** est concrétisée par la méthode $\mathcal{A}_i.addInputCouplingArtifact(\mathcal{C}_i^j, InPort)$ ajoutant l'artefact de couplage d'entrée $\mathcal{C}_i^j$ au m-agent $\mathcal{A}_i$, et ajoute le couple $(j, InPort)$ dans IN_i.
— L'association **uses** est concrétisée par la méthode $\mathcal{A}_i.setModelArtifact(\mathcal{I}_i)$ associant l'artefact de modèle $\mathcal{I}_i$ au m-agent $\mathcal{A}_i$.
— L'association **Transf. Event** est concrétisée par la méthode $\mathcal{C}_j^i.addEventOperation(o_j^i)$ ajoutant l'opération événementielle o_j^i à l'ensemble des opérations événementielles 0_j^i.
— L'association **Transf. Time** est concrétisée par la méthode $\mathcal{C}_j^i.addTimeOperation(ot_j^i, Order)$ ajoutant l'opération événementielle ot_j^i à l'ensemble des opérations événementielles $0T_j^i$.



(a) Multi-modèle AA4MM

$\mathcal{A}_1 = new\ MAgent(\phi, +\infty, \mathsf{false});$
$\mathcal{A}_2 = new\ MAgent(\phi, 2, \mathsf{false});$
$\mathcal{C}_2^1 = new\ CouplingArtifact();$
$\mathcal{C}_1^2 = new\ CouplingArtifact();$
$\mathcal{I}_1 = new\ ModelArtifact(m_1);$
$\mathcal{I}_2 = new\ ModelArtifact(m_2);$
$\mathcal{A}_1.addOutputCouplingArtifact(\mathcal{C}_2^1, "y");$
$\mathcal{A}_1.addInputCouplingArtifact(\mathcal{C}_1^2, "x");$
$\mathcal{A}_2.addOutputCouplingArtifact(\mathcal{C}_1^2, "y");$
$\mathcal{A}_2.addInputCouplingArtifact(\mathcal{C}_2^1, "x");$
$\mathcal{A}_1.setModelArtifact(\mathcal{I}_1);$
$\mathcal{A}_2.setModelArtifact(\mathcal{I}_2);$

(b) Code généré correspondant

FIGURE 7.6 – Exemple de génération de code à partir d'un multi-modèle AA4MM. Les couleurs indiquent les équivalences entre les symboles et le texte généré.

Règles de transformation d'un multi-modèle en code

Afin de systématiser le passage entre un multi-modèle AA4MM et un programme MECSYCO, nous définissons les transformations permettant de convertir un multi-modèle AA4MM en texte. Le texte généré à partir d'un multi-modèle va correspondre au code instanciant et connectant les classes de la bibliothèque partagée. Ce code va donc permettre d'effectuer la simulation du multi-modèle. La transformation d'un multi-modèle AA4MM se fait alors en deux étapes :

1. Pour chaque instance d'un concept de Méta-AA4MM, on crée un objet de la classe correspondante dans la bibliothèque. Le texte généré correspond alors à l'appel au constructeur de cette classe.
2. Pour chaque association entre les concepts du multi-modèle on génère le code appelant la méthode correspondante dans la bibliothèque.

La Figure 7.6 montre un exemple de génération de code à partir d'un multi-modèle.

7.3.4 Synthèse : processus de multi-modélisation

La Figure 7.7 résume le processus de multi-modélisation que nous proposons. A partir du méta-modèle générique AA4MM, il est possible d'intégrer un ensemble de concepts spécifiques à un domaine d'application particulier. Ces concepts sont (1) les artefacts de modèle interfaçant les différents simulateurs utilisés dans le domaine, ainsi que (2) les opérations de transformation faisant le pont entre les différentes représentations impliquées. Une fois le méta-modèle complété, celui-ci peut être utilisé pour définir différents multi-modèles représentant les systèmes étudiés dans le domaine. Les multi-modèles peuvent ensuite être automatiquement transcrits sous forme exécutable (i.e. simulable) dans MECSYCO grâce aux règles de transformation. Dans la section suivante, nous montrons comment nous avons implémenté ce processus.

7.4 Implémentation de l'approche IDM en Java

Dans cette section, nous détaillons l'implémentation de l'approche d'IDM pour AA4MM permettant de lancer des simulations MECSYCO en langage Java. Nous montrons tout d'abord

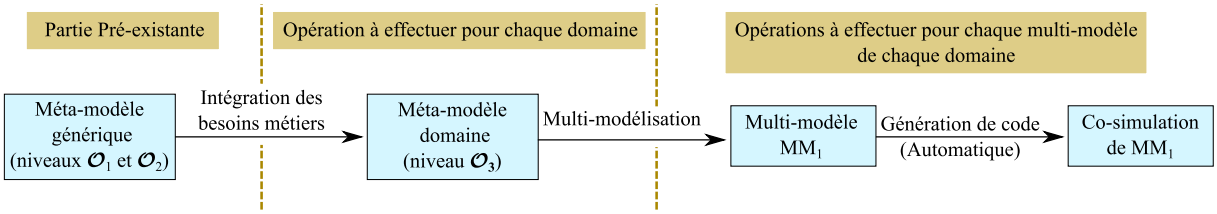


FIGURE 7.7 – Processus de multi-modélisation de AA4MM.

comment nous obtenons un environnement graphique rigoureux de multi-modélisation à partir de Méta-AA4MM (présenté précédemment en Figure 7.5). En implémentant les règles de transformation du méta-modèle définies précédemment, nous montrons ensuite comment nous générons automatiquement le code de simulation MECSYCO de différents multi-modèles AA4MM.

Nous avons tout d'abord implémenté Méta-AA4MM (présenté précédemment dans la Figure 7.5) dans le cadriciel Eclipse Modeling Framework (EMF). EMF se base sur le méta-méta-modèle Ecore implémentant la MOF.

Pour simplifier la génération de l'environnement graphique de multi-modélisation à partir du méta-modèle implémenté dans EMF, nous utilisons l'outil Eugenia [Kolovos et al., 2010]. Cet outil nous permet d'automatiser la configuration de l'environnement graphique dans EMF. Grâce à Eugenia, nous avons uniquement besoin d'associer chaque concept du méta-modèle à son symbole graphique pour effectuer le passage de Méta-AA4MM à l'environnement de multi-modélisation.

L'environnement de multi-modélisation obtenu est composé principalement de trois fenêtres (Figure 7.8) :

- **La bibliothèque de concepts** contient les concepts instanciables du méta-modèle implémenté en langage Java.
- **L'espace de multi-modélisation** permet d'instancier et de connecter les concepts à partir de la bibliothèque de concepts. L'environnement permet uniquement la création de connexions autorisées par Méta-AA4MM. L'environnement de multi-modélisation respecte alors rigoureusement Méta-AA4MM.
- **Le volet d'édition des attributs** permet de fixer les attributs des instances des concepts du méta-modèle.

Les multi-modèles spécifiés dans l'environnement sont décrits sous la forme d'un fichier XML respectant le standard XMI. Pour générer le code de simulation MECSYCO de ces multi-modèles, nous implémentons (1) la bibliothèque partagée dans le langage Java, **nous obtenons alors la bibliothèque MECSYCO-Java**, et (2) les règles de transformation avec le langage XSLT. Le choix de XSLT est motivé par la portabilité et la souplesse de ce langage. Le code XSLT obtenu permet alors de générer le programme exécutable de la simulation MECSYCO-Java de n'importe quel multi-modèle AA4MM décrit en XMI. La Figure 7.9 résume l'implémentation de notre approche IDM pour AA4MM.

7.5 Conclusion

Nous avons proposé dans ce chapitre, une approche d'ingénierie dirigée par les modèles pour AA4MM qui permet d'automatiser l'implémentation d'un multi-modèle AA4MM. **La mécanisation du processus d'implémentation permet d'en éliminer les erreurs humaines, et donc de systématiser la vérification des multi-modèles.**

Dans cette approche IDM, nous avons proposé (1) le méta-modèle Méta-AA4MM définis-

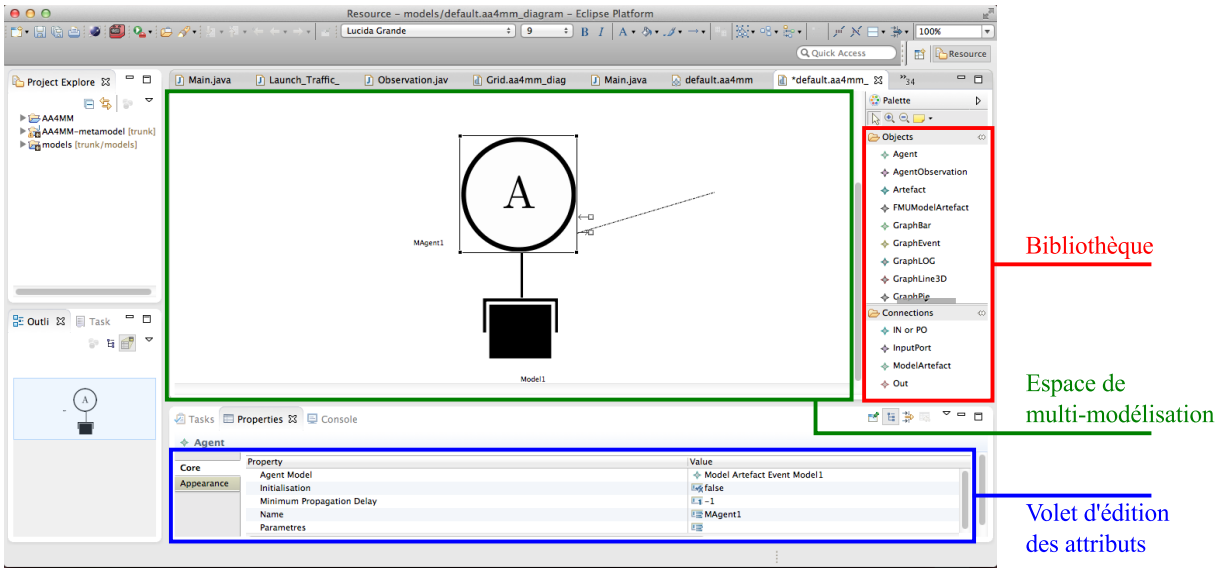


FIGURE 7.8 – L'environnement de Multi-modélisation pour AA4MM généré par Eugenia.

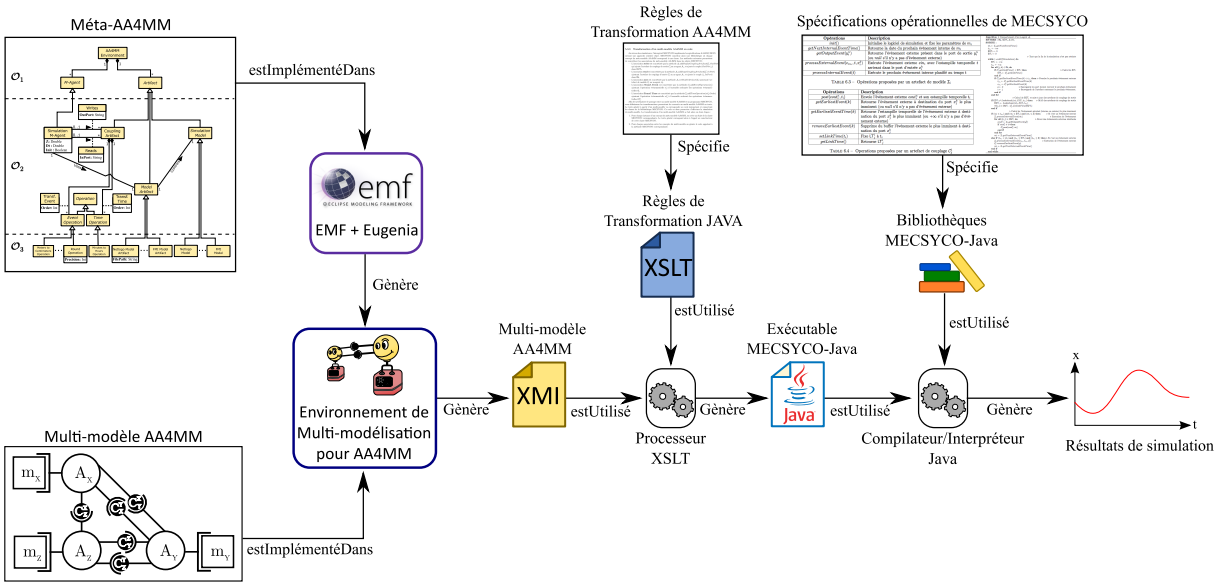


FIGURE 7.9 – Implémentation de l'approche d'IDM pour AA4MM.

sant le langage de multi-modélisation de AA4MM de manière rigoureuse, (2) une bibliothèque partagée implémentant les spécifications opérationnelles de MECSYCO du Chapitre 6 selon une conception orientée objet, (3) des règles de transformation permettant de générer automatiquement le code de simulation d'un multi-modèle.

Pour résumé, l'ensemble des propositions que nous avons présentées dans les Chapitres 6 et 7 permet :

1. de décrire un multi-modèle en combinant des modèles écrits dans plusieurs formalismes, implémentés dans plusieurs simulateurs et ayant des représentations différentes d'un système cible,
2. puis de passer automatiquement à une co-simulation parallèle décentralisée prouvée comme respectant la contrainte de causalité et n'ayant pas d'interblocage.

Nous disposons alors d'une approche permettant de servir de support à la démarche de multi-modélisation et simulation.

Dans le chapitre suivant, nous évaluons les propriétés de notre approche et nous illustrons son utilisation grâce à plusieurs preuves de concept. Nous détaillons ensuite dans le Chapitre 9 comment notre approche a été utilisée avec succès par les membres du projet Inria/EDF R&D Multi-Simulation for Smart Grid (MS4SG) pour simuler des réseaux électriques intelligents.

Chapitre 8

Evaluation

Sommaire

8.1 Introduction

Dans ce chapitre, nous montrons à travers trois preuves de concept réalisées grâce à l'implémentation de notre approche IDM pour AA4MM (décrite au chapitre précédent) que notre approche permet de décrire et simuler un multi-modèle dans le cadre rigoureux de la démarche de M&S. Plus précisément, nous évaluons les propriétés de notre approche suivantes :

D'une manière générale, nous montrons grâce à toutes les preuves de concept de ce chapitre que notre approche permet (1) de décrire des multi-modèles hétérogènes puis (2) de passer automatiquement à leurs co-simulations. Pour ce faire, nous réalisons toutes les preuves de concept de ce chapitre en suivant la même démarche : nous décrivons tout d'abord le multi-modèle dans l'environnement de multi-modélisation pour AA4MM, et nous générons et exécutons ensuite automatiquement le code de sa simulation.

Nous montrons également que MECSYCO-Java permet de simuler rigoureusement des multi-modèles de manière parallèle. En effet, si nous pouvons nous reposer sur les preuves formelles de [Zeigler et al., 2000] en ce qui concerne l'algorithme de co-simulation que nous avons proposé, nous devons toutefois vérifier que notre implémentation MECSYCO-Java

respecte nos spécifications opérationnelles. Dans cette optique, nous montrons en Section 8.2 que :

- **Un modèle couplé DEVS peut être rigoureusement simulé en parallèle.** Nous nous basons ici sur la simulation parallèle du modèle couplé DEVS d'un échange autoroutier.
- **Un modèle écrit dans un autre formalisme que DEVS (ici un formalisme équationnel) peut être simulé en parallèle comme un modèle couplé sans perdre ses propriétés.** Nous montrons cela grâce à une résolution numérique du système d'équations différentielles de Lorenz.

Nous évaluons ces deux preuves de concept selon deux métriques, à savoir (1) l'absence de situation d'interblocage, et (2) la non violation de la contrainte de causalité. La première métrique consiste à vérifier que les simulations MECSYCO-Java se terminent. La deuxième métrique prend un sens différent dans les deux preuves de concepts :

- **Pour la simulation parallèle du modèle couplé d'un échange autoroutier**, nous rappelons que dans le contexte d'une simulation DEVS, ne pas violer la contrainte de causalité signifie respecter le principe de causalité événementielle (voir Définition 15). Nous vérifions alors que dans la simulation MECSYCO-Java du modèle couplé, tous les modèles atomiques exécutent leurs événements dans un ordre temporel croissant.
- **Pour la résolution numérique du système d'équations différentielles de Lorenz,** respecter la contrainte de causalité signifie obtenir une solution identique à celle produite par une simulation séquentielle.

Nous montrons enfin en Section 8.3 comment les concepts de MECSYCO sont utilisés pour intégrer rigoureusement l'hétérogénéité d'un multi-modèle au niveau des représentations, des formalismes et des logiciels de simulation. Nous nous basons pour ce faire sur la simulation d'un multi-modèle représentant de manière hybride (i.e. mélangeant à la fois discret et continu) et multi-niveau un trafic autoroutier.

8.2 Vérification des simulations MECSYCO-Java

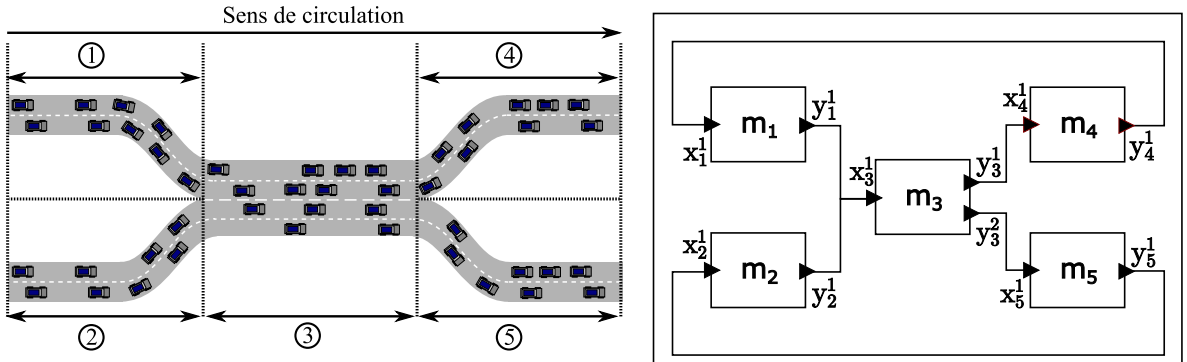
8.2.1 Respect du principe de causalité événementielle pour la simulation d'un modèle couplé DEVS

Principe

Dans cette section, nous montrons à travers une preuve de concept que l'algorithme de MECSYCO-Java permet d'effectuer la simulation parallèle d'un modèle couplé DEVS représentant un échangeur autoroutier (1) en respectant le principe de causalité événementielle (voir Définition 15), et (2) sans se retrouver en situation d'interblocage.

Pour bien comprendre quels sont les besoins de synchronisation de la simulation parallèle, il faut noter que dans le modèle couplé (Figure 8.1b) tous les modèles atomiques sont directement ou indirectement reliés entre eux. Tous les simulateurs vont alors devoir progresser de concert. Les modèles atomiques ont, quant à eux, les propriétés suivantes :

- Ils ont une dynamique interne stochastique. Le modèle couplé aura alors des besoins de synchronisation différents à chaque simulation.
- Les événements externes d'entrée causent la replanification d'événements internes.
- Le port de sortie par lequel un modèle génère un événement externe est déterminé aléatoirement en suivant une loi uniforme.



(a) Echangeur autoroutier (Projection plane de l'espace torique)

(b) Modèle couplé DEVS.

FIGURE 8.1 – Echangeur autoroutier et son modèle couplé

Dans la suite, nous détaillons ce modèle couplé et ses modèles atomiques, et comment il est simulé avec MECSYCO-Java. Enfin, nous expliquons comment nous vérifions au cours des simulations que le principe de causalité événementielle n'a pas été violé.

Modèle couplé DEVS

Le modèle couplé DEVS que nous définissons représente un échangeur autoroutier composé de cinq tronçons (Figure 8.1a). Chaque tronçon est représenté par un modèle atomique. Le modèle couplé est donc composé des modèles atomiques m_1, m_2, m_3, m_4 et m_5. Les événements externes échangés entre ces modèles vont alors correspondre aux voitures changeant de tronçon. Pour les besoins de notre preuve de concept, on considérera que l'échangeur est sur un espace torique. La Figure 8.1b montre comment les cinq modèles sont interconnectés dans le modèle couplé.

Chaque modèle atomique m_i a trois paramètres : la longueur l_i (en km) du tronçon qu'il représente, et les vitesses moyennes minimale v_{min_i} et maximale v_{max_i} (en km/h) que peuvent avoir les voitures dans le tronçon. L'état s_i de chaque modèle correspond à son temps courant de simulation t_i (en h), et à l'ensemble V_i des triplets (id, v_{id}, t_{id}) représentant les identifiants, les vitesses moyennes (en km/h), et les dates de sortie (en h) des voitures transitant actuellement dans le tronçon.

Chaque événement externe d'entrée reçu par un modèle correspond à l'ensemble des identifiants id des voitures entrant dans le tronçon à un instant t. La fonction de transition externe détermine aléatoirement entre v_{min_i} et v_{max_i} la vitesse moyenne v_{id} de chacune de ces voitures. La voiture est ensuite ajoutée à V_i avec une date de sortie égale à $t + (l_i/v_{id})$.

Un événement interne correspond à la sortie d'une ou plusieurs voitures du tronçon. L'exécution du modèle en l'absence d'événement externe d'entrée va alors consister à faire des bonds dans le temps pour aller directement au moment de la sortie de chaque voiture du tronçon. Nous rappelons que l'exécution d'un événement interne dans un modèle DEVS consiste à avancer son horloge interne jusqu'à la date de l'événement, et à appeler (1) sa fonction de sortie pour générer les événements externes à travers les ports de sortie, puis (2) sa fonction de transition interne pour changer l'état du modèle. Le temps du prochain événement interne est ensuite mis à jour grâce à la fonction ta qui détermine pendant combien de temps le modèle va rester dans ce nouvel état en l'absence d'événement externe.

Dans notre preuve de concept, la fonction de sortie de chaque modèle m_i génère un événement externe contenant la liste des identifiants et des vitesses moyennes des voitures ayant le plus

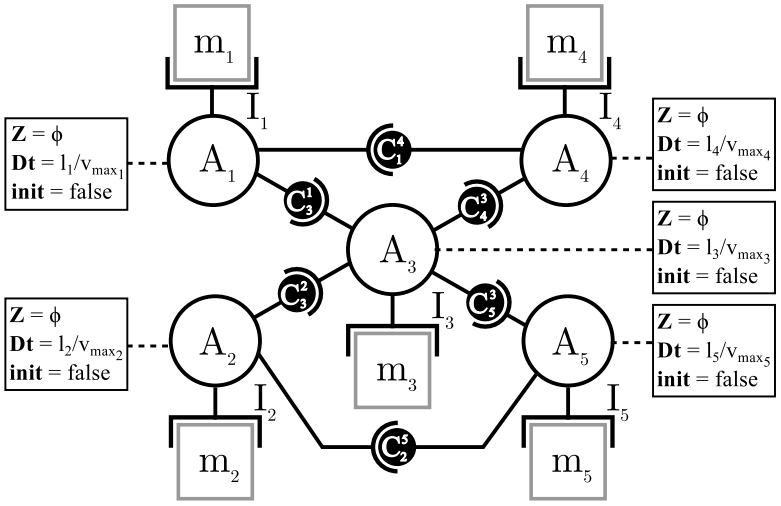


FIGURE 8.2 – multi-modèle AA4MM de l'échangeur autoroutier

imminent t_{id} de V_i. Le port de sortie à travers lequel cet événement est envoyé est déterminé aléatoirement en suivant une loi uniforme. La fonction de transition interne consiste alors à retirer de V_i toutes les voitures ayant le plus imminent t_{id}. Soit t_{s_i} le plus imminent t_{id} de V_i, $ta(s_i)$ correspond alors à $t_{s_i} - t_i$.

Chacun de ces modèles est associé à un simulateur DEVS écrit en Java. La Figure 8.2 et la Table 8.1 montrent le multi-modèle AA4MM permettant de simuler l'échangeur pour une durée Φ.

Dans ce multi-modèle, la procédure de co-initialisation n'est pas nécessaire car les modèles n'ont pas de voitures à s'échanger à l'initialisation (i.e. aucune voiture n'est en train de changer de tronçon à l'initialisation). Le délai minimum de propagation Dt_i (Définition 20) de chaque m-agent $\mathcal{A}_i$ correspond au temps minimum que peut prendre une voiture pour traverser le tronçon de m_i, soit $Dt_i = l_i/v_{max_i}$. La définition des artefacts de modèle est ici triviale car tous les modèles sont déjà formalisés en DEVS.

Respect du principe de causalité événementielle

Pour détecter quand le principe de causalité événementielle a été violé pendant une simulation, nous étendons l'algorithme du comportement des m-agents (Algorithme 1). Cette extension oblige les m-agents à vérifier que leurs modèles calculent leurs événements (externes et internes) suivant un ordre temporel croissant.

Nous avons vérifié expérimentalement que l'implémentation de ce dispositif permet effectivement de détecter une violation du principe de causalité événementielle en biaisant le multi-modèle de l'échangeur. Nous avons en effet introduit dans ce multi-modèle une erreur sur le calcul de Dt_3 en le fixant à $+\infty$. Nous indiquons donc fallacieusement qu'un événement externe d'entrée ne peut pas causer d'événement externe de sortie dans m_3. La conséquence est alors que $\mathcal{A}_3$ va surestimer son EOT pendant la simulation, ce qui entraîne des violations du principe de causalité événementielle pour m_4 ou m_5. On prouve alors que notre dispositif est capable de détecter les violations du principe de causalité événementielle pendant la simulation parallèle du modèle couplé DEVS.

Le dispositif de détection de violation du principe de causalité est le suivant. Chaque m-agent $\mathcal{A}_i$ maintient dans une variable tl_i le temps du dernier événement (interne ou externe) calculé

Descriptions	Notations
Liens de sortie de m_1	$\mathsf{OUT}_1 = \{(1,3)\}$
Liens d'entrée de m_1	$\mathsf{IN}_1 = \{(4,1)\}$
Liens de sortie de m_2	$\mathsf{OUT}_2 = \{(1,3)\}$
Liens d'entrée de m_2	$\mathsf{IN}_2 = \{(5,1)\}$
Liens de sortie de m_3	$\mathsf{OUT}_3 = \{(1,4),(2,5)\}$
Liens d'entrée de m_3	$\mathsf{IN}_3 = \{(1,1),(2,1)\}$
Liens de sortie de m_4	$\mathsf{OUT}_4 = \{(1,1)\}$
Liens d'entrée de m_4	$\mathsf{IN}_4 = \{(3,1)\}$
Liens de sortie de m_5	$\mathsf{OUT}_5 = \{(1,2)\}$
Liens d'entrée de m_5	$\mathsf{IN}_5 = \{(3,1)\}$
Liens de m_1 à m_3	$\mathsf{L}_3^1 = \{(1,1)\}$
Liens de m_2 à m_3	$\mathsf{L}_3^2 = \{(1,1)\}$
Liens de m_3 à m_4	$\mathsf{L}_4^3 = \{(1,1)\}$
Liens de m_3 à m_5	$\mathsf{L}_5^3 = \{(2,1)\}$
Liens de m_4 à m_1	$\mathsf{L}_1^4 = \{(1,1)\}$
Liens de m_5 à m_2	$\mathsf{L}_2^5 = \{(1,1)\}$

TABLE 8.1 – Graphe de dépendance du multi-modèle AA4MM de l'échangeur autoroutier.

dans m_i. tl_i a pour valeur initiale $-\infty$. Avant chaque calcul d'un événement interne (i.e. avant chaque appel à $\mathcal{I}_i.processInternalEvent(nt_i)$), le m-agent vérifie que $tl_i \leq nt_i$. tl_i est ensuite mis à jour à nt_i. De la même manière, avant chaque calcul d'un événement externe (i.e. avant chaque appel à $\mathcal{I}_i.processExternalEvent(e_{in_i}, t_{in_i}, x_i^p)$), le m-agent vérifie que $tl_i \leq t_{in_i}$, et tl_i est ensuite mis à jour à t_{in_i}. Si un de ces deux tests échoue à n'importe quel moment de la simulation, cette dernière est arrêtée et un code d'erreur est généré.

Plan d'expérience

10000 simulations du modèle de l'échangeur ont été effectuées (en remettant $Dt_3 = l_3/v_{max_3}$) sans générer d'erreurs, apportant ainsi la preuve expérimentale que la simulation parallèle MECSYCO-Java du modèle couplé de l'échangeur autoroutier (1) respecte le principe de causalité événementielle, et (2) ne peut se retrouver en situation d'interblocage.

Pour simuler le multi-modèle de l'échangeur autoroutier, l'état de chaque modèle est initialisé à 10 voitures. La date de sortie de chacune de ces voitures est déterminée de la même manière qu'avec la fonction de transition externe : à partir de sa vitesse moyenne déterminée aléatoirement entre v_{min_i} et v_{max_i}.

Le processus d'initialisation des modèles étant stochastique, chaque simulation du multi-modèle aura un état initial différent. Les modèles ont été paramétrés avec $l_i = 50, v_{min} = 1, v_{max} = 5$.

Le dispositif a de plus été par la suite conservé dans MECSYCO-Java. Toutes les preuves de concepts que nous montrons par la suite, ainsi que les simulations du projet MS4SG développées dans le Chapitre 9 ont également respecté le principe de causalité événementielle.

8.2.2 Exactitude des résultats de simulation pour un système d'équations différentielles

Principe général

Dans cette section, nous montrons à travers une preuve de concept qu'une simulation MECSYCO-Java du système de Lorenz respecte la contrainte de causalité (voir Définition 13). Pour ce faire, nous vérifions que la simulation du modèle de Lorenz effectuée en parallèle avec MECSYCO-Java donne les mêmes résultats que la simulation séquentielle de ce même modèle.

Le système de Lorenz [Lorenz, 1963], décrit l'évolution d'une cellule de convection atmosphérique grâce à trois variables d'état (x, y et z) et trois paramètres (α, ρ et β) :

$$\begin{cases} \frac{dx}{dt} = \alpha(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \tag{8.1}$$

Nous choisissons ici de résoudre le système grâce à la méthode d'Euler. Dans la suite, nous détaillons comment le système est simulé, de manière séquentielle avec une simulation monolithique, et en parallèle avec MECSYCO-Java. Nous détaillons enfin, comment les résultats de simulation sont générés et comparés.

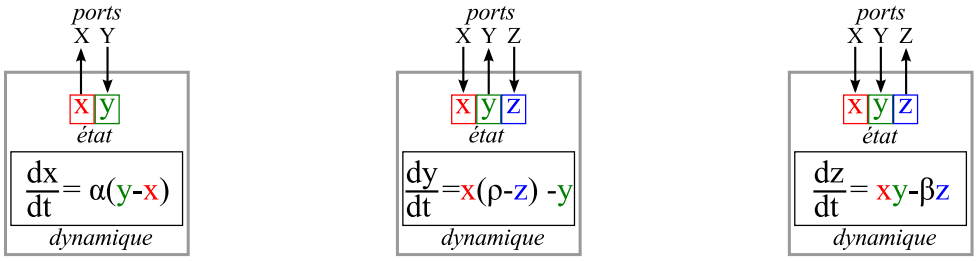
Simulation séquentielle

Pour effectuer la simulation séquentielle du système de Lorenz, un solveur d'équations différentielles est utilisé pour résoudre « simultanément » les trois équations à partir d'un état initial donné. L'Algorithme 2 décrit le comportement du solveur permettant de simuler le système de Lorenz selon la méthode d'Euler. Le solveur permet de résoudre numériquement le système avec des pas de temps d'une durée h jusqu'au temps de simulation Φ. A chaque pas de temps, les valeurs successives des variables x, y, z sont enregistrées pour comparer ensuite les résultats.

Algorithme 2 Solveur simulant le système de Lorenz selon la méthode d'Euler.

ENTREE : $h, \Phi, x, y, z, \alpha, \rho, \beta$

```
t ← 0
while (t < Φ) do                    ▷ tant que la fin de la simulation n'est pas atteinte
    dx ← α(y − x)                   ▷ Calcul des dérivées
    dy ← x(ρ − z) − y
    dz ← xy − βz
    x ← x + h ∗ dx                  ▷ Evolution des variables
    y ← y + h ∗ dy
    z ← z + h ∗ dz
    t ← t + h                       ▷ Evolution du temps de simulation courant
    save(x, y, z, t)                ▷ Sauvegarde des valeurs de x, y, z au temps t
end while
```



(a) solveur m_x (b) solveur m_y (c) solveur m_z

FIGURE 8.3 – Caractéristiques des trois solveurs utilisés pour simuler en parallèle le système de Lorenz

Simulation parallèle avec MECSYCO-Java

Pour simuler en parallèle le système de Lorenz, chaque équation est résolue dans un solveur indépendant doté de ports d'entrée et de sortie. Nous définissons donc trois solveurs m_x, m_y et m_z respectivement chargés de faire évoluer les variables x, y et z. L'état de chacune de ces variables est accessible via un port de sortie spécifique. Chaque solveur va garder une copie locale des variables d'état du système qui lui sont nécessaires pour résoudre son équation. Ces copies locales peuvent être mises à jour via les ports d'entrée des solveurs. La Figure 8.3 décrit les états et les interfaces des trois solveurs.

Les solveurs sont dotés d'une méthode *getCurrentTime* permettant de récupérer le temps courant de simulation, et d'une méthode *doStep* permettant de résoudre l'équation pour un pas de temps h selon la méthode d'Euler. Les méthodes *doStep* sont décrites par l'Algorithme 3.

Algorithme 3 Méthodes *doStep* des solveurs s_x, s_y et s_z.

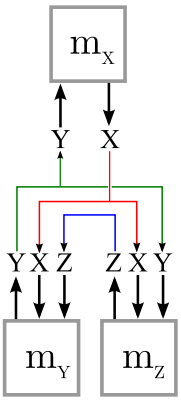
$s_x.doStep()$
- $x \leftarrow x + h * (\alpha(y - x))$ ▷ Evolution de la variable X
- $t \leftarrow t + h$ ▷ Evolution du temps de simulation courant
- $save(x, t)$ ▷ Sauvegarde de la valeur de x au temps t

$s_y.doStep()$
- $y \leftarrow y + h * (x(\rho - z) - y)$ ▷ Evolution de la variable Y
- $t \leftarrow t + h$ ▷ Evolution du temps de simulation courant
- $save(y, t)$ ▷ Sauvegarde de la valeur de y au temps t

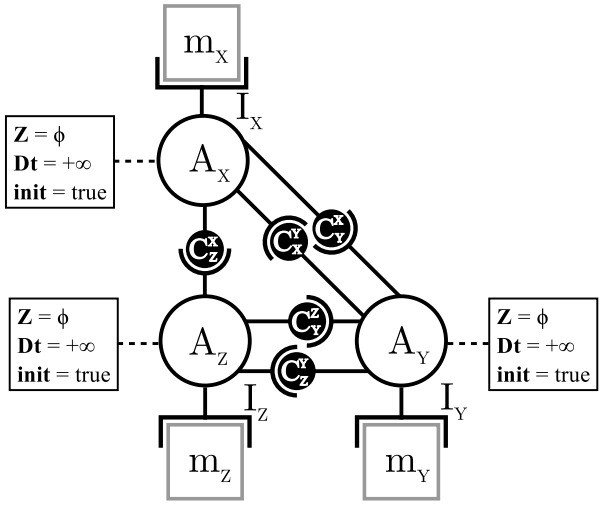
$s_z.doStep()$
- $z \leftarrow z + h * (xy - \beta z)$ ▷ Evolution de la variable Z
- $t \leftarrow t + h$ ▷ Evolution du temps de simulation courant
- $save(z, t)$ ▷ Sauvegarde de la valeur de z au temps t

Le système de Lorenz peut être simulé avec ces trois solveurs en suivant le graphe de dépendance de la Figure 8.4a. Ce graphe de dépendance permet d'assurer, tout au long de la simulation, la cohérence entre les copies locales des variables x, y et z.

Comme défini au Chapitre 6, pour simuler le système de Lorenz avec ces trois solveurs dans MECSYCO-Java, nous devons préalablement effectuer le wrapping des solveurs en DEVS. On pourra alors considérer chaque équation différentielle comme un modèle DEVS. Nous nous reposons pour cela sur la méthode proposée par [Zeigler et al., 2000] permettant l'intégration du



(a) Graphe de dépendance du système de Lorenz.



(b) Multi-modèle AA4MM du système de Lorenz

formalisme DESS (Differential Equation System Specification) dans DEVS par l'intermédiaire du formalisme DTSS (Discrete Time System Specification).

Cette méthode consiste à considérer le solveur d'une équation comme le modèle à pas de temps discret d'un intégrateur numérique. L'intégrateur peut ensuite être considéré comme un modèle DEVS comme suit : chaque pas d'exécution de l'intégrateur correspond à un événement interne. Les m-agents considèrent donc que leurs modèles planifient leurs événements internes de manière régulière tous les h pas de temps. Les événements externes échangés entre les modèles correspondent alors aux nouvelles valeurs des variables d'état des intégrateurs.

Dans cette optique, nous définissons les artefacts de modèle $\mathcal{I}_x$, $\mathcal{I}_y$ et $\mathcal{I}_z$ permettant de wrapper les solveurs s_x, s_y et s_z. Les méthodes de ces artefacts de modèle fonctionnent ainsi :

- *init*() initialise l'état et les paramètres du système de Lorenz.
- $processExternalEvent(e_{in_i}, t_i, x_i^k)$ fixe la valeur de la variable d'entrée du port x_i^k à la valeur de e_{in_i}.
- $processInternalEvent(t_i)$ appelle la méthode *doStep()* du solveur.
- $getOutputEvent(y_k^i)$ retourne la valeur de la variable d'état du port de sortie y_k^i.
- $getNextInternalEventTime()$ retourne *getCurrentTime()*$+h$.

Du graphe de dépendance de la Figure 8.4a, on déduit le multi-modèle AA4MM permettant de simuler en parallèle le système de Lorenz (voir Figure 8.4b et Table 8.2). Dans cette configuration, tous les m-agents vont effectuer la procédure de co-initialisation (voir Section 6.5.5) afin de démarrer la simulation dans un état global cohérent. De plus, comme tous les solveurs vont invariablement exécuter leurs événements internes tous les h pas de temps quels que soient les événements externes reçus, alors tous les m-agents ont un Dt_i (Définition 20) égal à $+\infty$.

Production et comparaison des résultats de simulation

Avant d'implémenter et de simuler nos modèles, il convient de noter que la représentation informatique des nombres réels va être fonction de l'environnement d'exécution et de compilation (e.g. l'architecture, le système d'exploitation, le compilateur, l'interpréteur). Les calculs effectués sur ces nombres réels pourront alors comporter des erreurs plus ou moins importantes selon l'environnement utilisé. La solution obtenue par simulation du système de Lorenz va donc différer en fonction de l'environnement utilisé pour compiler et exécuter les solveurs [Vicino et al., 2014].

Descriptions	Notations
Liens de sortie de m_x	$\mathsf{OUT}_x = \{(X,Y),(X,Z)\}$
Liens d'entrée de m_x	$\mathsf{IN}_x = \{(Y,Y)\}$
Liens de sortie de m_y	$\mathsf{OUT}_y = \{(Y,X),(Y,Z)\}$
Liens d'entrée de m_y	$\mathsf{IN}_y = \{(X,X),(Z,Z)\}$
Liens de sortie de m_z	$\mathsf{OUT}_z = \{(Z,Y)\}$
Liens d'entrée de m_z	$\mathsf{IN}_z = \{(X,X),(Y,Y)\}$
Liens de m_x à m_y	$\mathsf{L}_Y^X = \{(X,X)\}$
Liens de m_x à m_z	$\mathsf{L}_Z^X = \{(X,X)\}$
Liens de m_y à m_x	$\mathsf{L}_X^Y = \{(Y,Y)\}$
Liens de m_y à m_z	$\mathsf{L}_Z^Y = \{(Y,Y)\}$
Liens de m_z à m_y	$\mathsf{L}_Y^Z = \{(Z,Z)\}$

TABLE 8.2 – Graphe de dépendance du multi-modèle AA4MM de Lorenz.

Paramètres	Φ	h	x	y	z	α	ρ	β
Valeurs	100	0,01	1	1	4	10	28	2,67

TABLE 8.3 – Paramètres de la simulation du système de Lorenz.

Dans le cadre de notre preuve de concept, l'important pour pouvoir comparer les résultats de simulation est de s'assurer que le solveur séquentiel et les solveurs parallèles vont tous faire les mêmes erreurs de calcul. C'est pourquoi, nous avons compilé et exécuté les simulateurs et l'intergiciel MECSYCO-Java avec le même environnement.

Les simulations séquentielles et parallèles ont été effectuées pour les paramètres initiaux de la Table 8.3. Les résultats de la simulation du système de Lorenz correspondent aux valeurs successives prises par chacune des variables x,y et z à chaque pas de temps. La Figure 8.5 montre dans l'espace des phases, la solution obtenue avec la simulation séquentielle.

Afin de vérifier que MECSYCO-Java donne systématiquement la même solution des équations de Lorenz, nous enregistrons et comparons les résultats de 10000 simulations parallèles. Pour comparer les résultats de simulation, nous calculons à chaque pas de temps, la différence entre les valeurs de chaque variable x, y et z dans la simulation séquentielle et dans la simulation parallèle. **Pour les 10000 simulations MECSYCO-Java qui ont été lancées, nous trouvons que les résultats de simulation sont identiques à la simulation séquentielle. Cela montre que la simulation parallèle MECSYCO-Java du système de Lorenz donne les mêmes résultats que la simulation séquentielle. La simulation MECSYCO-Java du système de Lorenz respecte donc la contrainte de causalité.**

8.3 Intégration de l'hétérogénéité d'un multi-modèle

8.3.1 Caractéristiques générales du multi-modèle

Dans cette section nous illustrons, à travers une preuve de concept (préalablement publiée dans [Camus et al., 2015a] et inspirée de la modélisation hybride du trafic de [El Hmam et al., 2008]), la capacité de MECSYCO à intégrer l'hétérogénéité d'un multi-modèle au niveau des représentations, des formalismes et des logiciels de simulation. Le multi-modèle que nous simulons avec MECSYCO a les propriétés suivantes :

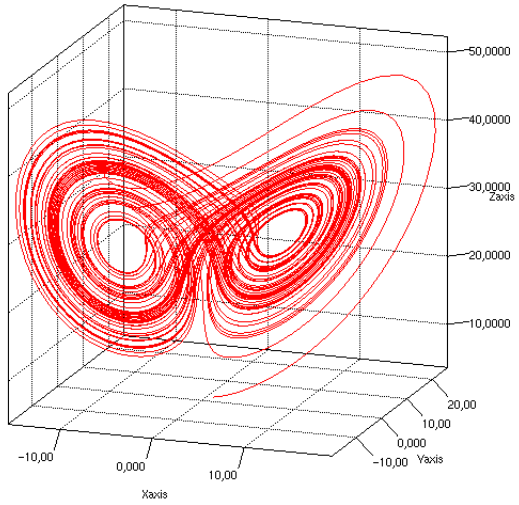


FIGURE 8.5 – Solution du système de Lorenz obtenue avec la simulation séquentielle pour les paramètres de la Table 8.3

— Il représente un système cible avec une vision multi-niveau de type agrégation/désagrégation (voir Chapitre 3). Des modèles avec des représentations micro et macro du système cohabitent donc dans le multi-modèle.
— Les modèles qui le composent sont écrits dans des formalismes différents (équationnel, événementiel et multi-agent).
— Les modèles qui le composent sont implémentés dans des logiciels de simulation différents (NetLogo et ad-hoc).

Nous retrouvons donc dans cette preuve de concept l'ensemble des problématiques relatives à l'hétérogénéité d'un multi-modèle, à savoir : Comment passer d'une représentation micro à une représentation macro du système et vice versa ? Comment intégrer formellement des modèles équationnels, multi-agents et événementiels dans un tout cohérent ? Comment coordonner les simulations de ces modèles alors qu'ils ont des politiques d'exécution événementielle et à pas de temps cyclique ? Comment gérer les échanges de données entre des logiciels de simulation qui ne sont pas faits pour communiquer ?

Dans cette preuve de concept, nous souhaitons modéliser le trafic autoroutier sur la partie d'autoroute de la Figure 8.6. Cette autoroute, que nous considérons dans un espace torique, peut être décomposée en trois tronçons. Dans chacun de ces tronçons, le nombre de voies de circulation disponibles est différent, ce qui créer des changements dans le comportement du trafic. Chacun de ces tronçons est alors représenté par un modèle différent :

— Dans le tronçon 1, trois voies de circulation sont disponibles et la vitesse est limitée à 100km/h. Le trafic peut ici être considéré comme un flux, et est alors représenté par le modèle équationnel m_1. Nous présentons ce modèle dans la Section 8.3.2.
— Dans le tronçon 2, deux voies de circulation sont disponibles et la vitesse est limitée à 90 km/h. **Ce tronçon est représenté par le modèle événementiel m_2 que nous avons utilisé pour représenter l'échangeur autoroutier dans la Section 8.2.1**.
— Dans le tronçon 3, une seule voie de circulation est disponible et la vitesse est limitée à 100 km/h. Des congestions peuvent alors apparaître, et un modèle multi-agent m_3 est utilisé pour représenter ces phénomènes. Ce modèle est décrit en Section 8.3.3.

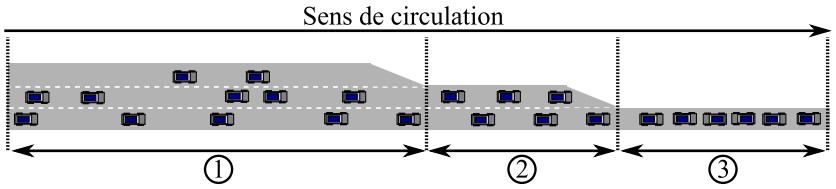


FIGURE 8.6 – Les différents tronçons de l'autoroute (Projection plane de l'espace torique)

8.3.2 Le modèle équationnel m_1

Le modèle équationnel m_1 correspond au système d'équations aux dérivées partielles (EDP) utilisées dans [El Hmam et al., 2008]. Ces équations décrivent l'évolution du trafic de manière macroscopique grâce à des variables agrégées. Le trafic est alors représenté comme un flux ayant une densité, un débit et une vitesse moyenne évoluant dans un temps et un espace continus.

Pour permettre la résolution numérique du modèle, l'espace modélisé par les EDP (i.e. le tronçon d'auroroute) est discrétisé en un ensemble $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ de cellules, le trafic allant de c_1 vers c_n. Les EDPs sont donc transformées en équations différentielles ordinaires. Un solveur va alors permettre de décrire l'évolution de chaque cellule c_i en discrétisant le temps. L'évolution de chaque cellule à chaque pas de temps est décrite par les équations aux différences suivantes [El Hmam et al., 2008] :

$$\rho_i(k+1) = \rho_i(k) + \frac{T}{L_i.\lambda}[q_{i-1}(k) - q_i(k)] \tag{8.2}$$

$$q_i(k) = \rho_i(k).v_i(k).\lambda \tag{8.3}$$

$$v_i(k+1) = v_i(k) + \frac{T}{\tau}[V(\rho_i(k)) - v_i(k)] + \frac{T}{L_i}v_i(k)[v_{i-1}(k) - v_i(k)] - \frac{\nu T[\rho_{i+1}(k) - \rho_i(k)]}{\tau L_i[\rho_i(k) + \kappa]} \tag{8.4}$$

$$V(\rho_i(k)) = v_f.\mathsf{exp}\left[-\frac{1}{\psi}\left(\frac{\rho_i(k)}{\rho_{cr}}\right)^{\psi}\right] \tag{8.5}$$

$$\psi = -\frac{1}{\mathsf{ln}\left(\frac{capacite}{v_f.\rho_{cr}}\right)} \tag{8.6}$$

avec

L_i, la longueur en km de la cellule c_i,
T, la durée d'un pas de temps en h,
k, le nombre entier de pas de temps de simulation,
$\rho_i(k)$, la densité en $veh/km/voie$ de c_i à l'instant $k.T$,
$q_i(k)$, le débit en veh/h de c_i à l'instant $k.T$,
$v_i(k)$, la vitesse en km/h de c_i à l'instant $k.T$,
$capacite$, le débit maximal en veh/h de $\mathcal{C}$,
τ, κ, ν, des paramètres globaux de $\mathcal{C}$,
λ, le nombres de voies de $\mathcal{C}$,
$\rho_c r$ la densité critique en $veh/km/voie$ de $\mathcal{C}$, et
v_f la limite de vitesse.

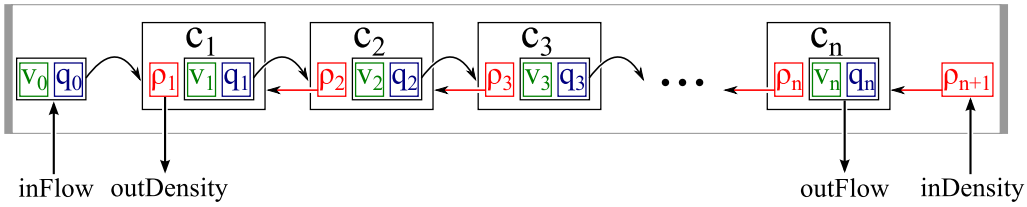


FIGURE 8.7 – Model m_1

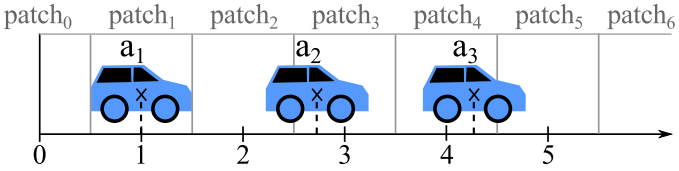


FIGURE 8.8 – Discrétisation de l'espace en patch dans le modèle m_3

A chaque pas de temps T, le solveur va donc faire évoluer la densité ρ_i, le débit q_i, et la vitesse v_i de chaque cellule $c_i \in \mathcal{C}$ en fonction (1) de la densité ρ_{i+1} de la cellule c_{i+1}, ainsi que (2) de la vitesse v_{i-1} et du débit q_{i-1} de la cellule c_{i-1}. Pour calculer les états des cellules situées aux extrémités du tronçon (i.e c_1 et c_n,), le solveur considère les variables q_0, v_0 et ρ_{n+1}. Le modèle dispose de deux ports d'entrée permettant de modifier ces variables :

— *inFlow* permet de recevoir les couples (v, q) correspondant aux nouvelles valeurs des variables v_0 et q_0.

— *inDensity* permet de mettre à jour la nouvelle valeur de ρ_{n+1}.

La Figure 8.7 résume le fonctionnement du modèle.

8.3.3 Le modèle multi-agent m_3

Le modèle multi-agent m_1 correspond au modèle de [Wilensky, 1997a] implémenté dans le simulateur NetLogo [Wilensky, 1999]. Ce modèle a une vision microscopique de l'autoroute car il représente chaque voiture individuellement comme un agent autonome (que nous appellerons dans la suite agent-voiture). L'environnement dans lequel ces agents évoluent représente le tronçon 3 de l'autoroute. Ce tronçon ne comportant qu'une seule voie de circulation, l'environnement correspond à un espace à une dimension discrétisée en un ensemble de cellules appelées patchs, chacun de ces patchs ayant une coordonnée $x \in \mathbb{N}$ (voir Figure 8.8).

Nous notons A l'ensemble des agents-voitures contenus dans l'environnement. Chaque agent-voiture $a_i \in A$ est défini par un identifiant id_i, une position $x_i \in \mathbb{R}$, et une vitesse $v_i \in \mathbb{R}$ (en patchs par pas de temps). Le comportement de chaque agent-voiture suit deux règles :

— si un autre agent-voiture est situé juste devant lui, il se cale sur sa vitesse et décélère, éventuellement jusqu'à l'arrêt complet. Le paramètre *deceleration* détermine le taux de décélération d'une voiture.

— si aucun autre agent-voiture n'est situé juste devant lui, l'agent-voiture accélère jusqu'à atteindre la limite de vitesse de 1 patch par unité de temps. Le paramètre *acceleration* détermine le taux d'accélération d'une voiture.

Ce modèle est simulé par pas de temps cyclique : à chaque pas de temps, tous les agents-voitures se déplacent dans l'environnement (en se basant sur une copie de l'environnement comme expliqué précédemment en Section 2.6.3). Le simulateur dispose d'une méthode *go()* permettant d'exécuter la simulation pour un pas de temps. Le comportement d'un agent-voiture a_i à chaque

pas de temps correspond alors à l'algorithme 4.

Algorithme 4 comportement d'un agent a_i pendant un pas de temps

▷ S'il y a un agent sur le patch juste devant a_i
if $(\exists a_j \in A, arrondirAuxEntiers(x_j) = arrondirAuxEntiers(x_i) + 1)$ **then**
  $v_i \leftarrow v_j$ ▷ a_i cale sa vitesse sur celle de a_j.
  $v_i \leftarrow min(0, v_i - deceleration)$ ▷ a_i décélère et s'arrête éventuellement.
else
  $v_i \leftarrow min(1, v_i + acceleration)$ ▷ Sinon, a_i accélère sans dépasser la vitesse maximale.
end if
$x_i \leftarrow x_i + v_i$ ▷ a_i avance en fonction de sa vitesse.

L'interface du modèle est constitué d'un port d'entrée in_a permettant d'ajouter une liste d'agents-voitures dans le tronçon, et d'un port de sortie out_s permettant de récupérer les états de tous les agents-voitures contenus dans le tronçon.

8.3.4 Construction du multi-modèle

Graphe de dépendance

Pour construire le multi-modèle de l'autoroute à partir des modèles m_1, m_2 et m_3, nous souhaitons connecter les modèles comme suit (Figure 8.9) :

— Le débit sortant de m_1 à chaque pas de temps (disponible dans le port $outFlow$) détermine les véhicules entrant dans m_2 (i.e. détermine les événements externes entrant par le port x_2^1).
— Les voitures sortant de m_2 (i.e. les événements externes sortant du port y_2^1) correspondent aux agents-voitures entrant dans m_3.
— Comme nous considérons que l'autoroute est sur un espace torique, l'état du modèle m_3 (disponible dans le port out_s) est utilisé à chaque pas de temps pour déterminer le flux d'entrée du modèle m_1 (entrant par le port $inFlow$).

Nous pouvons noter ici que nous ne nous servons pas des ports $inDensity$ et $outDensity$. En effet, contrairement au modèle hybride de [El Hmam et al., 2008], nous ne faisons pas remonter, en amont du flux de trafic, d'information sur la concentration à l'entrée de chaque tronçon. Cette information, pourtant utile à m_1 pour déterminer avec précision son débit sortant, ne peut toutefois pas être obtenue à partir de m_2. Le modèle événementiel représente en effet uniquement les entrées et sorties de véhicules du tronçon, et non leurs déplacements spatiaux à l'intérieur du tronçon. **Nous touchons du doigt ici une limite fondamentale des approches de multi-modélisation : il n'y a rien qui puisse être fait pour rendre compatibles deux modèles faisant des hypothèses ou des simplifications contradictoires** [Page et al., 2004, Tolk et al., 2007, Hofmann, 2005]. Dans notre cas, cette contradiction cause une imprécision que l'on peut considérer comme négligeable : le trafic dans les tronçons 1 et 2 étant considéré comme fluide, la concentration moyenne à l'entrée de ces tronçons peut être considérée comme quasi nulle.

Le multi-modèle AA4MM correspondant à cette configuration est décrit par la Figure 8.10 et la Table 8.4. Ce multi-modèle permet de simuler l'évolution du trafic autoroutier pour une durée σ. Les modèles m_1 et m_3 étant à pas de temps fixe, alors $Dt_1 = Dt_3 = +\infty$. Comme pour le multi-modèle de l'échangeur autoroutier, $Dt_2 = l_2/v_{max_2}$. Les modèles n'ayant pas besoin de s'échanger des informations pour démarrer la simulation dans un état global cohérent, nous n'effectuons pas ici la procédure d'initialisation.

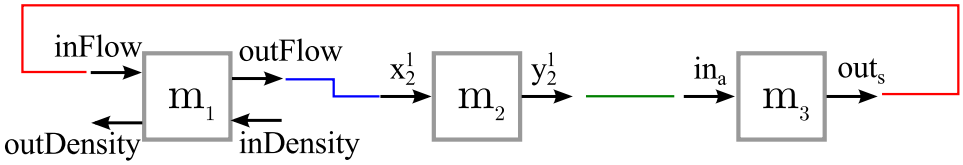


FIGURE 8.9 – Graphe de dépendance du multi-modèle de l'autoroute.

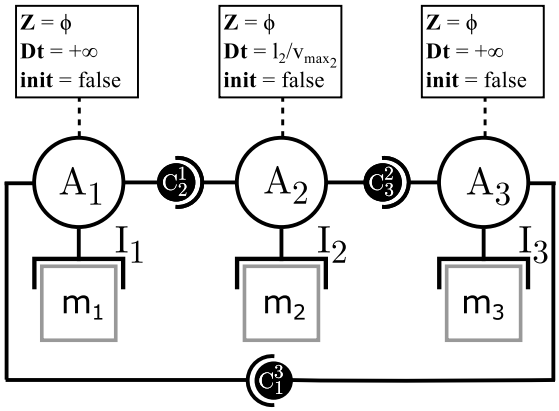


FIGURE 8.10 – Multi-modèle AA4MM de l'autoroute.

Interopérabilité des logiciels de simulation

La gestion de l'interopérabilité des logiciels de simulation des modèles m_1 et m_2 avec MECSYCO-Java est triviale car ces modèles sont également implémentés dans le langage Java. Les artefacts de modèles $\mathcal{I}_1$ et $\mathcal{I}_2$ peuvent alors directement interagir avec les objets Java des simulateurs. Pour contrôler m_3, l'artefact de modèle $\mathcal{I}_3$ utilise l'API Java de NetLogo [2].

Intégration des formalismes

Pour ce multi-modèle l'artefact de modèle $\mathcal{I}_2$ est identique à celui du multi-modèle de l'échangeur autoroutier (voir Section 8.2.1).

Le modèle m_1 peut être intégré dans DEVS de la même manière que pour les équations du système de Lorenz (voir Section 8.2.2). La spécification de l'artefact de modèle $\mathcal{I}_1$ est alors identique à celle des artefacts de modèle du multi-modèle de Lorenz.

Le modèle m_3 étant un modèle à pas de temps discret, nous nous basons sur le mapping du formalisme DTSS en DEVS établi dans [Zeigler et al., 2000] pour définir les primitives de l'artefact de modèle $\mathcal{I}_3$. Ces fonctions sont alors définies ainsi :

- $init()$ lance le logiciel NetLogo et initialise le modèle m_1.
- $processExternalEvent(e_{in_i}, t_i, x_i^k)$ ajoute les agents contenus dans e_{in_i} dans l'environnement.
- $processInternalEvent(t_i)$ appelle la méthode *go()* du simulateur.
- $getOutputEvent(y_k^i)$ retourne l'ensemble des agents sortant du modèle.
- $getNextInternalEventTime()$ retourne le temps courant du modèle$+h_1$.

2. `http://ccl.northwestern.edu/netlogo/4.1/docs/javadoc/index.html`

Descriptions	Notations
Liens de sortie de m_1	$\mathsf{OUT}_1 = \{(outFlow, 2)\}$
Liens d'entrée de m_1	$\mathsf{IN}_1 = \{(3, inFlow)\}$
Liens de sortie de m_2	$\mathsf{OUT}_2 = \{(y_2^1, 3), \}$
Liens d'entrée de m_2	$\mathsf{IN}_2 = \{(1, x_2^1)\}$
Liens de sortie de m_3	$\mathsf{OUT}_3 = \{(out_s, 1)\}$
Liens d'entrée de m_3	$\mathsf{IN}_3 = \{(2, in_a)\}$
Liens de m_1 à m_2	$\mathsf{L}_2^1 = \{(outFlow, x_2^1)\}$
Liens de m_2 à m_3	$\mathsf{L}_3^2 = \{(y_2^1, in_a)\}$
Liens de m_3 à m_1	$\mathsf{L}_1^3 = \{(out_s, inFlow)\}$

TABLE 8.4 – Graphe de dépendance du multi-modèle AA4MM de l'autoroute.

Intégration des représentations

Maintenant que notre multi-modèle formel est homogénéisé grâce à DEVS, nous pouvons, grâce aux opérations de transformation, faire le pont entre les différentes représentations des modèles. Les problèmes que ces opérations doivent résoudre sont : comment intégrer dans un tout cohérent des modèles décrivant l'évolution du trafic autoroutier avec des échelles temporelles et spatiales différentes ? Comment assurer le passage d'une représentation micro à une représentation macro de l'autoroute et vice versa ?

Pour répondre à ces questions, nous devons tout d'abord nous arrêter sur la représentation particulière du modèle multi-agent m_3. En effet, ce modèle ne fournit pas, tel quel, d'information sur la représentation réelle des unités spatiales (i.e. les coordonnées des voitures) et temporelles (i.e. la durée d'un pas de temps). Nous donnons donc ici une estimation de la représentation de m_3, afin de pouvoir l'intégrer avec celles, plus précises, des modèles m_1 et m_2.

Dans ce modèle, la taille d'un agent est équivalente à celle d'une unité spatiale (i.e. d'un patch) (cf. Figure 8.8). Soit l_v la longueur moyenne d'une voiture (convertie en km), chaque agent a_i du modèle représente alors une voiture située à une distance de $x_i.l_v$ kilomètres de l'entrée du tronçon 3.

Pour estimer la durée d'un pas de temps, nous faisons l'hypothèse que la limite de vitesse de 100 km/h du tronçon correspond à la vitesse maximale d'un agent dans le modèle, soit 1 patch par pas de temps (i.e. nous faisons l'hypothèse que les agents respectent la limitation de vitesse). La durée d'un pas de temps est alors de $l_v/100$. Dans le multi-modèle, nous estimons l_v à 4 mètres. Un pas de temps correspond alors à $0,00004$ heures (i.e. environ un dixième de seconde).

Maintenant que nous avons clarifié la représentation de m_3, nous pouvons gérer l'intégration des représentations des modèles.

L'intégration des différentes représentations temporelles est résolue par l'ajout d'une opération temporelle à $\mathcal{C}_1^3$ et $\mathcal{C}_3^2$. En effet, m_1 et m_2 s'expriment tous les deux en heures. En revanche, m_3 considère uniquement le nombre de pas de temps écoulés depuis le début de la simulation. Il faut alors traduire les informations temporelles envoyées et reçues par m_3. Dans cette optique, l'opération temporelle de $\mathcal{C}_1^3$ permet de multiplier par $l_v/100$ les informations temporelles envoyées par m_3. De la même manière, l'opération temporelle de $\mathcal{C}_3^2$ permet de diviser par $l_v/100$ les informations temporelles envoyées par m_3.

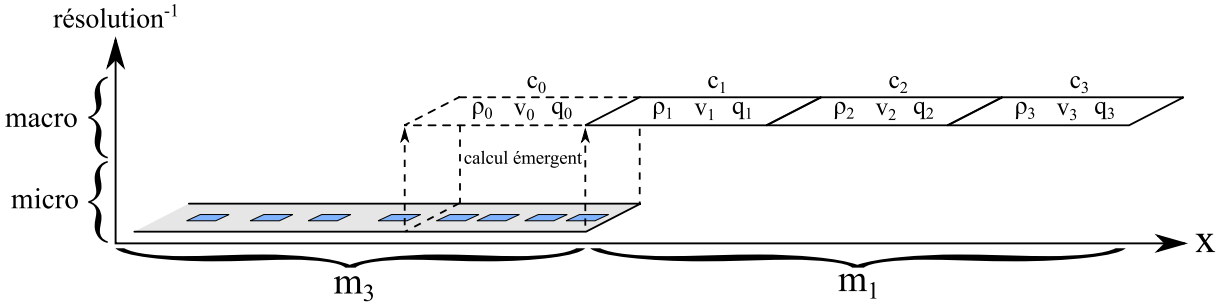


FIGURE 8.11 – Utilisation du calcul émergeant pour relier les représentations des modèles m_3 et m_1.

L'intégration des différentes représentations spatiales est résolue par l'ajout d'une opération événementielle à $\mathcal{C}_3^2$. En effet, contrairement à m_1, m_2 ne représente pas l'espace à l'intérieur de son tronçon. La conséquence est qu'un véhicule sortant de m_2 correspond au couple (id, v_{id}) alors qu'un agent entrant dans m_3 est représenté par le tuple (id_i, v_i, x_i). Nous devons alors déterminer la coordonnée x_i des véhicules sortant de m_2. Les véhicules entrants dans m_3 au point de coordonnée 0, l'opération événementielle de $\mathcal{C}_3^2$ associe la coordonnée 0 à chaque voiture.

L'intégration des différents niveaux de résolution est effectuée en ajoutant une opération événementielle à $\mathcal{C}_2^1$ et à $\mathcal{C}_1^3$. En effet, m_2 et m_3 ont des représentations microscopiques du trafic car ils considèrent chaque voiture individuellement. A l'opposé, m_1 a une représentation macroscopique du trafic car il considère ce dernier de manière agrégée. Nous devons alors assurer le passage d'un niveau de résolution à un autre.

Pour permettre les échanges de données de m_1 vers m_2, nous devons passer d'un flux de sortie (i.e. un débit en véhicules par heure, et une vitesse moyenne) à un ensemble de véhicules, c'est à dire passer d'une représentation macroscopique à une représentation microscopique. Pour ce faire, l'opération événementielle de $\mathcal{C}_2^1$ calcule à partir d'un débit d généré par m_1, le nombre n_v de véhicules comme étant égal à $d.T$, T correspondant à la durée d'un pas de temps du solveur macroscopique. L'opération retourne alors un événement contenant n_v tuples (id, v_{id}) avec v_{id} égale à la vitesse moyenne du flux.

Pour permettre les échanges de données de m_3 vers m_1, nous devons passer d'un ensemble agents/véhicules (i.e. l'état de m_3 envoyé à chaque pas de temps via le port out_s) à un flux d'entrée (i.e. un débit en véhicules par heure, et une vitesse moyenne). Autrement dit, nous devons passer d'une représentation microscopique à une représentation macroscopique du trafic. En reprenant le principe de calcul émergeant proposé par [Duboz et al., 2003], nous faisons l'hypothèse que la simulation du modèle multi-agent m_3 fait émerger le calcul de l'évolution de la densité moyenne ρ_0 et de la vitesse moyenne v_0 d'une cellule macroscopique c_0 (Figure 8.11). En reprenant les équations du modèle m_1, nous pouvons alors déterminer le débit $q_0(t)$ au pas de temps microscopique t, comme étant égal à $\rho_0(t).v_0(t)$. La cellule c_0 émulée par m_3 va alors pouvoir être connectée (via MECSYCO) avec la cellule c_1 du modèle macroscopique m_1.

Dans cette optique, l'opération événementielle de l'artefact $\mathcal{C}_1^3$ assure le passage de l'état de m_3 à l'état de c_0. Pour déterminer ρ_0 et v_0, cette opération considère à la sortie du tronçon 3, une zone de taille équivalente à celle d'une cellule macroscopique (i.e. de longueur L_i). L'opération sélectionne alors tous les agents de m_3 se trouvant dans cette zone, c'est à dire tous les agents a_i ayant $l_v.x_i \geq L_3 - L_i$, L_3 correspondant à la longueur (en km) du tronçon 3. Soit N_a le

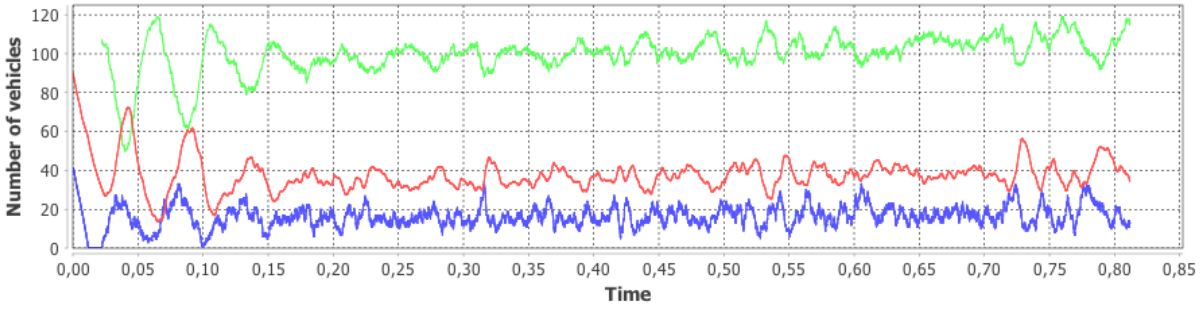


FIGURE 8.12 – Évolution du nombre de véhicules dans les tronçons 1 (courbe rouge), 2 (courbe verte) et 3 (courbe bleue) en fonction du temps de simulation (en heures).

nombre d'agents satisfaisant cette condition, ρ_0 correspond alors à N_a/L_i. Enfin, le calcul de v_0 correspond à la moyenne des vitesses des agents se trouvant dans la zone. L'événement à destination de m_1 retourné par l'opération correspond alors au couple (v_0, q_0).

Une fois ces opérations de transformation définies et ajoutées aux artefacts de couplage, l'intégration des différentes représentations du système dans le multi-modèle est complète. Le multi-modèle AA4MM de l'autoroute est alors terminé, et a été simulé avec succès. La Figure 8.12 montre les résultats de cette simulation.

Bilan

Dans cette preuve de concept, nous avons montré comment les artefacts de modèles permettent à la fois de gérer l'intéropérabilité des différents logiciels de simulation, et l'intégration des modèles formels au formalisme DEVS. Nous avons également montré comment les opérations des artefacts de couplage permettent de faire le pont entre plusieurs représentations spatiales et temporelles entre les modèles. Une fois ces composants spécifiés et implémentés, nous avons pu simuler le multi-modèle avec MECSYCO-Java.

8.4 Conclusion

Dans ce Chapitre, nous avons présenté plusieurs preuves de concepts qui nous ont permis d'évaluer nos propositions des Chapitres 6 et 7. Pour réaliser ces preuves de concepts nous avons suivi à chaque fois la même procédure, à savoir (1) décrire les multi-modèles à partir de l'environnement graphique de multi-modélisation de AA4MM, puis (2) générer automatiquement les co-simulations MECSYCO-Java correspondantes.

Nous avons tout d'abord montré à travers deux preuves de concepts comment nous avons vérifié que MECSYCO-Java permettait de simuler un modèle couplé DEVS et un système d'équations différentielles en respectant la contrainte de causalité et sans se retrouver en situation d'interblocage. Nous avons ensuite montré comment MECSYCO permettait de construire et simuler le multi-modèle multi-niveau d'une autoroute. A travers cet exemple, nous avons illustré le rôle de chaque concept de MECSYCO dans l'intégration de l'hétérogénéité d'un multi-modèle.

Grâce à l'environnement de multi-modélisation, nous avons pu décrire et générer automatiquement le code de simulation de tous ces multi-modèles. Mis à part pour l'implémentation des concepts du niveau domaine (i.e. les artefacts de modèle et les opérations de transformation), nous n'avons pas eu à produire manuellement de code pour passer d'un multi-modèle AA4MM à

sa simulation MECSYCO-Java. **Le passage des multi-modèles à leurs simulations a alors été fait automatiquement sans introduire d'erreur d'implémentation.**

En ce qui concerne l'évaluation de notre approche, ces preuves de concept, bien que conçues dans une volonté de se rapprocher de contraintes applicatives réelles, ne peuvent toutefois se substituer pleinement à un cas d'étude concret. C'est pourquoi, nous décrivons dans le chapitre suivant comment notre approche a été appliquée avec succès pour l'étude des smart grids par les membres du projet MS4SG dans le cadre d'un partenariat entre Inria et EDF R&D.

Chapitre 9

Applications

Sommaire

9.1 Introduction

Dans ce chapitre, nous détaillons comment notre approche a été utilisé avec succès par les membres du projet MS4SG [3] (que l'auteur a assisté) pour simuler un smart grid appelé Concept-Grid. L'objectif de ce chapitre est triple :

Il permet tout d'abord d'évaluer les capacités de MECSYCO dans un cadre applicatif concret et représentatif des défis de la multi-modélisation. En effet, le multi-modèle Concept-Grid mobilise des modèles métiers pré-existants et hétérogènes, qui sont implémentés dans différents logiciels de simulation faisant autorité dans les domaines des réseaux électriques et des télécommunications. Nous montrons alors comment MECSYCO permet d'intégrer ces composants hétérogènes dans un tout cohérent.

Nous montrons également quelles sont les étapes du processus de multi-modélisation qui ont nécessité le plus de temps et poser le plus de difficultés. Concept-Grid nous fournit en effet un retour d'expérience sur un cas concret de multi-modélisation avec notre approche.

3. Multi-Simulation for Smart Grids issu d'un partenariat entre EDF R&D et Inria

Enfin, nous montrons comment la richesse du paradigme multi-agent de AA4MM peut être exploitée pour faciliter davantage l'expérimentation sur les multi-modèles. Ainsi, au travers de Concept-Grid, nous présentons certaines des extensions de notre approche mises au point par les membres du projet MS4SG.

Ce chapitre, qui reprend en partie l'article [Vaubourg et al., 2015b], est organisé comme suit. La Section 9.2 présente le multi-modèle Concept-Grid. La Section 9.3 détaille la mise en place du multi-modèle, les problèmes rencontrés et les solutions apportées en utilisant notre approche. La Section 9.4 présente les extensions qui ont pu être apportées à notre approche pour faciliter la M&S de Concept-Grid.

9.2 Le multi-modèle de Concept-Grid

Le système de Concept-Grid (Figure 9.1) comprend cinq maisonnettes, chacune équipée d'un compteur intelligent (Linky) raccordé à un dispositif thermique de type pompe à chaleur (PAC) : on peut connecter/déconnecter la PAC en envoyant un ordre au compteur à partir d'un centre décisionnel appelé FAE (Fonction Avancée d'Effacement). La communication entre le FAE et les compteurs se fait par un réseau Ethernet puis par courant porteur en ligne (CPL). Le scénario consiste à effacer (i.e. déconnecter du réseau électrique) les pompes à chaleur des maisonnettes une à une en suivant un cycle d'effacement et à vérifier à partir de remontées d'informations que l'effacement permet effectivement de limiter la consommation.

Concept-Grid [Vaubourg et al., 2015b, Vaubourg et al., 2015c] est un cas représentatif des défis de la multi-modélisation car il combine simultanément plusieurs types d'hétérogénéité : trois domaines d'expertise interviennent ; plusieurs types de simulateurs doivent interagir ; ces simulateurs s'appuient sur des formalismes différents (équationnel ou événementiel) ; ils utilisent des échelles temporelles différentes. De plus, des contraintes logicielles et matérielles doivent être respectées en termes de plateforme (Windows ou GNU/Linux) et de langage d'interfaçage (Java/C++). La Table 9.1 récapitule ces différents points.

La simulation de ce cas fait intervenir plusieurs simulateurs existants : le réseau de télécommunication est simulé avec le logiciel NS-3 qui intègre les communications Ethernet sous IPv4 et les communications CPL en IPv6 ; les composants du réseau électrique sont simulés sous forme de FMU (Functional Mockup Unit présentée au Chapitre 4) : une FMU simule le réseau de distribution et les PAC, cinq autres FMUs simulent les compteurs ; enfin, le système de décision est un automate écrit de manière ad-hoc en Java.

Les échanges d'informations entre simulateurs correspondent au graphe de dépendance de la Figure 9.2. Le multi-modèle AA4MM correspondant est celui de la Figure 9.3.

9.3 Mise en œuvre de notre contribution

Comme avec n'importe quel autre multi-modèle, la réalisation de Concept Grid a été le fruit d'une suite d'essais et d'erreurs. Nous présentons ici le résultat de ce processus itératif en donnant les artefacts de modèles et les opérations de transformation qui ont été nécessaires.

9.3.1 Intégration de l'hétérogénéité des représentations

Pour assurer l'intégration des différentes échelles temporelles utilisées, des opérations temporelles permettant de passer de la nanoseconde à la seconde et vice et versa, ont été ajoutées aux artefacts de couplage du multi-modèle. La mise en place de ces opérations est triviale car

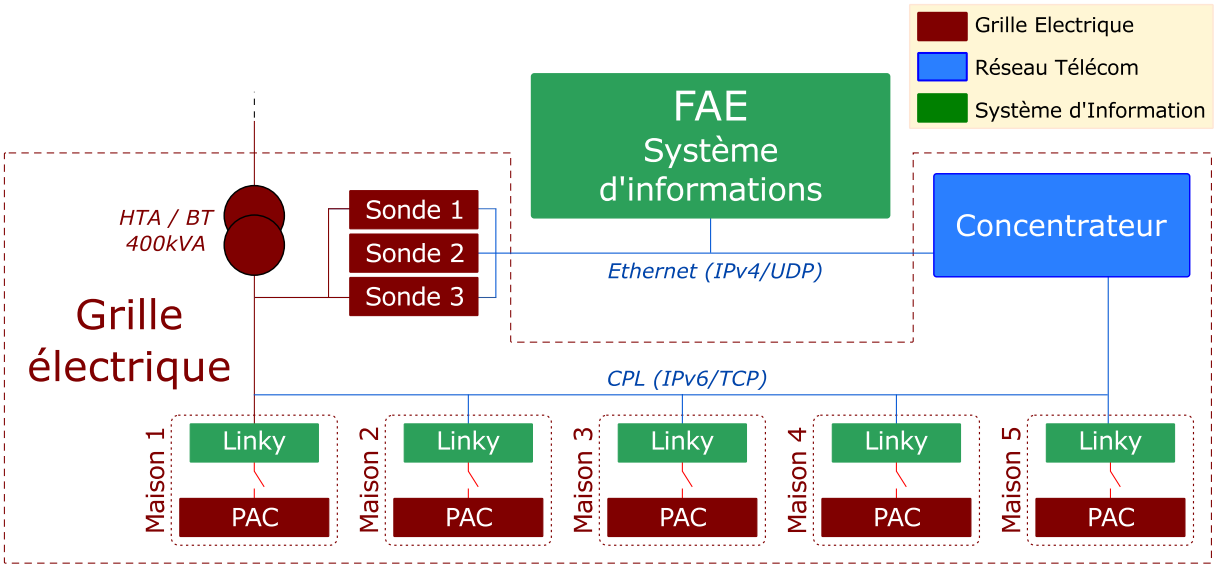


FIGURE 9.1 – Système modélisé par Concept-Grid (Source [Vaubourg et al., 2015c])

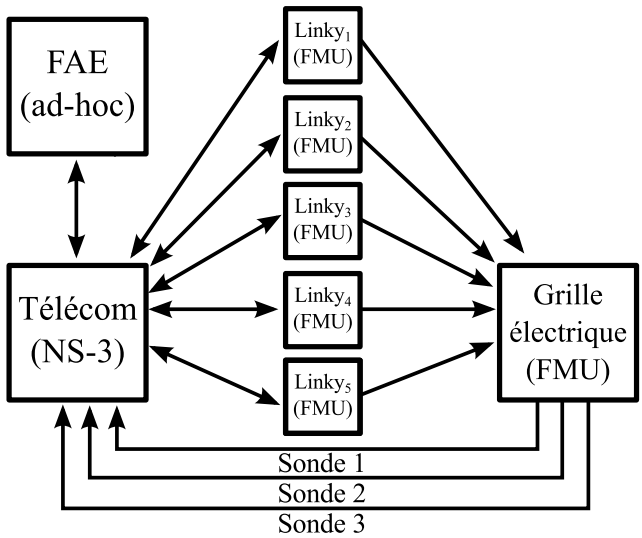


FIGURE 9.2 – Graphe de dépendance de Concept-Grid (Source [Vaubourg et al., 2015c])

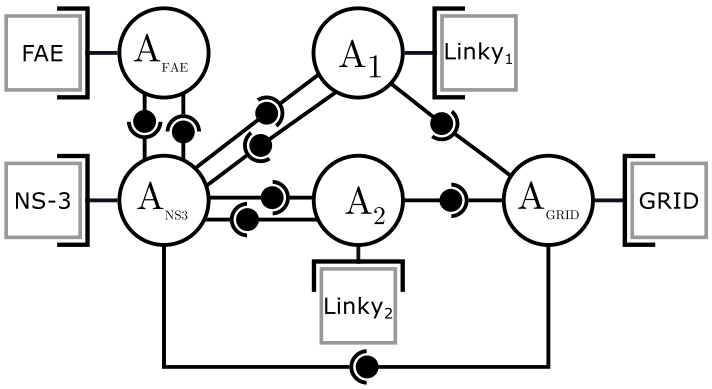


FIGURE 9.3 – Multi-modèle AA4MM de Concept-Grid (seulement deux modèles de maisonnettes sont représentés ici par souci de clarté du schéma) (Source [Vaubourg et al., 2015c])

Domaines	Composants	Simulateurs	Plate-formes	Langages	Formalismes	Echelles Temporelles
Télécom	TCP/IPv6/CPL	NS-3	GNU/Linux	C++	Evénement	Nanoseconde
	UDP/IPv4/Ethernet					
	Noeuds (FAE, Linkys, Sondes, Concentrateur)					
SI	FAE	Ad-hoc	Indifférent	Java	Automate	Seconde
	Linky	FMUs	Windows	Java	Equationnel	Seconde
Réseau électrique	PACs	FMU	Windows	Java	Equationnel	Seconde
	Transfo. HTA/BT					
	Lignes électriques					
	Charges (maisons)					
	Sondes					

TABLE 9.1 – Les Différents niveaux d'hétérogénéité du multi-modèle Concept Grid (Source [Vaubourg et al., 2015c])

ces dernières effectuent simplement des changements d'ordre de grandeur sur les informations temporelles échangées entre m-agents.

9.3.2 Intégration de l'hétérogénéité formelle

L'intégration formelle des modèles de Concept-Grid a été réalisée en définissant les artefacts de modèles suivants :

Pour intégrer le formalisme équationnel du standard FMI, l'artefact de modèle FMU est défini comme pour les modèles équationnels du chapitre précédent, en se basant sur l'intégration du formalisme DESS en DEVS via le formalisme DTSS. FMI étant un standard d'interopérabilité, les solveurs des FMU sont *de facto* contrôlables depuis l'extérieur. L'interfaçage d'un solveur FMU avec l'artefact de modèles a alors été très aisé.

La définition de l'artefact de modèle intégrant l'automate à états finis temporels du modèle de FAE est triviale, car les automates à états finis temporels sont très proches du formalisme DEVS. De plus, l'interfaçage du simulateur abstrait de la FAE avec l'artefact de modèle a été très aise car le modèle est défini de manière ad-hoc.

L'artefact de modèle gérant l'intégration formelle de NS-3 [Vaubourg et al., 2015a] considère un modèle NS-3 comme un modèle couplé DEVS. En effet, un modèle NS-3 correspond à un ensemble de composants événementiels (i.e. des modèles de liens et de noeuds réseaux) interconnectés, chacun de ces composants pouvant être considérés comme un modèle atomique DEVS.

Cet artefact de modèle a été le plus complexe à mettre en place à cause du formalisme de modélisation de NS-3 d'une part, et de son simulateur abstrait d'autre part. En effet, NS-3 ne permet pas de définir des interfaces pour ses modèles car il décrit des systèmes fermés uniquement. Le modèle NS-3, même traduit en DEVS, ne pourra alors pas être couplé avec l'extérieur. Pour répondre à ce problème, [Vaubourg et al., 2015a] ont créé une bibliothèque NS-3 permettant de définir dans un modèle réseau, l'équivalent des couplages externes d'entrée et de sortie d'un modèle couplé DEVS. La bibliothèque permet alors d'ouvrir le système en définissant des ports d'entrée/sortie localisés dans la topologie réseau.

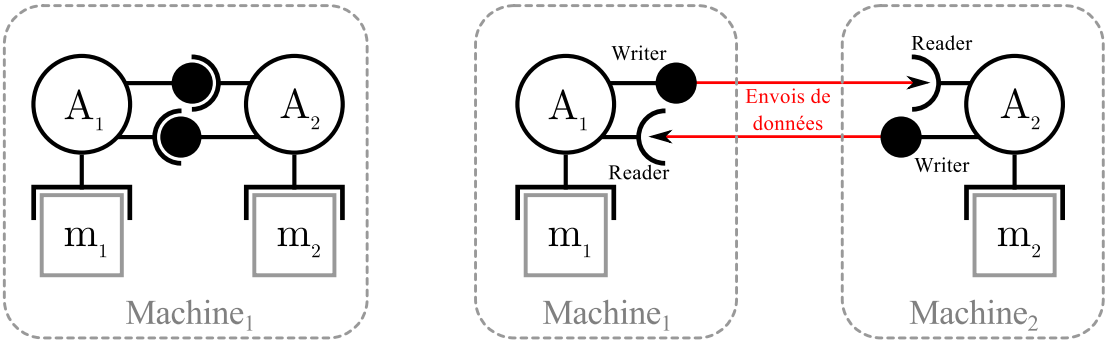
De son côté, le simulateur abstrait de NS-3 ne peut pas être contrôlé depuis l'extérieur : une fois lancé, il effectue une simulation d'un seul trait. Le simulateur ne peut pas alors être interfacé avec un artefact de modèle. Pour résoudre ce problème, la bibliothèque NS-3 de [Vaubourg et al., 2015a] propose un simulateur abstrait respectant le protocole de simulation DEVS.

L'artefact de modèle NS-3 s'appuie alors sur la bibliothèque créée pour permettre le wrapping DEVS du simulateur de réseaux télécom.

9.4 Extensions de l'approche

9.4.1 Intégration de l'hétérogénéité logicielle : extension de l'implémentation

Pour faciliter la gestion de l'interopérabilité des logiciels de simulation de Concept-Grid, les membres du projet MS4SG utilisent conjointement deux implémentations de MECSYCO : (1) celle en Java présentée précédemment, et (2) une implémentation en C++ réalisée par leurs soins. L'utilisation de ces deux implémentations facilite la création des artefacts de modèle qui peuvent alors être écrits directement dans le même langage de programmation que le logiciel



(a) Exécution mono-machine (b) Exécution distribuée

FIGURE 9.4 – Distribution d'un multi-modèle AA4MM avec MECSYCO

qu'ils interfacent. **Nous voyons là un avantage des spécifications opérationnelles de MECSYCO : elles font abstraction du langage de programmation qui peut donc être choisi en fonction des contraintes applicatives.**

Pour faire cohabiter les deux implémentations de MECSYCO, et pour pouvoir utiliser simultanément les différents systèmes d'exploitation nécessaires à la simulation du multi-modèle, **l'implémentation de MECSYCO a pu être étendue pour permettre une simulation non plus uniquement parallèle, mais également distribuée**. Cette extension consiste à éclater les artefacts de couplage en deux parties appelées *reader* et *writer* (voir Figure 9.4). Les communications entre ces deux parties sont effectuées grâce à un intergiciel de communication, en l'occurence Data Distribution Service (DDS) [4].

9.4.2 Observation de la simulation : extension du méta-modèle

Le multi-modèle Concept Grid a été validé en comparant les profils de tension, intensité et puissance générés par la simulation MECSYCO à ceux observés sur un démonstrateur physique. Pour ce faire, **les résultats de simulation MECSYCO ont été collectés en utilisant l'extension AA4MM-Visu que nous avons développée conjointement avec l'équipe de MS4SG**. Cette extension, basée sur une idée de [Siebert, 2011], permet d'ajouter à un multi-modèle AA4MM, un système d'observation.

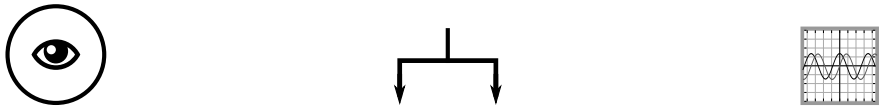
AA4MM-Visu étend Méta-AA4MM en proposant :

— **des modèles d'observation** (symbole en Figure 9.5c). Un modèle d'observation est un outil pré-existant permettant de stocker et/ou mettre en forme les données de simulation. Ce peut être un outil de présentation graphique (e.g. JFreeChart), un outil de traitement statistique (e.g. R), ou un simple fichier de données (e.g. un fichier CSV).

— **des artefacts d'observation** (symbole en Figure 9.5b). Un artefact d'observation est un artefact permettant de manipuler un modèle d'observation (i.e. d'injecter des données dans le modèle).

— **des m-agents d'observation** (symbole en Figure 9.5a). Un m-agent d'observation est un agent autonome associé à un modèle d'observation et son artefact d'observation. Un m-agent d'observation est chargé de collecter les données de simulation (dans un ordre temporel croissant) via ses artefacts de couplage d'entrée, et de les injecter dans son modèle d'observation.

4. `http://www.omg.org/spec/DDS/1.2`



(a) M-agent d'observation (b) Artefact d'observation (c) Modèle d'observation

FIGURE 9.5 – Symboles de AA4MM-Visu

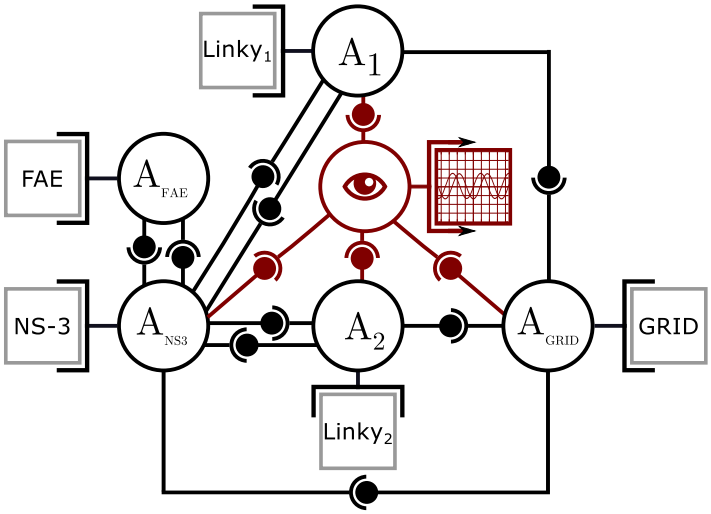


FIGURE 9.6 – Le multi-modèle Concept-Grid et son système A&A d'observation (en rouge sur le schéma)

Comme AA4MM-Visu reprend les propriétés de AA4MM en s'inscrivant dans le paradigme A&A, le système d'observation est distribuable et opère en parallèle de la simulation du multi-modèle. **Le système d'observation a alors un impact minimum sur les performances de la simulation lorsqu'il est exécuté sur une machine dédiée**.

La Figure 9.6 montre le multi-modèle Concept-Grid avec le système d'observation AA4MM-Visu qui a été utilisé pour collecter les données de simulation. La Figure 9.7 montre une partie des résultats de simulation collectés.

9.5 Conclusion

Dans ce chapitre, nous avons vu comment notre approche a été utilisée avec succès par les membres du projet MS4SG pour simuler un smart grid. Plus précisément, nous avons montré comment MECSYCO a permis de décrire et de simuler le multi-modèle Concept Grid. Ce cas concret montre les capacités de MECSYCO à construire et simuler un multi-modèle regroupant (1) différents domaines d'expertise, (2) utilisant différents formalismes et (3) différents outils métier de simulation faisant autorité dans leur domaine respectif, ou implémentant des standards d'interopérabilité logicielle.

Pour ce faire, l'équipe MS4SG a uniquement eu à développer un artefact de modèle pour chaque outil utilisé, et à définir des opérations de transformation. Sur ce point, nous avons vu que **le développement des artefacts de modèle est potentiellement la partie la plus complexe à effectuer.** En effet, à travers l'exemple de NS-3, nous voyons que l'interfaçage d'un simulateur qui n'est pas fait pour communiquer avec l'extérieur peut être une tâche ardue.

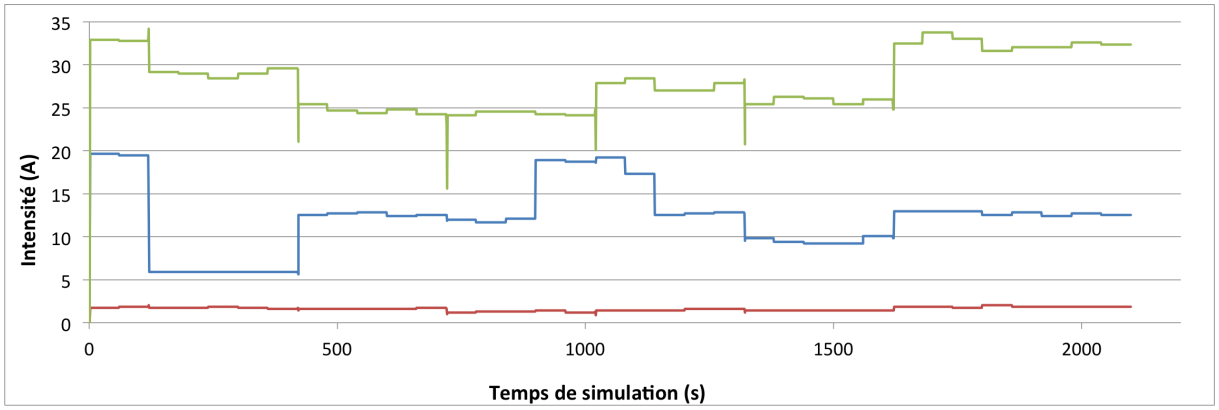


FIGURE 9.7 – Partie des résultats de simulation de Concept-Grid collectés grâce au système d'observation. Consommation du réseau électriques sur les trois phases (une courbe par phase).

Cependant, il faut noter que **MECSYCO permet de cibler les efforts d'intégration en les concentrant dans la définition de l'artefact de modèle. L'intégration d'un modèle dans MECSYCO est alors guidée par la question : « Quel est l'équivalent DEVS de ce modèle et de son simulateur ? ».**

De plus, une fois définis, les artefacts de modèle peuvent être réutilisés dans d'autres multi-modèles. **L'effort fourni pour définir l'artefact de modèle peut alors être rentabilisé dans le temps**. Il en a été ainsi par exemple pour l'artefact de modèle FMI qui a pu être directement réutilisé dans tous les autres multi-modèles impliquant des FMUs. De même, l'artefact de modèle NS-3 peut être utilisé pour interfacer n'importe quel modèle NS-3.

Une fois que les concepts spécifiques au domaine sont définis, la coordination de la simulation et l'intégration des formalismes sont gérées grâce aux spécifications opérationnelles que nous avons proposées dans les chapitres précédents. Ces spécifications ont permis une exécution parallèle et décentralisée du multi-modèle.

Enfin, nous avons montré comment notre approche a pu être aisément étendue pour permettre (1) la distribution d'un multi-modèle sur plusieurs machines (utilisant éventuellement des systèmes d'exploitation différents), (2) l'utilisation simultanée de plusieurs implémentations de MECSYCO (utilisant des langages de programmation différents), et (3) l'observation du déroulement de la simulation et la collecte des résultats de simulation.

Chapitre 10

Conclusion

Sommaire

10.1 Hypothèses de départ

Dans ce travail de thèse, nous nous sommes intéressés à l'étude des systèmes complexes dans le cadre de la démarche de M&S. Nous avons plus précisément été guidé par la question : comment décrire sans ambiguïté et simuler rigoureusement le modèle d'un système complexe ?

Ce questionnement nous a tout d'abord amené à faire l'hypothèse que le modèle d'un système complexe correspond à un multi-modèle : un ensemble hétérogène de modèles en interactions. A partir de cette hypothèse, nous avons défini un ensemble de pré-requis qui doivent être remplis pour répondre à notre problématique :

1. Permettre l'intégration rigoureuse de l'hétérogénéité d'un multi-modèle au niveau des représentations, des formalismes et des logiciels de simulation.
2. Assurer le passage automatique de la description d'un multi-modèle à sa simulation.
3. Gérer l'exécution parallèle d'un multi-modèle en respectant la contrainte de causalité.

Ces pré-requis ont amené la volonté de nous doter d'une approche apportant des solutions à la fois aux niveaux logiciel, formel et conceptuel. Parce qu'il nous semblait scientifiquement important de réutiliser dans la mesure du possible des travaux pré-existants, nous avons choisi d'étudier comment trois approches de M&S, à savoir le formalisme DEVS, l'intergiciel de co-simulation AA4MM et la démarche d'IDM, permettaient une fois combinées de répondre à notre problématique.

10.2 Contributions

Notre contribution correspond à une approche d'IDM de M&S des systèmes complexes permettant :

1. De décrire un multi-modèle de manière modulaire en combinant des modèles écrits dans plusieurs formalismes, implémentés dans plusieurs simulateurs et ayant des représentations différentes d'un système cible.

2. Puis de passer automatiquement à une co-simulation parallèle décentralisée et prouvée comme respectant la contrainte de causalité.

Cette contribution s'appuie sur deux apports.

Nous avons d'une part conçu l'intergiciel de co-simulation MECSYCO qui permet d'intégrer toute l'hétérogénéité d'un multi-modèle, et de gérer son exécution parallèle. Nous avons pour ce faire adapté aux concepts A&A de AA4MM, les spécifications opérationnelles de DEVS, c'est à dire ses protocoles de simulation et de communication ainsi que son algorithme de simulation parallèle.

Nous avons d'autre part conçu une approche d'IDM qui permet par une suite de transformations, de passer de la description d'un multi-modèle AA4MM à un programme exécutable MECSYCO. Cette approche d'IDM nous a amené à formaliser le méta-modèle de AA4MM, et à définir des règles de transformation qui permettent de générer du code MECSYCO à partir d'une description d'un multi-modèle. En systématisant l'implémentation d'un multi-modèle, notre approche d'IDM s'affranchit de toute manipulation de code, et ouvre ainsi la démarche de M&S des systèmes complexes aux non-informaticiens, notamment aux thématiciens et modélisateurs issus d'autres domaines scientifiques.

Nous avons évalué les propriétés de notre approche à travers plusieurs preuves de concept portant sur la modélisation et simulation du trafic autoroutier ainsi que la résolution numérique de systèmes d'équations différentielles. Ces preuves de concept ont été complétées par le compte rendu d'une mise en œuvre réussie de l'approche en situation réelle : la multi-modélisation d'un smart grid dans le cadre du projet MS4SG issu d'un partenariat entre Inria et EDF R&D.

A l'heure ou nous écrivons ces lignes, MECSYCO est en constant développement pour étudier les smart grids dans le cadre du projet MS4SG. Le code de l'intergiciel MECSYCO-Java est disponible à `http://mecsyco.com`.

10.3 Perspectives

Nous pensons que certains aspects de notre travail de thèse mériterait d'être étudiés à plus ou moins long terme.

Dans le court terme, nous pensons qu'il serait important de définir une méthodologie pour perfectionner le processus de multi-modélisation. Il ressort en effet des utilisations pratiques de MECSYCO que la construction d'un multi-modèle peut requérir la collecte d'un grand nombre d'informations auprès d'un grand nombre d'acteurs de la M&S (à savoir les thématiciens, les modélisateurs et les informaticiens ayant créé chaque modèle). On peut notamment vouloir savoir pour chaque modèle : Sur quelles hypothèses est basé le modèle ? Quelles sont les échelles temporelles et spatiales utilisées ? Avec quel formalisme est écrit le modèle ? En quel langage est écrit le logiciel de simulation ? Y a-t-il une API disponible ? La collecte de toutes ces informations peut vite devenir une tâche complexe qui va freiner le processus de multi-modélisation. Certaines de ces informations sont plus importantes que les autres car elles vont conditionner la mise en place du multi-modèle. Une méthodologie de construction de multi-modèle permettrait de savoir quelles sont toutes les informations qui doivent être recueillies, auprès de qui, et avec quel niveau de priorité.

Ensuite, si nous suivons jusqu'au bout le paradigme des approches basées composants dans lequel s'inscrit la multi-modélisation, nous voudrions pouvoir considérer les multi-modèles AA4MM comme des modèles à part entière capables d'être réutilisés comme des composants dans un autre multi-modèle. Ce dernier serait à son tour susceptible d'être intégré dans un multi-modèle, etc.

En d'autres mots, nous avons besoin d'une démarche de multi-modélisation hiérarchique. Pour ce faire, nous devons, comme avec DEVS, définir la fermeture par composition dans MECSYCO, c'est à dire décrire un multi-modèle de manière équivalente à celle d'un modèle atomique.

Nous tenons également à souligner que nous n'avons pas exploité dans cette thèse tout le potentiel de l'IDM pour la multi-modélisation, notamment sa capacité de traduire des modèles d'un langage à un autre grâce aux transformations de modèle à modèle. En effet, il nous paraît important de pouvoir définir un multi-modèle en utilisant un Domain Specific Language (DSL), c'est à dire un langage métier propre aux experts du domaine et taillé spécifiquement pour décrire une classe de systèmes particuliers (e.g. les smart grids). Grâce à l'IDM, des multi-modèles décris avec des DSL pourraient être automatiquement traduit sous la forme de multi-modèles AA4MM. Ces derniers pourront à leur tour être traduits sous la forme d'un programme exécutable comme décrit dans cette thèse. L'intérêt est alors que chaque thématicien pourra utiliser son propre langage pour décrire des multi-modèles, tout en profitant des avantages de MECSYCO pour simuler son système.

A plus long terme, comme mentionné dans le Chapitre 3, plusieurs auteurs de la communauté M&S insistent sur le besoin de se doter d'outils capables d'adapter dynamiquement la structure d'un multi-modèle (i.e. d'ajouter, d'enlever, ou de changer un ou plusieurs modèles) pendant la simulation pour prendre en compte l'apparition de phénomènes émergents dans les modèles. Par exemple, dans [Xiong et al., 2009], deux modèles sont utilisés successivement pour représenter un ensemble de piétons se déplaçant dans un couloir. Un modèle équationnel est utilisé pour représenter le système lorsque celui-ci est stable, alors qu'un modèle multi-agent est utilisé lorsqu'un événement perturbant le flux se produit. Un dispositif intelligent est mis en place pour monitorer l'évolution de la simulation et changer automatiquement de modèle pendant l'exécution. Les spécifications de MECSYCO que nous avons présentées dans cette thèse ne permettent de décrire que des multi-modèles à structure statique. Cependant, nous pensons que MECSYCO peut être étendu pour profiter de la richesse d'expression du paradigme multi-agent (voir Section 2.6.3). En effet, dans [Yilmaz and Ören, 2004], les auteurs ont montré que le concept d'agent autonome est adapté pour représenter les processus intelligents nécessaires à l'adaptation dynamique du multi-modèle en cours de simulation.

Enfin, nous pourrions envisager à terme d'utiliser les capacités d'auto-organisation des systèmes multi-agents pour automatiser complètement la démarche de modélisation et simulation. Plus précisément, il s'agirait de définir un ensemble d'agents capables de s'auto-organiser pour construire des multi-modèles à partir des besoins des thématiciens et d'une base de donnée de modèles. Une telle approche offrirait alors des dispositifs intelligents capables de fournir directement des réponses aux questionnements des thématiciens.

Publications de l'auteur

Revue internationale avec comité de lecture

[1] Camus, B., Bourjot, C., and Chevrier, V. (2015). Considering a multi- level model as a society of interacting models : Application to a collective motion example. *Journal of Artificial Societies and Social Simulation*, 18(3) :7.

Conférences internationales avec actes

[2] Camus, B., Bourjot, C., and Chevrier, V. (2015). Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 15)*. Society for Computer Simulation International.

[3] Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J.-P., and Morais, H. (2015). Multi-agent multi-model simulation of smart grids in the MS4SG project. In Demazeau, Y., Decker, K. S., Bajo Pérez, J., and de la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, vol. 9086 of Lecture Notes in Computer Science. Springer.

[4] Camus, B., Bourjot, C., and Chevrier, V. (2013). Multi-level modeling as a society of interacting models. In *Proceedings of the Spring Simulation Multi-Conference (SpringSim), Agent-Directed Simulation (ADS) Symposium, ADSS 13*, pages 3 :1–3 :8. SCS International.

Conférence nationale avec actes

[5] Camus, B., Siebert, J., Bourjot, C., and Chevrier, V. (2012). Modélisation multi-niveaux dans AA4MM. In Chevailler, P., and Mermet, B. *Journées Francophones sur les Systèmes Multi-Agents*, Oct 2012, Honfleur, France. Cépaduès, pp.43-52.

Démonstrations

[6] Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J-P., Morais, H., Deneuville B., and Chilard, O. (2015) Smart Grids Simulation with MECSYCO. In Demazeau, Y., Decker, K. S., Bajo Pérez, J., and de la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, volume 9086 of Lecture Notes in Computer Science. Springer.

[7] Vaubourg, J., Presse, Y., Camus, B., Ciarletta, L., Chevrier, V., Tavella, J.-P., Deneuville, B., and Chilard, O. (2015). Simulation de smart grids avec MECSYCO. In Vercouter, L. and Picard, G., editors, *23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15)*, pages 217–218, Rennes, France. Cépaduès.

Bibliographie

[Adelantado, 2012] Adelantado, M. (2012). Guide HLA (High Level Architecture). Technical Report DGA S-CAT N° 10006, Direction Générale de l'Armement (DGA).

[Anderson et al., 1972] Anderson, P. W. et al. (1972). More is different. *Science*, 177(4047) :393–396.

[Argent, 2004] Argent, R. M. (2004). An overview of model integration for environmental applications-components, frameworks and semantics. *Environmental Modelling and Software*, 19(3) :219–234.

[Balci, 1998] Balci, O. (1998). Verification, validation, and accreditation. In *Proceedings of the 30th conference on Winter simulation*, pages 41–4. IEEE Computer Society Press.

[Batty, 2009] Batty, M. (2009). Urban modeling. *International Encyclopedia of Human Geography, Elsevier, Oxford*.

[Belem and Müller, 2009] Belem, M. and Müller, J.-P. (2009). Toward a conceptual framework for multi-points of view analysis in complex system modeling : OREA model. In Demazeau, Y., Pavón, J., Corchado, J., and Bajo, J., editors, *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, volume 55, pages 548–556. Springer Berlin Heidelberg.

[Bézivin and Gerbé, 2001] Bézivin, J. and Gerbé, O. (2001). Towards a precise definition of the OMG/MDA framework. In *Proceedings. 16th Annual International Conference on Automated Software Engineering, 2001.(ASE 2001).*, pages 273–280. IEEE.

[Bisgambiglia and Franceschini, 2013] Bisgambiglia, P.-A. and Franceschini, R. (2013). Agent-oriented approach based on discrete event systems (wip). In *SpringSim (TMS-DEVS)*, page 27.

[Blochwitz et al., 2012] Blochwitz, T., Otter, M., Åkesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., et al. (2012). Functional mockup interface 2.0 : The standard for tool independent exchange of simulation models. In *9th International Modelica Conference*.

[Blochwitz et al., 2011] Blochwitz, T., Otter, M., Arnold, M., et al. (2011). The functional mockup interface for tool independent exchange of simulation models. In *8th International Modelica Conference, Dresden*, pages 20–22.

[Borland and Vangheluwe, 2003] Borland, S. and Vangheluwe, H. (2003). Transforming statecharts to DEVS. In *Summer Computer Simulation Conference (Student Workshop)*, pages S154–S159.

[Bretagnolle et al., 2006] Bretagnolle, A., Daudet, E., and Pumain, D. (2006). From theory to modelling : urban systems as complex systems. *CyberGeo : European Journal of Geography*, (335) :1–17.

[Breunese et al., 1998] Breunese, A., Broenink, J., Top, J., and Akkermans, J. (1998). Libraries of reusable models : theory and application. *Simulation*, 71(1) :7–22.

[Broman et al., 2013] Broman, D., Brooks, C., Greenberg, L., Lee, E. A., Masin, M., Tripakis, S., and Wetter, M. (2013). Determinate composition of FMUs for co-simulation. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, EMSOFT '13, pages 2 :1–2 :12, Piscataway, NJ, USA. IEEE Press.

[Brooks et al., 2008a] Brooks, C., Lee, E., Liu, X., Neuendorffer, S., Zhao, Y., and Zheng, H. (2008a). Heterogeneous concurrent modeling and design in java (volume 1 : Introduction to ptolemy ii). Technical Report UCB/EECS-2008-28, EECS Department, University of California, Berkeley.

[Brooks et al., 2008b] Brooks, C., Lee, E., Liu, X., Neuendorffer, S., Zhao, Y., and Zheng, H. (2008b). Heterogeneous concurrent modeling and design in java (volume 3 : Ptolemy ii domains). Technical Report UCB/EECS-2008-28, EECS Department, University of California, Berkeley.

[Bryant, 1979] Bryant, R. E. (1979). Simulation on a distributed system. In *Proc. of the 16th Design Automation Conference*, pages 544–552.

[Camazine et al., 2001] Camazine, S., Deneubourg, J.-L., Franks, N. R., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2001). *Self-Organization in Biological Systems*. Princeton University Press.

[Camus et al., 2013] Camus, B., Bourjot, C., and Chevrier, V. (2013). Multi-level modeling as a society of interacting models. In *Proceedings of the Spring Simulation Multi-Conference (SpringSim), Agent-Directed Simulation (ADS) Symposium*, ADSS 13, pages 3 :1–3 :8. Society for Computer Simulation International.

[Camus et al., 2015a] Camus, B., Bourjot, C., and Chevrier, V. (2015a). Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 15)*. Society for Computer Simulation International.

[Camus et al., 2015b] Camus, B., Bourjot, C., and Chevrier, V. (2015b). Considering a multi-level model as a society of interacting models : Application to a collective motion example. *Journal of Artificial Societies and Social Simulation*, 18(3) :7.

[Carothers et al., 1997] Carothers, C. D., Fujimoto, R. M., Weatherly, R. M., and Wilson, A. L. (1997). Design and implementation of HLA time management in the RTI version f. 0. In *Proceedings of the 29th conference on Winter simulation*, pages 373–380. IEEE Computer Society.

[Cellier, 1979] Cellier, F. E. (1979). Combined continuous/discrete system simulation languages–usefulness, experiences and future development. *Methodology in systems modelling and simulation*, pages 201–220.

[Cetinkaya et al., 2010a] Cetinkaya, D., Verbraeck, A., and Seck, M. D. (2010a). Applying a model driven approach to component based modeling and simulation. In *Simulation Conference (WSC), Proceedings of the 2010 Winter*, pages 546–553. IEEE.

[Cetinkaya et al., 2010b] Cetinkaya, D., Verbraeck, A., and Seck, M. D. (2010b). A metamodel and a devs implementation for component based hierarchical simulation modeling. In Biaz, S., editor, *Proceedings of the 43rd Annual Simulation Symposium (ANSS'10) part of the 2010 Spring Simulation Multi Conference (SpringSim'10)*, pages 130–137. Society for Modeling and Simulation International (SCS).

[Chandy and Misra, 1979] Chandy, K. M. and Misra, J. (1979). Distributed simulation : A case study in design and verification of distributed programs. *Software Engineering, IEEE Transactions on*, (5) :440–452.

[Charrier, 2009] Charrier, R. (2009). *L'intelligence en essaim sous l'angle des systèmes complexes : étude d'un système multi-agent réactif à base d'itérations logistiques couplées.* These, Université Nancy 2.

[Chavalarias et al., 2009] Chavalarias, D., Bourgine, P., and E. Perrier, F. Amblard, F. Arlabosse, et al. (2009). French Roadmap for complex Systems 2008-2009.

[Chazelle, 2012] Chazelle, B. (2012). Natural algorithms and influence systems. *Communications of the ACM*, 55(12) :101–110.

[Dacharry and Giambiasi, 2005] Dacharry, H. P. and Giambiasi, N. (2005). Formal verification with timed automata and DEVS models : a case study. In *Proc. of Argentine Symposium on Software Engineering*, pages 251–265. Citeseer.

[Dahmann and Morse, 1998] Dahmann, J. and Morse, K. (1998). High level architecture for simulation : an update. In *Distributed Interactive Simulation and Real-Time Applications, 1998. Proceedings. 2nd International Workshop on*, pages 32–40.

[Dahmann et al., 1997] Dahmann, J. S., Fujimoto, R. M., and Weatherly, R. M. (1997). The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149. IEEE Computer Society.

[Davis and Hillestad, 1993] Davis, P. K. and Hillestad, R. (1993). Families of models that cross levels of resolution : issues for design, calibration and management. In *Proceedings of the 25th conference on Winter simulation*, WSC '93, pages 1003–1012, New York, NY, USA. ACM.

[De Lara and Vangheluwe, 2002] De Lara, J. and Vangheluwe, H. (2002). Using AToM as a meta-CASE tool. In *ICEIS*, volume 2, pages 642–649.

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behaviour in agent-based systems. In *Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*, pages 117–132.

[Diallo et al., 2011] Diallo, S. Y., Herencia-Zapana, H., Padilla, J. J., and Tolk, A. (2011). Understanding interoperability. In *Proceedings of the 2011 Emerging M&S Applications in Industry and Academia Symposium*, EAIA '11, pages 84–91, San Diego, CA, USA. Society for Computer Simulation International.

[Duboz et al., 2012] Duboz, R., Bonté, B., and Quesnel, G. (2012). Vers une spécification des modèles de simulation de systèmes complexes. *Stud. Inform. Univ.*, 10(1) :7–37.

[Duboz et al., 2003] Duboz, R., Ramat, É., and Preux, P. (2003). Scale transfer modelling : Using emergent computation for coupling an ordinary differential equation system with areactive agent model. *Systems Analysis Modelling Simulation*, 43(6) :793–814.

[Duboz et al., 2006] Duboz, R., Versmisse, D., Quesnel, G., Muzy, A., and Ramat, E. (2006). Specification of dynamic structure discrete event multiagent systems. *Simulation Series*, 38(2) :103.

[Durak, 2015] Durak, U. (2015). Model based simulation engineering. SpringSim2015 Tutorial.

[Eker et al., 2003] Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., and Xiong, Y. (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1) :127–144.

[El Hmam et al., 2006] El Hmam, M., Abouaissa, Hassane ; Jolly, D., and Benasser, A. (2006). Macro-micro simulation of traffic flow. In *Proceeding of the 12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM*, volume 12-1, pages 351–356.

[El Hmam et al., 2008] El Hmam, M. S., Jolly, D., Abouaissa, H., and Benasser, A. (2008). Modelisation hybride du flux de trafic. *Méthodologies et Heuristiques pour l'Optimisation des Systèmes Industriels (MHOSI'08)*, pages 193–198.

[Esquembre and Christian, 2007] Esquembre, F. and Christian, W. (2007). Ordinary differential equations. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*. CRC Press.

[Ferber, 1995] Ferber, J. (1995). *Les systèmes multi-agents : vers une intelligence collective*. InterEditions, Paris.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Multi Agent Systems, 1998. Proceedings. International Conference on*, pages 128–135.

[Fishwick and Zeigler, 1992] Fishwick, P. A. and Zeigler, B. P. (1992). A multimodel methodology for qualitative model engineering. *ACM Trans. Model. Comput. Simul.*, 2(1) :52–81.

[Fujimoto, 2001] Fujimoto, R. M. (2001). Parallel simulation : parallel and distributed simulation systems. In *Proceedings of the 33nd conference on Winter simulation*, WSC '01, pages 147–157. IEEE Computer Society.

[Galán et al., 2009] Galán, J. M., Izquierdo, L. R., Izquierdo, S. S., Santos, J. I., del Olmo, R., López-Paredes, A., and Edmonds, B. (2009). Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1) :1.

[Gašević et al., 2007] Gašević, D., Kaviani, N., and Hatala, M. (2007). On metamodeling in megamodels. In Engels, G., Opdyke, B., Schmidt, D., and Weil, F., editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 91–105. Springer Berlin Heidelberg.

[Gaud et al., 2008] Gaud, N., Galland, S., Gechter, F., Hilaire, V., and Koukam, A. (2008). Holonic multilevel simulation of complex systems : Application to real-time pedestrians simulation in virtual urban environment. *Simulation Modelling Practice and Theory*, 16(10) :1659 – 1676. The Analysis of Complex Systems.

[Gil-Quijano et al., 2012] Gil-Quijano, J., Louail, T., and Hutzler, G. (2012). From biological to urban cells : Lessons from three multilevel agent-based models. In Desai, N., Liu, A., and Winikoff, M., editors, *Principles and Practice of Multi-Agent Systems*, volume 7057 of *Lecture Notes in Computer Science*, pages 620–635. Springer Berlin Heidelberg.

[Goderis et al., 2009] Goderis, A., Brooks, C., Altintas, I., Lee, E. A., and Goble, C. (2009). Heterogeneous composition of models of computation. *Future Generation Computer Systems*, 25(5) :552–560.

[Gray et al., 2007] Gray, J., Tolvanen, J.-P., Kelly, S., Gokhale, A., Neema, S., and Sprinkle, J. (2007). Domain-specific modeling. In Fishwick, P. A., editor, *Handbook of Dynamic System Modeling*, pages 7–1. CRC Press.

[Henderson et al., 2006] Henderson, T. R., Roy, S., Floyd, S., and Riley, G. F. (2006). NS-3 project goals. In *Proceeding of WNS2 '06*, page 13. ACM.

[Hofmann, 2005] Hofmann, M. A. (2005). Modeling assumptions : How they affect validation and interoperabilitiy, 05e-siw-002,. In *Proceedings of the European Simulation Interoperability Workshop*.

[Hooper, 1986] Hooper, J. W. (1986). Strategy-related characteristics of discrete-event languages and models. *SIMULATION*, 46(4) :153–159.

[Hopkinson et al., 2006] Hopkinson, K., Wang, X., Giovanini, R., Thorp, J., Birman, K., and Coury, D. (2006). EPOCHS : a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *Power Systems, IEEE Transactions on*, 21(2) :548–558.

[IEEE, 2010a] IEEE (2010a). IEEE standard for modeling and simulation – high level architecture (HLA)– federate interface specification. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, pages 1–378.

[IEEE, 2010b] IEEE (2010b). IEEE standard for modeling and simulation – high level architecture (HLA)– framework and rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pages 1–38.

[IEEE, 2010c] IEEE (2010c). IEEE standard for modeling and simulation – high level architecture (HLA)– object model template (OMT) specification. *IEEE Std 1516.2-2010 (Revision of IEEE Std 1516.2-2000)*, pages 1–110.

[Jacques and Wainer, 2002] Jacques, C. J. and Wainer, G. A. (2002). Using the CD++ DEVS toolkit to develop petri nets. In *Summer Computer Simulation Conference*, pages 51–56. Society for Computer Simulation International ; 1998.

[Jouault and Kurtev, 2006] Jouault, F. and Kurtev, I. (2006). Transforming models with ATL. In Bruel, J.-M., editor, *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138. Springer Berlin Heidelberg.

[Karsai et al., 2000] Karsai, G., Nordstrom, G., Ledeczi, A., and Sztipanovits, J. (2000). Specifying graphical modeling systems using constraint-based meta models. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 89–94. IEEE.

[Kim et al., 1996] Kim, K. H., Seong, Y. R., Kim, T. G., and Park, K. H. (1996). Distributed simulation of hierarchical DEVS models : Hierarchical scheduling locally and time warp globally. *Transactions of the Society for Computer Simulation International*, 13(0) :3.

[Kim et al., 2009] Kim, S., Sarjoughian, H. S., and Elamvazhuthi, V. (2009). DEVS-suite : A simulator supporting visual experimentation design and behavior monitoring. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 161 :1–161 :7, San Diego, CA, USA. Society for Computer Simulation International.

[Kim, 2007] Kim, T. G. (2007). Devs formalism for modeling of discrete event systems. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*. CRC Press.

[Kim and Kim, 1998] Kim, Y. J. and Kim, T. G. (1998). A heterogeneous simulation framework based on the DEVS BUS and the high level architecture. In *Simulation Conference Proceedings, 1998. Winter*, volume 1, pages 421–428. IEEE.

[Kolovos et al., 2010] Kolovos, D. S., Rose, L. M., Abid, S. B., Paige, R. F., Polack, F. A., and Botterweck, G. (2010). Taming EMF and GMF using model transformation. In Petriu, D. C., Rouquette, N., and Haugen, O., editors, *Model Driven Engineering Languages and Systems*, volume 6394 of *Lecture Notes in Computer Science*, pages 211–225. Springer Berlin Heidelberg.

[Kubera et al., 2007] Kubera, Y., Mathieu, P., Picault, S., et al. (2007). La complexité dans les simulations multi-agents. *Actes des 15e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2007)*, pages 139–148.

[Lane, 2006] Lane, D. (2006). Hierarchy, complexity, society. In Pumain, D., editor, *Hierarchy in Natural and Social Sciences*, volume 3 of *Methodos Series*, pages 81–119. Springer Netherlands.

[Lara and Vangheluwe, 2002] Lara, J. and Vangheluwe, H. (2002). AToM3 : A tool for multi-formalism and meta-modelling. In Kutsche, R.-D. and Weber, H., editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg.

[Leclerc et al., 2010] Leclerc, T., Siebert, J., Chevrier, V., Ciarletta, L., and Festor, O. (2010). Multi-modeling and co-simulation-based mobile ubiquitous protocols and services development and assessment. In Sénac, P., Ott, M., and Seneviratne, A., editors, *7h International ICST Conference on Mobile and Ubiquitous Systems : Computing, Networking, and Services*, volume 73 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 273–284. Springer Berlin Heidelberg.

[Lédeczi et al., 2001] Lédeczi, Á., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., and Karsai, G. (2001). Composing domain-specific design environments. *Computer*, 34(11) :44–51.

[Ledeczi et al., 2001] Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., and Volgyesi, P. (2001). The generic modeling environment. In *Workshop on Intelligent Signal Processing*.

[Lee, 2010] Lee, E. A. (2010). Disciplined heterogeneous modeling. In D.C. Petriu, N. Rouquette, O. H., editor, *Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering, Languages, and Systems (MODELS)*, pages 273–287. LNCS 6395, Springer-Verlag.

[Lorenz, 1963] Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2) :130–141.

[Michel, 2007] Michel, F. (2007). The IRM4S model : the influence/reaction principle for multiagent based simulation. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 133. ACM.

[Minsky, 1965] Minsky, M. (1965). Matter, mind and models. In *International Federation of Information Processing Congress*, volume 1, pages 45–49.

[Mosterman and Vangheluwe, 2002] Mosterman, P. J. and Vangheluwe, H. (2002). Computer automated multi-paradigm modeling. *ACM Transactions on Modeling and Computer Simulation*, 12(4) :1–7.

[Müller, 2007] Müller, J.-P. (2007). Mimosa : Using ontologies for modelling and simulation. In *GI Jahrestagung (1)*, pages 227–231.

[Müller, 2010] Müller, J.-P. (2010). A framework for integrated modeling using a knowledge-driven approach. In Swayne, D. A., Yang, W., Voinov, A. A., Rizzoli, A., and Filatova, T., editors, *Proceedings of the iEMSs Firth Biennial Meeting : International Congress on Environmental Modelling and Software (iEMSs 2010)*.

[Müller and Aubert, 2011] Müller, J.-P. and Aubert, S. (2011). Une ontologie pour une représentation multi-niveau de et par les systèmes sociaux, rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels : Echelles et modélisations multi-niveaux. *Spatial Simulation for the Social Systèmes (SSSS)*.

[Müller and Diallo, 2012] Müller, J.-P. and Diallo, A. (2012). Vers une méthode multi-point de vue de modélisation multi-agent (présentation courte). In Chevaillier, P. and Mermet, B., editors, *JFSMA*, pages 33–42. Cepadues Editions.

[Muller and Widl, 2013] Muller, W. and Widl, E. (2013). Linking FMI-based components with discrete event systems. In *Systems Conference (SysCon), 2013 IEEE International*, pages 676–680.

[Natrajan and Reynolds, 2001] Natrajan, A. and Reynolds, Jr., P. F. (July 2001). Concurrent representations for jointly-executing models. Technical Report CS-2001-20, University of Virginia.

[Nordstrom et al., 1999] Nordstrom, G., Sztipanovits, J., Karsai, G., and Ledeczi, A. (1999). Metamodeling-rapid design and evolution of domain-specific modeling environments. In *Engineering of Computer-Based Systems, 1999. Proceedings. ECBS '99. IEEE Conference and Workshop on*, pages 68–74.

[Nutaro, 2007] Nutaro, J. (2007). Discrete-event simulation of continuous systems. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*. CRC Press.

[OMG, 2003] OMG (2003). *Model Driven Architecture (MDA) Guide*. OMG doc. ab/2003-06-01.

[OMG, 2014] OMG (2014). OMG meta object facility (MOF) core specification v2.4.2. *OMG Document*.

[OMG, 2015a] OMG (2015a). Meta object facility (MOF) 2.0 query/view/transformation specification. *OMG Document*.

[OMG, 2015b] OMG (2015b). MOF model to text transformation language, v1.0. *OMG Document*.

[Omicini et al., 2008] Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A metamodel for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3) :432–456.

[Omicini et al., 2004] Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004). Coordination artifacts : Environment-based coordination for intelligent agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, pages 286–293, Washington, DC, USA. IEEE Computer Society.

[Ören, 1987] Ören, T. (1987). Model update : A model specification formalism with a generalized view of discontinuity. In *Proceedings of the Summer Computer Simulation Conference*, pages pp. 689–694, Montreal, Quebec, Canada.

[Ören, 2011a] Ören, T. (2011a). A critical review of definitions and about 400 types of modeling and simulation. *SCS Magazine*, 3 :142–151.

[Ören, 2011b] Ören, T. (2011b). The many facets of simulation through a collection of about 100 definitions. *SCS M&S Magazine*, 2 :82–92.

[Page et al., 2004] Page, E. H., Briggs, R., and Tufarolo, J. (2004). Toward a family of maturity models for the simulation interconnection problem. In Press, I. C., editor, *Proceedings of the Spring Simulation Interoperability Workshop*.

[Perumalla, 2007] Perumalla, K. S. (2007). Model execution. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*. CRC Press.

[Polhill and Gotts, 2006] Polhill, J. G. and Gotts, N. M. (2006). A new approach to modelling frameworks. In *First World Conference on Social Simulation (WCSS 2006), Kyoto, Japan*.

[Praehofer, 1991] Praehofer, H. (1991). System theoretic formalisms for combined discrete-continuous system simulation. *International Journal of General System*, 19(3) :226–240.

[Praehofer and Reisinger, 1994] Praehofer, H. and Reisinger, G. (1994). Distributed simulation of DEVS-based multiformalism models. In *AIS*, volume 94, pages 150–156. DTIC Document.

[Pun-Cheng, 2000] Pun-Cheng, L. S. (2000). A new face-entity concept for modeling urban morphology. *URISA-WASHINGTON DC-*, 12(3) :47–56.

[Quesnel, 2006] Quesnel, G. (2006). *Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes : apports pour la simulation de Systèmes Multi-Agents.* PhD thesis, Université du Littoral côte d'Opale.

[Quesnel et al., 2009] Quesnel, G., Duboz, R., and Ramat, É. (2009). The virtual laboratory environment – an operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17(4) :641 – 653.

[Quesnel et al., 2005] Quesnel, G., Duboz, R., Versmisse, D., and Ramat, É. (2005). DEVS coupling of spatial and ordinary differential equations : VLE framework. In *Proceedings of the Open International Conference on Modeling & Simulation Conference*, pages 281–294.

[Ramat, 2006] Ramat, E. (2006). Introduction à la modélisation et à la simulation à événements discrets. *Modélisation et simulation multiagents : applications aux Sciences de l'Homme et de la Société, Lavoisier, Coll. Science informatique et SHS.*

[Reynolds et al., 1997] Reynolds, Jr., P. F., Natrajan, A., and Srinivasan, S. (1997). Consistency maintenance in multiresolution simulation. *ACM Trans. Model. Comput. Simul.*, 7(3) :368–392.

[Ricci et al., 2011] Ricci, A., Piunti, M., and Viroli, M. (2011). Environment programming in multi-agent systems : an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2) :158–192.

[Ricci et al., 2007a] Ricci, A., Viroli, M., and Omicini, A. (2007a). The A&A programming model and technology for developing agent environments in MAS. In *PROMAS*, pages 89–106.

[Ricci et al., 2007b] Ricci, A., Viroli, M., and Omicini, A. (2007b). Give agents their artifacts : the A&A approach for engineering working environments in MAS. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, AAMAS, pages 150 :1–150 :3. ACM.

[Sarjoughian and Zeigler, 1998] Sarjoughian, H. S. and Zeigler, B. (1998). DEVSJAVA : Basis for a DEVS-based collaborative M&S environment. *Simulation Series*, 30 :29–36.

[Seck and Honig, 2012] Seck, M. D. and Honig, H. J. (2012). Multi-perspective modelling of complex phenomena. *Comput. Math. Organ. Theory*, 18(1) :128–144.

[Sedaghat, 2007] Sedaghat, H. (2007). Difference equations as discrete dynamical systems. In Fishwick, P. A., editor, *Handbook of dynamic system modeling.* CRC Press.

[Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE Softw.*, 20(5) :26–32.

[Siebert, 2011] Siebert, J. (2011). *Approche multi-agent pour la multi-modélisation et le couplage de simulations. Application à l'étude des influences entre le fonctionnement des réseaux ambiants et le comportement de leurs utilisateurs.* These, Université Henri Poincaré, Nancy 1.

[Siebert et al., 2010a] Siebert, J., Ciarletta, L., and Chevrier, V. (2010a). Agents & artefacts for multiple models coordination : Objective and decentralized coordination of simulators. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2024–2028, New York, NY, USA. ACM.

[Siebert et al., 2010b] Siebert, J., Ciarletta, L., and Chevrier, V. (2010b). Agents and artefacts for multiple models co-evolution : building complex system simulation as a set of interacting models. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems : volume 1 - Volume 1*, AAMAS '10, pages 509–516. International Foundation for Autonomous Agents and Multiagent Systems.

[Siebert et al., 2009] Siebert, J., Rehm, J., Chevrier, V., Ciarletta, L., and Méry, D. (2009). AA4MM coordination model : Event-B specification. Research Report 7081. Version 0.2.

[Sprinkle et al., 2010] Sprinkle, J., Rumpe, B., Vangheluwe, H., and Karsai, G. (2010). Metamodelling : state of the art and research challenges. In *Proceedings of the 2007 International Dagstuhl conference on Model-based engineering of embedded real-time systems*, MBEERTS'07, pages 57–76. Springer-Verlag.

[Taillandier et al., 2012] Taillandier, P., Vo, D.-A., Amouroux, E., and Drogoul, A. (2012). GAMA : a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *Principles and Practice of Multi-Agent Systems*, pages 242–258. Springer.

[Taylor et al., 2013] Taylor, S. J., Khan, A., Morse, K. L., Tolk, A., Yilmaz, L., and Zander, J. (2013). Grand challenges on the theory of modeling and simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium*, page 34. Society for Computer Simulation International.

[Taylor, 2003] Taylor, S. J. E. (2003). HLA - CSPIF : The high level architecture - COTS simulation package interoperation forum. In *2003 Fall Simulation Interoperability Workshop*, number 03F-SIW-126.

[Taylor, 2007] Taylor, S. J. E. (2007). Distributed modeling. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*. CRC Press.

[Taylor et al., 2006] Taylor, S. J. E., Wang, X., Turner, S., and Low, M. (2006). Integrating heterogeneous distributed COTS discrete-event simulation packages : an emerging standards-based approach. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on*, 36(1) :109–122.

[Thomas, 2005] Thomas, V. (2005). *Proposition d'un formalisme pour la construction automatique d'interactions dans les systèmes multi-agents réactifs*. PhD thesis, Université Henri Poincaré, Nancy 1.

[Tisseau, 2001] Tisseau, J. (2001). Réalité virtuelle : autonomie in virtuo. *Habilitationa diriger des recherches, Université de Rennes*, 1.

[Tolk et al., 2007] Tolk, A., Saikou, D., and Charles, T. (2007). Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systems engineering. *Journal of Systemics, Cybernetics and Informatics*.

[Traoré and Muzy, 2006] Traoré, M. K. and Muzy, A. (2006). Capturing the dual relationship between simulation models and their context. *Simulation Modelling Practice and Theory*, 14(2) :126–142.

[Turnitsa and Tolk, 2008] Turnitsa, C. and Tolk, A. (2008). Knowledge representation and the dimensions of a multi-model relationship. In *Simulation Conference, 2008. WSC 2008. Winter*, pages 1148–1156.

[Uhrmacher, 2012] Uhrmacher, A. M. (2012). Seven pitfalls in modeling and simulation research. In *Proceedings of the Winter Simulation Conference*, WSC '12, pages 318 :1–318 :12. Winter Simulation Conference.

[Uhrmacher and Schattenberg, 1998] Uhrmacher, A. M. and Schattenberg, B. (1998). Agents in discrete event simulation. In *European Simulation Symposium - ESS'98*, pages 129–136. SCS.

[Vangheluwe, 2000] Vangheluwe, H. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Computer-Aided Control System Design. CACSD. IEEE International Symposium on*, pages 129–134.

[Vangheluwe et al., 2002] Vangheluwe, H., De Lara, J., and Mosterman, P. J. (2002). An introduction to multi-paradigm modelling and simulation. In *Proc. AIS2002.*, pages 9–20.

[Varga and Hornig, 2008] Varga, A. and Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In *Proceedings of ICST*, page 60.

[Vaubourg et al., 2015a] Vaubourg, J., Chevrier, V., and Ciarletta, L. (2015a). Co-Simulation of IP Network Models in the Smart Grids Context, using a DEVS-based Platform. Technical report.

[Vaubourg et al., 2015b] Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J.-P., and Morais, H. (2015b). Multi-agent multi-model simulation of smart grids in the MS4SG project. In Demazeau, Y., Decker, K. S., Bajo Pérez, J., and de la Prieta, F., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, volume 9086 of *Lecture Notes in Computer Science*, pages 240–251. Springer International Publishing.

[Vaubourg et al., 2015c] Vaubourg, J., Presse, Y., Camus, B., Ciarletta, L., Chevrier, V., Tavella, J.-P., Deneuville, B., and Chilard, O. (2015c). Simulation de smart grids avec MECSYCO. In Vercouter, L. and Picard, G., editors, *23es Journées Francophones sur les Systèmes Multi-Agents (JFSMA'15)*, pages 217–218, Rennes, France. Cépaduès.

[Vicino et al., 2014] Vicino, D., Dalle, O., and Wainer, G. (2014). A data type for discretized time representation in DEVS. In *7th International ICST Conference on Simulation Tools and Techniques*.

[Vo et al., 2012] Vo, D.-A., Drogoul, A., and Zucker, J.-D. (2012). An operational meta-model for handling multiple scales in agent-based simulations. In *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on*, pages 1–6.

[Wainer and Giambiasi, 2001] Wainer, G. A. and Giambiasi, N. (2001). Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation*, 76(1) :22–39.

[Whitner and Balci, 1989] Whitner, R. B. and Balci, O. (1989). Guidelines for selecting and using simulation model verification techniques. In *Proceedings of the 21st Conference on Winter Simulation*, WSC '89, pages 559–568, New York, NY, USA. ACM.

[Wilensky, 1997a] Wilensky, U. (1997a). Netlogo traffic basic model. http ://ccl.northwestern.edu/netlogo/models/trafficbasic. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[Wilensky, 1997b] Wilensky, U. (1997b). Netlogo wolf sheep predation model. http ://ccl.northwestern.edu/netlogo/models/wolfsheeppredation. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[Wilensky, 1999] Wilensky, U. (1999). Netlogo. http ://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[Xiong et al., 2009] Xiong, M., Cai, W., Zhou, S., Low, M. Y.-H., Tian, F., Chen, D., Ong, D. W. S., and Hamilton, B. D. (2009). A case study of multi-resolution modeling for crowd simulation. In Wainer, G. A., Shaffer, C. A., McGraw, R. M., and Chinni, M. J., editors, *SpringSim*. SCS/ACM.

[Yilmaz et al., 2007] Yilmaz, L., Lim, A., Bowen, S., and Oren, T. (2007). Requirements and design principles for multisimulation with multiresolution, multistage multimodels. In *Simulation Conference, 2007 Winter*, pages 823–832.

[Yilmaz and Ören, 2004] Yilmaz, L. and Ören, T. (2004). Dynamic model updating in simulation with multimodels : A taxonomy and a generic agent-based architecture. In *In Proceedings of SCSC 2004 - Summer Computer Simulation Conference,*, pages 3–8.

[Yilmaz and Tolk, 2008] Yilmaz, L. and Tolk, A. (2008). A unifying multimodel taxonomy and agent-supported multisimulation strategy for decision-support. In Phillips-Wren, G., Ichalkaranje, N., and Jain, L., editors, *Intelligent Decision Making : An AI-Based Approach*, volume 97 of *Studies in Computational Intelligence*, pages 193–226. Springer Berlin Heidelberg.

[Zeigler et al., 2000] Zeigler, B., Praehofer, H., and Kim, T. (2000). *Theory of Modeling and Simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

[Zeigler, 2006] Zeigler, B. P. (2006). Embedding DEV&DESS in DEVS. In *Proc. DEVS Integrative M&S Symp.*

[Zeigler, 2013] Zeigler, B. P. (2013). *Guide to Modeling and Simulation of Systems of Systems - User's Reference*. Springer Briefs in Computer Science. Springer.

[Zeigler et al., 1999] Zeigler, B. P., Ball, G., Cho, H., Lee, J., and Sarjoughian, H. (1999). Implementation of the DEVS formalism over the HLA/RTI : Problems and solutions. In *Simulation Interoperation Workshop (SIW)*, number 99S-SIW, page 065.

[Zeigler et al., 1998] Zeigler, B. P., Cho, H., Lee, J., and Sarjoughian, H. (1998). The DEVS/HLA distributed simulation environment and its support for predictive filtering. *DARPA Contract N6133997K-0007 : ECE Dept., UA, Tucson, AZ.*

Résumé

Ce travail de thèse porte sur l'étude des systèmes complexes par une démarche de modélisation et simulation (M&S). La plupart des questionnements sur ces systèmes nécessitent de prendre en compte plusieurs points de vue simultanément. Il faut alors considérer des phénomènes évoluant à des échelles (temporelles et spatiales) et des niveaux de résolutions (de microscopique à macroscopique) différents. De plus, l'expertise nécessaire pour décrire le système vient en général de plusieurs domaines scientifiques. Les défis sont alors de concilier ces points de vues hétérogènes, et d'intégrer l'existant de chaque domaine (formalismes et logiciels de simulation) tout en restant dans le cadre rigoureux de la démarche de M&S. Pour répondre à ces défis, nous mobilisons à la fois des notions de modélisation multi-niveau (intégration de représentations micro/macro), de modélisation hybride (intégration de formalismes discrets/continus), de simulation parallèle, et d'ingénierie logicielle (interopérabilité logiciel, et ingénierie dirigée par les modèles). Nous nous inscrivons dans la continuité des travaux de M&S existants autour de l'approche AA4MM et du formalisme DEVS. Nous étudions en effet dans cette thèse en quoi ces approches sont complémentaires et permettent, une fois combinées dans une démarche d'Ingénierie Dirigée par les Modèles (IDM), de répondre aux défis de la M&S des systèmes complexes. Notre contribution est double. Nous proposons d'une part les spécifications opérationnelles de l'intergiciel de co-simulation MECSYCO permettant de simuler en parallèle un modèle de manière rigoureuse et complétement décentralisée. D'autre part, nous proposons une approche d'IDM permettant de décrire de manière non-ambiguë des modèles, puis de systématiser leur implémentation dans MECSYCO. Nous évaluons les propriétés de notre approche à travers plusieurs preuves de concept portant sur la M&S du trafic autoroutier et sur la résolution numérique d'un système d'équations différentielles.

Mots-clés: multi-modélisation, co-simulation, système complexe, multi-agent, méta-modélisation, ingénierie dirigée par les modèles, formalisme DEVS.

Abstract

This thesis is focused on the study of complex systems through a modeling and simulation (M&S) process. Most questions about such systems requiere to take simultaneously account of several points of view. Phenomena evolving at different (temporal and spatial) scales and at different levels of resolution (from micro to macro) have to be considered. Moreover, several expert skills belonging to different scientific fields are needed. The challenges are then to reconcile these heterogeneous points of view, and to integrate each domain tools (formalisms and simulation software) within the rigorous framework of the M&S process. In order to solve these issues, we mobilise notions from multi-level modeling, hybrid modeling, parallel simulation and software engineering. Regarding these fields, we study the complementarity of the AA4MM approach and the DEVS formalism into the scope of the model-driven engineering (MDE) approach. Our contribution is twofold. We propose the operational specifications of the MECSYCO co-simulation middleware enabling the parallel simulation of complex systems models in a rigorous and decentralized way. We also define an MDE approach enabling the non-ambiguous description of complex systems models and their automatic implementation in MECSYCO. We show the properties of our approach with several proofs of concept.

Keywords: multi-modeling, co-simulation, complex system, multi-agent, meta-modeling, model-driven engineering, DEVS formalism.