UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Modélisation de Systèmes Complexes par Composition :

Une démarche hiérarchique pour la co-simulation de composants hétérogènes

THÈSE

présentée et soutenue publiquement le 28/05/2019

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Thomas PARIS

Composition du jury

Président :	Claude GODART	
Rapporteurs :	Claudia FRYDMAN	Professeur, Université d'Aix-Marseille, Marseille
	Stéphane GALLAND	Professeur, Université de Technologie de Belfort-Montbéliard, Belfort
Examinateurs :	Anne-Cécile ORGERIE	Chargée de Recherche, CNRS, Rennes
	Claude GODART	Professeur, Université de Lorraine, Nancy
	Laurent CIARLETTA	Maître de Conférences, Université de Lorraine, Nancy
Directeur de thèse :	Vincent CHEVRIER	Professeur, Université de Lorraine, Nancy

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

Mis en page avec la classe thesul.

Remerciements

Mes premiers remerciements vont à Vincent pour sa confiance, sa disponibilité et son soutien durant ces 4 années de recherche. Je remercie aussi Laurent pour son co-encadrement qui, bien que non officiel, a été néanmoins très bénéfique, et Christine pour sa relecture minutieuse de mon manuscrit.

Je tiens à remercier mes deux rapporteurs, Claudia FRYDMAN et Stéphane GALLAND, pour leur relecture de mon manuscrit, leurs retours et leur participation à mon jury. Je remercie également Anne-Cécile ORGERIE et Claude GODART pour avoir accepté de faire partie de mon jury.

La possibilité de faire une thèse s'est présentée lors de mon stage de fin d'études au sein du LORIA. Encore hésitant à l'époque bien qu'intéressé par le sujet de recherche lui-même, c'est sans conteste l'environnement de travail et l'ambiance chaleureuse au sein de l'équipe qui m'ont incité à choisir cette voie. Pour cela, je souhaite remercier notamment l'ancien trio du bureau C024 (Benjamin, Julien et Yannick), pour leur bonne humeur, l'ambiance qu'ils apportaient dans l'équipe, et leur aide.

La richesse de l'expérience de thèse provient aussi des différentes personnes avec qui j'ai eu la chance d'échanger ou de travailler. Je pense ici, sans être exhaustif, aux membres de l'équipe SIMBIOT (Louis, Jean-Baptiste, Théo, Denis, Ye-Qiong, Virgile, Antoine, Paul, Enrico, Sylvain, Virginie, Françoise, Emmanuel), aux anciens collègues (Alexandre, Corentin, Raphaël), aux stagiaires MECSYCO (Thomas, Maxime, Paul, Grégoire...), mais aussi aux membres de l'APHEEN (Bernard et François).

La thèse ne fut pas qu'un travail de recherche, ce fut aussi une expérience d'enseignement, et je tiens à exprimer ma reconnaissance envers les équipes pédagogiques et administratives de l'IUT Charlemagne puis de l'ENSEM pour leur aide et leurs conseils.

Enfin, je tiens à remercier mes parents qui m'ont donné le goût des études, ma grande famille qui est une source inépuisable de soutien, ma marraine pour ses encouragements et ses conseils réguliers, les amis de longue date (en particulier Thomas, qui me suit depuis quelques années, et Germain sur qui j'ai toujours pu compter), et Clothilde pour m'avoir suivi dans l'Est et avoir été là au quotidien.

À ma famille et à mes amis qui m'ont toujours soutenu et encouragé.

Table des matières

Table des figures

1

Introduction

1.1 Contexte de la thèse

Les travaux présentés dans ce manuscrit s'inscrivent dans le contexte des nouveaux défis posés par la Modélisation et Simulation (M&S) de systèmes complexes. Notre objectif est la spécification d'une démarche structurée de M&S permettant de construire itérativement et hiérarchiquement des multi-modèles à partir de modèles hétérogènes qui sont ensuite co-simulés. En complément, nous avons développé un environnement logiciel, basé sur l'utilisation de langages dédiés, qui propose diverses fonctionnalités pour soutenir notre démarche de multi-modélisation hiérarchique et de co-simulation.

Nos contributions viennent enrichir les travaux existants autour de l'intergiciel MECSYCO [Camus, 2015].

1.1.1 Modélisation et simulation

La M&S consiste à étudier des systèmes naturels ou artificiels dans le but de **comprendre** leur fonctionnement et/ou de **prévoir** leurs comportements. C'est à la fois un outil de compréhension du monde qui nous entoure et de conception de nouveaux objets [Ramat, 2006].

L'activité de modélisation commence par l'identification d'un système naturel ou artificiel, existant ou en voie de conception, qui sera le sujet de l'étude. Elle conduit à la création d'un modèle, i.e. une représentation abstraite simplifiée du système que nous souhaitons étudier. Ce modèle va alors servir de substitut au système réel pour répondre à des questions sur celui-ci [Minsky, 1965]. Nous nous intéressons plus particulièrement aux modèles dynamiques, où la simulation consiste à interroger le modèle en déroulant son comportement au cours du temps à l'aide d'un simulateur. Cette simulation revient à réaliser une expérience sur un modèle plutôt que sur le système réel. Il faut ensuite vérifier que les résultats obtenus via le modèle sont bien, soit ceux attendus (si c'est un travail de conception), soit proches de ceux fournis par le système réel (si c'est un travail de compréhension).

Avec les progrès de l'informatique, la M&S se fait de plus en plus à l'aide d'ordinateurs. Les modèles sont alors implémentés sur machine et la simulation consiste en l'exécution d'un programme. De plus en plus de sciences (biologie, sciences sociales, physique des bâtiments, etc) se servent alors de la modélisation et simulation informatique comme alternative à des expériences qui peuvent être coûteuses, non éthiques ou trop complexes à mettre en place [Zeigler et al., 2012]. Un exemple courant est celui du feu de forêt, il vaut généralement mieux étudier sa propagation par simulation informatique plutôt que par une expérience réelle.

Avant toute expérience par simulation, il faut construire un modèle. Cette étape de création est guidée par une ou plusieurs interrogations qui peuvent être par exemple :

1. Quelle sera la trajectoire d'un boulet de canon ? Pouvons-nous négliger les forces de frottements de l'air et obtenir un point de chute précis au mètre près ?
2. Quelle sera la consommation électrique dédiée au chauffage d'un bâtiment ? Quelles sont les consommations des différents systèmes de chauffage envisageables ? Pouvons-nous négliger l'influence des bâtiments adjacents ?

C'est seulement au travers de ces questions que les modélisateurs pourront délimiter le système cible à étudier, puis effectuer des hypothèses sur ce système pour en concevoir un modèle. Il faut garder en tête qu'un modèle dépend de comment le problème a été posé et des hypothèses qui ont été effectuées. Chacun d'eux doit donc être accompagné d'un cadre d'utilisation qui spécifie ce qu'il représente et son domaine de validité.

Comme pour toute expérience scientifique, la création d'un modèle et sa simulation suit un protocole précis et rigoureux que nous verrons Figure 2.2. Cela nous amène à deux autres aspects essentiels du travail de M&S : la vérification et la validation. La vérification consiste à vérifier que l'implémentation correspond bien au modèle conçu par le modélisateur et qu'elle n'introduit pas d'erreur. La validation consiste à vérifier que les résultats produits par le modèle lors de sa simulation correspondent bien au système étudié [Zeigler and Sarjoughian, 2002]. Enfin, il est important que les résultats d'une simulation soient reproductibles car sans cette reproductibilité il ne peut y avoir de confiance dans les résultats produits.

Il existe de nombreux logiciels et méthodes spécialisés pour répondre aux besoins des modélisateurs dans des domaines particuliers (e.g. Modelica [Fritzson and Bunus, 2002] pour les systèmes continus basés sur des équations différentielles du premier ordre, ns-3 [Henderson et al., 2006] ou OMNet++/Inet [Varga and Hornig, 2008] pour les réseaux IP, NetLogo [Wilensky, 1999] ou GAMA [Grignard et al., 2013] pour les simulations multi-agents situés...). Ces outils simplifient le travail de modélisation pour les experts domaines, notamment en facilitant, voire en automatisant, le travail d'implémentation. De plus, les logiciels de M&S proposent aussi généralement nombre de fonctionnalités et d'outils dédiés aux différentes étapes de l'activité de M&S, dont la conception du modèle, sa vérification, l'analyse des résultats etc. En ce sens, le développement et l'enrichissement d'outils est une part non négligeable du domaine de la Modélisation et Simulation.

1.1.2 Systèmes complexes

En M&S, un "système complexe" désigne un système composé d'un grand nombre d'entités hétérogènes en interaction et dont le comportement émerge de l'interaction entre ces entités [Ramat, 2006]. Les mouvements collectifs (nuées d'oiseaux, mouvements de foules, ...) sont un exemple de systèmes complexes où des comportements individuels, au niveau microscopique, amènent à l'observation, au niveau macroscopique, de phénomènes collectifs qui influencent en retour chaque individu. L'étude de tels systèmes nécessite de se poser des questions sur plusieurs niveaux (individuel, collectif). Les réseaux électriques intelligents ou *smart-grids* sont un autre exemple de systèmes complexes. Ce sont des réseaux électriques dont le fonctionnement est optimisé (en terme de consommation, qualité de service...) en les couplant à des réseaux de télécommunication et à des systèmes de commande. Ils illustrent une autre facette des systèmes complexes, ils sont généralement pluridisciplinaires.

La multiplicité des entités et leur degré d'interaction, dont les interactions entre différents niveaux ou domaines, rendent les systèmes complexes difficiles à modéliser. Il est nécessaire de

hiérarchiser et d'organiser les différentes entités, de différentes manières, pour pouvoir appréhender le système dans sa globalité [Lane, 2006]. Cela entraine que l'étude d'un système complexe doit se faire sous plusieurs perspectives différentes mais complémentaires [Seck and Honig, 2012]. La difficulté est alors de réconcilier les différents points de vue, des différents experts, de manière cohérente au sein de la démarche de M&S.

1.2 Challenges et contributions

La modélisation des systèmes complexes, organisés sur plusieurs niveaux, composés d'entités associées à des domaines variés, est un challenge qui nécessite de nouvelles approches rassemblant les différents points de vue au sein d'un même modèle cohérent. C'est un challenge interdisciplinaire dont le point central est l'utilisation de l'outil informatique comme support.

Pour répondre à ce challenge, nous nous plaçons dans le cadre de deux approches prometteuses qui permettent de rassembler plusieurs perspectives complémentaires au sein d'un même projet de M&S : la multi-modélisation et la co-simulation [Gomes et al., 2018]. La première consiste à construire des modèles particuliers pour chaque perspective d'étude puis de les coupler au sein d'un multi-modèle qui représente le système global. La seconde consiste à simuler séparément les modèles tout en assurant l'échange de données et la synchronisation entre leurs simulateurs. Le premier défi est d'assurer que les modèles peuvent être couplés et que leurs simulateurs peuvent interagir de manière cohérente.

La M&S étant une science expérimentale, le développement d'un modèle qui répond au problème posé ne se fait pas directement et il faut généralement tester différentes alternatives, les vérifier, ajouter ou lever certaines hypothèses etc. Dans un multi-modèle, la multiplicité des entités entraine une multiplicité des sous-modèles qu'il faut organiser, coupler, remplacer pour tester ces différentes alternatives. Il faut pouvoir organiser le multi-modèle pour éviter qu'il devienne aussi complexe que le système qu'il doit représenter. Le deuxième défi est alors de proposer une démarche, i.e une suite structurée d'étapes, modulaire pour construire itérativement le multi-modèle en produisant, étape par étape, des multi-modèles intermédiaires qui serviront à construire le multi-modèle global qui sera co-simulé.

Ce manuscrit s'inscrit dans la continuité des travaux effectués sur l'intergiciel de co-simulation MECSYCO [Camus, 2015] qui propose des solutions pour intégrer des modèles hétérogènes issus de différents logiciels, rendre compatible leur représentation, et assurer leur co-simulation. Pour cela l'intergiciel se base sur le méta-modèle *Agent & Artifact for Multi-Modeling* (AA4MM) [Siebert, 2011] et sur le formalisme *Discrete EVent System specification* (DEVS) [Zeigler et al., 2000]. MECSYCO apporte des solutions pour permettre la co-simulation de modèles hétérogènes, mais ne propose pas de démarche de multi-modélisation associée. En particulier l'étape de multi-modélisation est limitée à un seul niveau, i.e. il n'est pas possible de construire itérativement des multi-modèles à partir de ceux déjà créés.

Notre contribution principale consiste alors à proposer une démarche de multi-modélisation et co-simulation hiérarchique allant de l'intégration des modèles (étape nécessaire pour une co-simulation) jusqu'à l'expérimentation sur le multi-modèle. Nous associons à cette démarche un environnement de multi-modélisation pour MECSYCO qui permet sa mise en œuvre. L'environnement est basé sur l'utilisation de langages dédiés associés aux différentes étapes de la démarche proposée. Nous apportons aussi plusieurs réflexions et contributions liées aux différentes étapes de l'activité de multi-modélisation et co-simulation.

1.3 Organisation du manuscrit

Les chapitres 2 à 5 détaillent l'état de l'art en présentant progressivement les notions des plus génériques aux plus spécifiques pour poser le cadre de la contribution. Les chapitres 6, 7 et 8 présentent ensuite les travaux et leur évaluation.

Le chapitre 2 reprend les bases de la Théorie de la Modélisation et Simulation, notamment les différentes étapes de création d'un modèle et les besoins de développement d'outils logiciels pour la M&S.

Le chapitre 3 introduit la problématique de M&S des systèmes complexes et les enjeux particuliers liés à ce type de système. Ce chapitre détaille notamment les approches de multi-modélisation et co-simulation et les besoins qu'elles soulèvent.

Le chapitre 4 concerne les approches et outils existants et permet de positionner notre travail. Il introduit notamment le formalisme DEVS et l'intergiciel MECSYCO qui servent de base à cette thèse.

Le chapitre 5 présente notre cadre de travail et notre problématique en détaillant l'état de l'intergiciel MECSYCO pour introduire plus spécifiquement notre proposition.

Le chapitre 6 détaille notre contribution principale, une démarche de multi-modélisation hiérarchique et de co-simulation et l'environnement de M&S associé basé sur des langages dédiés.

Le chapitre 7 est dédié à d'autres contributions plus spécifiques situées à différentes étapes de notre démarche de multi-modélisation et co-simulation. Il y est question de l'intégration d'un simulateur multi-agent au sein d'une co-simulation, de la définition de modèles sources qui viennent approvisionner les modèles pour tester leur dynamique, de l'ajout d'un mécanisme de couplage reliant plus de deux modèles et enfin du développement de moyens d'enregistrement de mesures de performance pour analyser les modèles et les simulateurs.

Le chapitre 8 présente deux expériences destinées à illustrer et à évaluer notre démarche de multi-modélisation hiérarchique et l'environnement proposé.

Enfin le chapitre 9 conclut ce travail et présente quelques perspectives.

2

Notions générales en Modélisation et Simulation

Ce premier chapitre va permettre de faire un tour d'horizon du domaine de la Modélisation et Simulation (M&S). Nous commençons par en présenter les notions fondamentales au travers de plusieurs définitions. Nous détaillons ensuite la démarche de M&S et ses différents enjeux dont le développement d'outils logiciels dédiés qui lui sont dédiés.

2.1 Introduction

La M&S consiste en l'étude de systèmes naturels (e.g. des nuées d'oiseaux) ou artificiels (e.g. un réseau électrique intelligent) dans le but de **comprendre**, **analyser** leur fonctionnement interne ou de **prévoir** leur évolution [Ramat, 2006]. Cette étude sera guidée par une interrogation, un objectif de modélisation, et sera réalisée à l'aide d'un modèle qui va servir de substitut au système cible pour la réalisation d'expériences. Il faut toujours garder à l'esprit que le modèle n'est qu'un moyen pour atteindre un but [Delinchant, 2011].

Le domaine de la M&S est au croisement de nombreuses disciplines, des sciences sociales avec des modèles de population, de mouvements de foule, de comportements routiers ; aux sciences naturelles ou physiques avec des modèles de croissance de plantes, de thermique des bâtiments, de réseaux électriques... Dans chacune de ces disciplines, l'activité de M&S est effectuée en utilisant des théories, des outils et des vocabulaires variés. Cependant, même si les systèmes étudiés sont différents, la Théorie de la M&S regroupe tout un ensemble de concepts, de démarches et de pratiques partagés par toutes les disciplines expérimentales. L'objectif de ce chapitre est de présenter ces éléments communs.

Dans notre cas, bien que restant très général, nous nous intéressons plus particulièrement 1) aux modèles dynamiques, i.e. des modèles qui représentent l'évolution du comportement de systèmes au cours du temps, et 2) aux modèles informatiques et aux expériences virtuelles, i.e. les simulations. La M&S est structurée en plusieurs étapes résumées Figure 2.1, nous les détaillons dans la suite.

2.1.1 La définition du système cible

La première étape consiste à fixer l'objectif que nous visons et le système cible que nous souhaitons modéliser. La notion de système est relativement ambigüe [Morin, 2015]. Dans un contexte général, un système peut se définir comme suit (Définition 1).

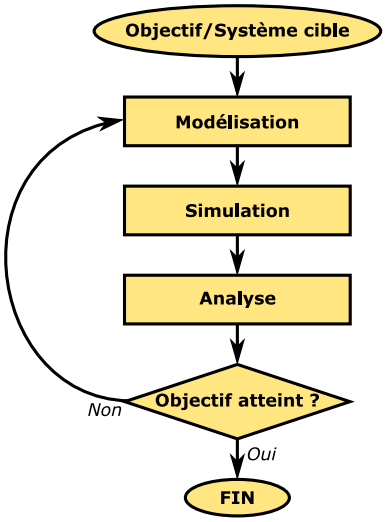


FIGURE 2.1 – Schéma simplifié des différentes étapes de la M&S. Inspiré de [Camus, 2015]

Définition 1 (Système)**.** Un système est un objet qui, dans un environnement, doté de finalités, exerce une activité et voit sa structure interne évoluer au fil du temps, sans qu'il perde pourtant son identité unique. [Le Moigne, 2006]

Cette définition très générale est applicable partout. Son intérêt est qu'elle met l'accent sur un ensemble de notions permettant d'identifier un système : un environnement, une finalité, une activité, une structure interne.

La notion d'environnement représente ici le contexte dans lequel nous identifions le système. Dans le cadre de la modélisation, la définition complète de cet environnement est un objectif en soit qui va déterminer le contexte dans lequel le modèle peut être utilisé [Traoré and Muzy, 2006]. Ce contexte est aussi appelé le cadre expérimental [Zeigler et al., 2000].

En M&S, nous définissons un système cible par l'intermédiaire des questions auxquelles le modélisateur souhaite répondre (Définition 2). La première étape est donc de déterminer les éléments qu'il faut prendre en compte pour répondre à la question posée, i.e. il faut en premier lieu identifier un système cible à étudier et le contexte dans lequel il évolue.

Définition 2 (Système cible)**.** Le système cible est l'ensemble des éléments concernés par l'interrogation. [Ramat, 2006]

Par exemple, si l'objectif est de faire des prévisions météorologiques et en particulier de prévoir les précipitations dans une région donnée, le modélisateur peut choisir de prendre en compte la pression atmosphérique, la vitesse des vents, la présence de nuages... Il fixe les éléments qu'il considère comme étant importants pour savoir s'il va pleuvoir ou non dans les prochaines heures. Il choisit alors d'ignorer d'autres éléments qui ne lui semblent pas pertinents.

Il est important de noter aussi que le modélisateur va définir les éléments à prendre en compte en fonction de ce qu'il est capable d'observer. Il doit alors déterminer les éléments influençant le système et comment ils impactent son comportement au cours du temps.

2.1.2 La création d'un modèle et simulation

Une fois que le système cible a été déterminé, un modèle (Définition 3) de celui-ci va être créé. Ce modèle sera une version abstraite, simplifié du système cible et devra reproduire son compor-

tement. Le modèle va donc contenir des variables/attributs qui représentent l'état du système cible, et des règles de comportement qui vont le faire évoluer au cours du temps notamment en fonction de son état interne et des influences qu'il subit [Zeigler et al., 2000].

Définition 3 (Modèle)**.** Pour un observateur B, un objet A* est un modèle d'un objet A dans la mesure où B peut utiliser A* pour répondre aux questions qu'il se pose sur A. [Minsky, 1965]

Les expériences seront réalisées sur le modèle plutôt que sur le système cible. Cela permet de conduire des expériences sur des systèmes auxquels nous n'avons pas accès (galaxies...), qui n'existent pas encore (prototype...), ou sur lesquels les expériences sont coûteuses ou non éthiques (étude de feux de forêt, de mouvements de foule...).

Les expériences sur les modèles sont appelées simulations (Définition 4). Pour chaque simulation, le modèle va être placé dans un état initial (correspondant par exemple à un état observé chez le système cible), puis simulé pour observer son évolution au cours du temps.

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

Une fois qu'il est possible de réaliser des simulations sur le modèle, les données qu'il fournit sont analysées pour vérifier leur cohérence avec les données observées sur le système cible, ou avec celles attendues. Cette analyse permet de vérifier la cohérence des résultats et de voir si le modèle permet bien de répondre à l'objectif fixé. En cas d'incohérence (due à une hypothèse erronée, à une erreur de formalisation ou d'implémentation etc) le modélisateur reprend son modèle pour le corriger ou le modifier.

2.2 Étapes de l'activité de M&S

L'activité de M&S, résumée Figure 2.1, peut en fait encore être découpée en plusieurs étapes précises. La Figure 2.2, inspirée des étapes identifiées par Galàn dans [Galán et al., 2009] pour la construction d'un modèle multi-agent, représente les différentes phases nécessaires à la création d'un modèle et à sa simulation. Bien que pensées dans le cadre des systèmes multi-agents, les étapes identifiées sont généralisables à toute activité de modélisation et simulation liée à l'informatique.

2.2.1 Système cible et conceptualisation

Comme précisé dans la section précédente, la première étape de l'activité de M&S est la détermination du système cible en fonction de l'objectif fixé. Nous pouvons nous représenter un système comme une boite avec des entrées et des sorties (Figure 2.3). La détermination du système cible impose de se poser plusieurs niveaux de questions :

- Quelles sont les entrées du système, i.e. quels éléments vont l'influencer ? Quelles sont les sorties du système, i.e. quelles variables peut-on mesurer sur le système ? Quels observations peut-on faire ? Sur quelles échelles de temps ?
- Comment sont liées les entrées du système avec ses sorties, i.e. quels sont les impacts des entrées du système sur les variables mesurées en sortie ?
- Peut-on associer un stimulus d'entrée à une unique réponse en sortie ? Comment stimuler le système de sorte à produire un comportement souhaité ?
- Peut-on identifier des sous-composants de notre système cible et leur organisation ?

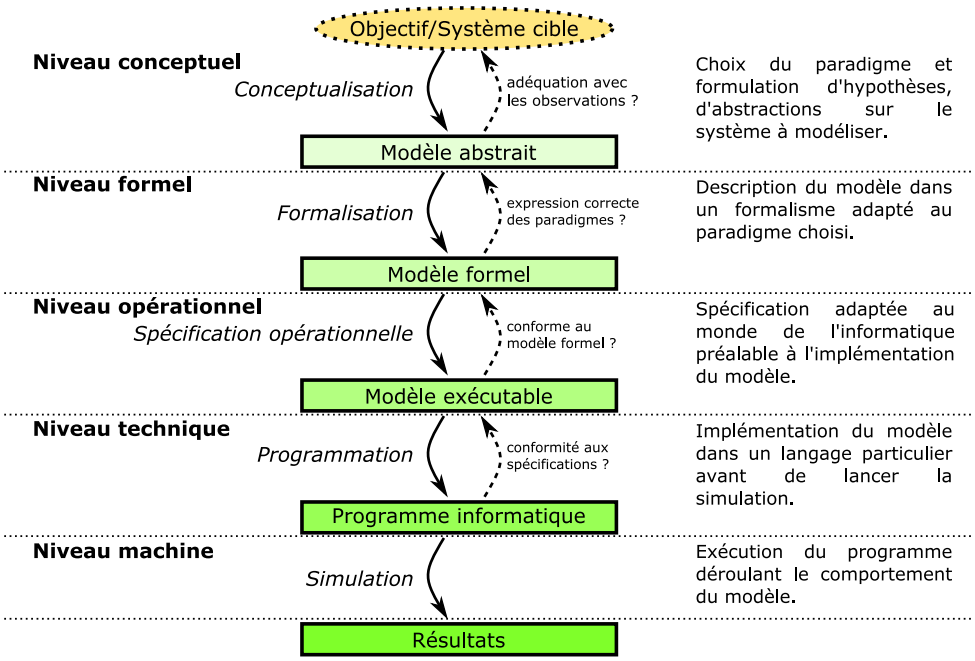


FIGURE 2.2 – Étapes de l'activité M&S, inspiré de [Galán et al., 2009]

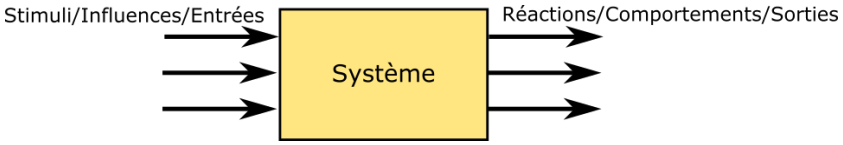


FIGURE 2.3 – Une vision du système

Cet ensemble d'interrogations va guider la définition du système cible. Les réponses à ces questions vont se formuler au travers d'un paradigme (Définition 5) particulier, i.e. le modélisateur aura à formuler des hypothèses et à faire des abstractions qui vont permettre l'élaboration du modèle [Le Moigne, 2006].

Définition 5 (Paradigme)**.** Un paradigme est un cadre de pensée composé d'un ensemble d'hypothèses fondamentales, de lois et de moyens sur la base desquels les modèles peuvent se développer. [Kuhn, 1972]

La notion de paradigme dépasse le cadre du modèle et englobe aussi les principes ou préceptes que le modélisateur cherchera à suivre pour construire le modèle. Le paradigme systémique par exemple s'accompagne d'une démarche de modélisation basée sur 4 préceptes (précepte de pertinence, précepte de globalisme, précepte téléologique et précepte d'agrégativité) qui définissent la manière dont le modélisateur doit considérer le système, le modèle et l'activité de modélisation en général [Le Moigne, 2006]. Nous pouvons citer aussi le paradigme multi-agent qui consiste à décomposer notre système en de multiple entités autonomes en interaction évoluant dans un environnement selon des règles précises.

Cette première étape de conceptualisation va mener à l'élaboration d'un **modèle abstrait**, généralement décrit en langage commun et rassemblant les différents choix de paradigme, hypothèses, abstractions effectués pour construire le modèle (voir Figure 2.2).

2.2.2 Formalisation

Le modèle abstrait regroupe l'ensemble des concepts, hypothèses et abstractions qui sont choisis pour construire le modèle. Ces idées sont généralement écrites en langage naturel et de manière ambigüe. Le choix d'un formalisme (Définition 6) permet d'écrire le modèle sous une forme mathématique univoque [Camus, 2015]. En ce sens, un formalisme peut être vu comme un outil d'expression des paradigmes [Quesnel, 2006].

Définition 6 (Formalisme)**.** Un formalisme est un langage mathématique formel permettant de spécifier la structure et les comportements de modèles, associé à un algorithme permettant de les simuler. [Sarjoughian, 2006]

D'après la définition, un formalisme est composé de deux parties :
— Une spécification de modèle, i.e. une théorie mathématique décrivant le type de structures et de comportements pouvant être représentés par le modèle.
— Un (ou plusieurs) algorithme d'exécution qui peut exécuter correctement tout type de modèle respectant la spécification.

C'est lors de cette étape de formalisation que sont spécifiées les variables du modèle et les différentes méthodes qui permettront de les faire évoluer. Un formalisme permet d'écrire tout un ensemble de modèles basés sur les mêmes hypothèses, construits avec le même type de structure, et pouvant être simulés avec le même type d'algorithme d'exécution ou simulateur (Définition 7).

Définition 7 (Simulateur)**.** Le simulateur est l'entité qui génère le comportement d'entrée sortie décrit par le modèle. [Duboz et al., 2012]

Le choix du formalisme est conditionné par les hypothèses et les abstractions faites au niveau du paradigme. Il faut bien sûr s'assurer que le modèle formel et son simulateur, écrits dans le formalisme choisi, expriment bien les hypothèses faites à l'étape de conceptualisation et seulement celles-ci. La Figure 2.4 présente une classification des formalismes en fonction des hypothèses faites sur la représentation des variables d'états, du temps, ou de l'espace.

Cette seconde étape mène à la création du **modèle formel** qui donne une représentation mathématique du système cible adaptée au raisonnement (Figure 2.2). Cette représentation décrit l'état du modèle (description statique) et la manière de faire évoluer cet état (description dynamique). Elle est généralement associée avec le **modèle abstrait** sous le terme de **modèle conceptuel** (Définition 8). Ce modèle conceptuel doit contenir l'ensemble des informations nécessaires à la réalisation de l'expérience (implémentation et simulation).

Définition 8 (Modèle conceptuel)**.** Le modèle conceptuel est une description du modèle indépendante d'un logiciel ou d'un langage particulier et qui décrit les entrées, les sorties, le contenu, les hypothèses et les simplifications du modèle. [Diallo et al., 2014] tiré de [Robinson, 2004]

2.2.3 Spécification opérationnelle et implémentation

Le modèle formel est une représentation mathématique du modèle qui n'est pas directement adaptée au monde de l'informatique. La spécification opérationnelle va effectuer la transition du modèle formel vers un modèle implémentable sur machine. Cela passe notamment par le choix de représentations des variables du modèle. Par exemple, les réels ne peuvent être représentés avec une précision infinie comme c'est le cas dans le modèle formel. Cela entraine des erreurs numériques [Galán et al., 2009] qu'il faudra prendre en compte par la suite lorsque les résultats

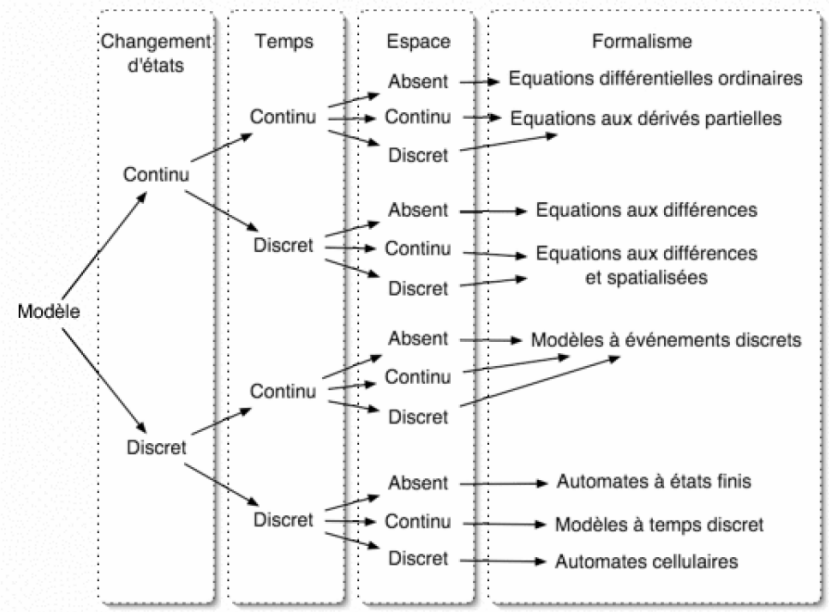


FIGURE 2.4 – Classification des formalismes. Source [Ramat, 2006]

du modèle seront analysés. La spécification opérationnelle est encore indépendante de toute implémentation, elle aboutit à la création du modèle exécutable (voir Figure 2.2).

Une fois cette spécification définie, il convient de passer à l'implémentation sur machine. À ce niveau, il est obligatoire de lever toutes les ambiguïtés pour écrire le programme représentant le modèle et son simulateur. Le modèle exécutable et le modèle implémenté forment ce que nous appelons le modèle numérique.

En parallèle de l'implémentation du modèle, il est nécessaire d'implémenter des fonctionnalités supplémentaires liées à :

— L'observation et la collecte de résultats : il est en effet impératif de pouvoir observer les résultats fournis par le modèle lors de la simulation et de les stocker en vue de leur analyse.
— Le déploiement : pour augmenter la vitesse de simulation des modèles lors de calculs parallèles.
— L'expérimentation : généralement un modèle n'est pas utilisé pour une seule simulation mais pour un ensemble de simulations visant à observer ou vérifier son comportement dans des cadres expérimentaux différents. Il est donc nécessaire de pouvoir simplement modifier les paramètres du modèle voire d'automatiser les tests de différents jeux de paramètres pour l'exploration du modèle.

2.2.4 Bilan

Cette section nous a permis d'introduire les 4 étapes de l'activité de M&S permettant de passer d'un système cible à un programme exécutable :

1. La conceptualisation qui part du système cible et qui aboutit à la formulation du modèle abstrait rassemblant les hypothèses et le cadre théorique.
2. La formalisation qui transforme le modèle abstrait et l'exprime dans un format mathématique non ambigüe, c'est le modèle formel.
3. La spécification opérationnelle, qui signe le passage vers le monde numérique.

4. La programmation qui fait le lien entre les spécifications opérationnelles et un exécutable qui permettra de lancer la simulation.

Il faut noter que nous présentons ici une démarche générique. Dans des domaines ou lors de l'utilisation d'approches particulières, d'autres démarches plus détaillées et spécifiques sont utilisées. Nous pouvons citer par exemple ASPECS [Cossentino et al., 2010] qui précise les différentes étapes permettant de représenter un système complexe dans une approche orientée agent, e.g. la réflexion débute alors par la définition du domaine d'étude puis par l'identification des différents rôles, organisations et règles qui régiront la société d'agents.

Un autre élément important à retenir est que chacune de ces étapes peut être effectuée par un individu différent, e.g. un théoricien peut s'occuper de la conceptualisation et de la formalisation tandis qu'un informaticien définira les spécifications opérationnelles et produira le code. L'un des challenges est donc de maintenir la cohérence entre chacune des étapes, ce qui nous amène à la notion de rigueur en science expérimentale.

2.3 Rigueur en science expérimentale

Cette section permet d'introduire plusieurs notions importantes en science expérimentale : la vérification, la validation et la reproductibilité des expériences. Ce sont ces différents aspects qui vont permettre de donner de la crédibilité aux résultats obtenus dans le cadre d'expériences.

Dans la suite de cette section, ces concepts vont tout d'abord être présentés dans le cadre général des sciences expérimentales puis seront placés dans le cas particulier des expériences numériques.

2.3.1 Vérification et validation

Nous nous intéressons dans cette partie aux processus de Vérification et de Validation (V&V) des modèles. Les modèles sont développés dans un but particulier, les processus de vérification et validation doivent permettre d'assurer que le modèle est effectivement utilisable pour répondre à l'objectif visé [Sargent, 2013]. Effectuer un ensemble de vérifications sur un modèle et valider son comportement permet aussi d'accroître la confiance que nous pouvons avoir dans ce modèle [Robinson, 1997], nous parlons alors de la **crédibilité** du modèle. C'est une notion fondamentale, car un modèle n'est jamais qu'une représentation abstraite, simplifiée et donc fausse du système cible. Il est impératif de savoir dans quel cadre ses résultats s'approchent de la réalité et avec quel niveau de précision. Cela est particulièrement important lorsque l'utilisateur du modèle n'est pas son concepteur et que le modèle est utilisé pour prendre des décisions.

Lors des différentes étapes de l'activité de M&S, le modèle passe par différents niveaux de représentation : conceptuel, formel, opérationnel, programme. Les transitions entre chaque niveau peuvent générer des erreurs [Galán et al., 2009], il est donc important de vérifier les concordances entre chaque niveau de représentation du modèle. Les activités de V&V sont variées et s'opèrent à chaque étape de l'activité de M&S. C'est notamment le cas en raison de l'évolution du questionnement sur le système qui se précise au fur et à mesure des travaux. Cela amène des modifications directement au niveau de la conception et ces modifications se propagent ensuite aux niveaux successifs [Robinson, 1997]. Dans la pratique, il y a souvent des aller-retours entre chaque niveau au fur et à mesure que notre compréhension du modèle évolue. La Figure 2.5, inspirée des schémas de [Galán et al., 2009] pour l'activité de M&S et de [Sargent, 2013] pour la V&V, place les activités de Vérification et Validation au sein de la démarche de Modélisation et Simulation.

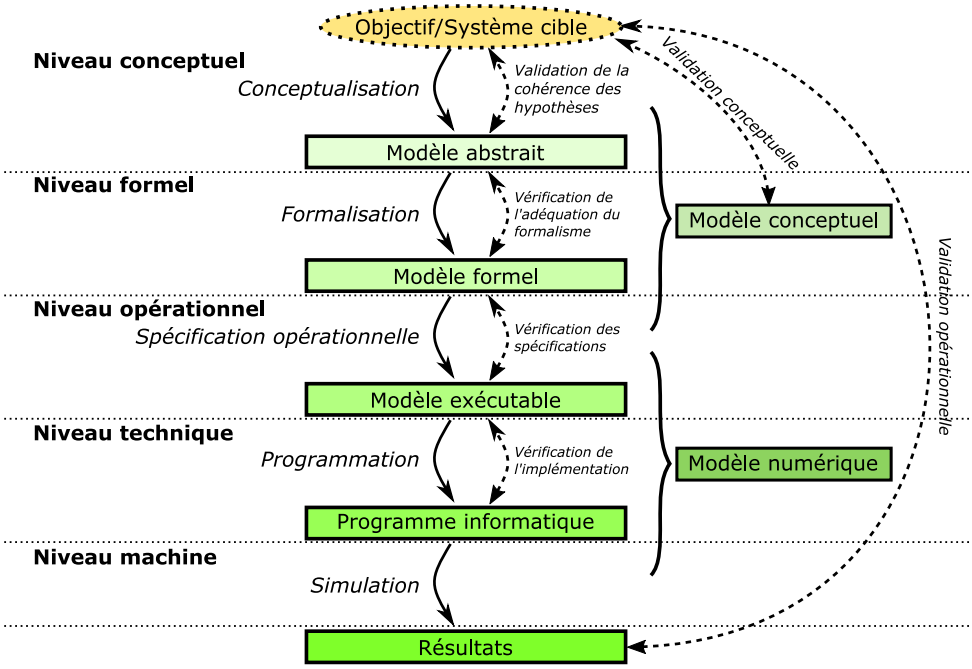


FIGURE 2.5 – Vérification et Validation dans l'activité de M&S. Inspiré de [Galán et al., 2009] et [Sargent, 2013]

Le modèle conceptuel rassemble les hypothèses choisies pour construire le modèle et la description formelle du modèle qui permet de le décrire de manière non ambiguë indépendamment des contraintes numériques. De son côté, le modèle numérique représente le passage au monde numérique et à l'implémentation concrète.

Dans ce cadre, la **Vérification** (Définition 9) est l'activité consistant à vérifier la conformité entre les spécifications et le modèle implémenté [Amblard et al., 2006]. Cette vérification d'adéquation du modèle numérique avec le modèle conceptuel se découpe en deux étapes : la vérification des spécifications qui s'occupe de la cohérence entre le modèle formel et les spécifications opérationnelles, et la vérification de l'implémentation qui s'assure que celle-ci est correcte.

Définition 9 (Vérification)**.** La vérification est le processus déterminant si le modèle implémenté et les données associées représentent bien la description conceptuelle et les spécifications formelles du modèle. [Zeigler and Sarjoughian, 2002]

De son côté, la **Validation** (Définition 10) consiste à vérifier l'adéquation entre le modèle et le système qu'il est censé représenter dans le cadre expérimental précis qui a conditionné sa création [Amblard et al., 2006]. La validation met en relation le modèle, le système cible et le cadre expérimental qui encadre la modélisation.

Définition 10 (Validation)**.** La validation consiste à déterminer si le modèle et les données associées rendent bien une représentation fidèle du système réel dans le cadre d'expérimentation et d'utilisation du modèle. [Zeigler and Sarjoughian, 2002]

Une première étape de validation, la validation conceptuelle, consiste à vérifier la cohérence des hypothèses qui doivent servir à construire le modèle, et si la représentation formelle du modèle permet bien de répondre à l'objectif de modélisation. L'étude de cohérence des hypothèses se fait en vérifiant par exemple qu'elles s'accordent bien avec les comportements observés chez le

système cible. La deuxième partie de la validation du modèle, la validation opérationnelle, permet de vérifier que les résultats produits par le programme lors des simulations s'accordent bien, i.e. avec la précision souhaitée, avec les résultats obtenus sur le système cible [Sargent, 2013]. Cette deuxième étape de validation met en jeu l'ensemble du processus de modélisation et simulation et permet notamment de donner de la crédibilité au modèle numérique.

Il existe de nombreux mécanismes formels ou empiriques permettant de vérifier, valider et d'augmenter la crédibilité d'un modèle, en voici ci-dessous une liste non-exhaustive :

— Preuve formelle au niveau du formalisme, des algorithmes de simulation ou du programme
— Comparaison des résultats avec les données obtenues sur le système
— Comparaison des résultats avec les données obtenues sur un autre modèle valide
— Analyse visuelle de la cohérence des résultats
— Analyse de sensibilité des paramètres
— ...

Il faut avoir conscience que l'activité de V&V est coûteuse en temps et peut s'avérer compliquée. Par exemple, lors du projet MS4SG (*Multi-Simulation for Smart-Grid*) [Vaubourg et al., 2015] qui résulte d'une collaboration entre Inria-LORIA et EDF R&D, les modèles de réseaux électriques intelligents étaient validés tout d'abord par une analyse visuelle de la cohérence des résultats puis par des comparaisons à un système réel (qu'il a fallu construire) sur des cas d'usage particuliers.

2.3.2 Reproduction des expériences

La reproductibilité est la capacité à reproduire une expérience et à obtenir les mêmes résultats. C'est l'une des bases de la recherche scientifique [Fomel and Claerbout, 2009]. Elle permet de valider et vérifier les protocoles expérimentaux ainsi que les résultats d'un chercheur, donnant ainsi de la crédibilité à ses travaux [Yilmaz et al., 2014]. Pour cela, les scientifiques doivent publier leurs travaux de sorte que d'autres chercheurs puissent reproduire leurs résultats, i.e. en précisant clairement le but de l'expérience, les résultats, le protocole et le cadre expérimental.

Le terme "reproductibilité" peut en fait être découpé en plusieurs aspects [Collberg et al., 2015] [Vitek and Kalibera, 2011] :

— La **répétabilité** qui correspond à la capacité à re-jouer la même expérience et à obtenir les mêmes résultats. En prenant le schéma 2.5, cela correspondrait à réussir la reproduction de l'expérience à partir du même modèle numérique.
— La **reproductibilité** qui est la capacité à obtenir les mêmes résultats en utilisant une expérience indépendante, i.e. dans les sciences expérimentales cela correspond à valider une hypothèse de deux manières différentes pour augmenter le poids de la preuve. En informatique, en reprenant le schéma 2.5, cela peut correspondre par exemple à reproduire une simulation à partir du modèle conceptuel, i.e. en utilisant une implémentation différente. La reproductibilité permet notamment de s'assurer que les résultats ne sont pas dus à un choix de conception au niveau de l'implémentation mais bien aux choix opérés au niveau conceptuel. Lorsqu'un modèle conceptuel n'est pas suffisamment spécifié, i.e. s'il reste des ambigüités, il est difficile de reproduire les résultats. Par exemple dans [Fatès and Chevrier, 2010], les expériences sur un modèle multi-agent particulier, le modèle multi-termites, montrent l'influence du schéma de mise à jour sur les résultats obtenus. Or cet élément est rarement spécifié dans la description du modèle qui se concentre généralement sur les règles de comportement des agents.

Dans le cas de la modélisation et simulation informatique, la répétabilité et la reproductibilité des expériences de simulation sont un problème dû notamment à la multiplicité, à l'hétérogénéité

et à l'évolution rapide des logiciels, des systèmes d'exploitation, et des environnements utilisés. De plus, les simulations peuvent être effectuées par des experts qui ne sont pas forcément compétents pour assurer la reproductibilité de simulations informatiques [Mesirov, 2010]. Ainsi, de nombreuses simulations publiées ces dernières années ne sont pas reproduites, et ne sont pas reproductibles (ni même répétables) [Taylor et al., 2013], certains parlent même de crise de la reproductibilité [Yilmaz and Ören, 2013].

2.4 Développement d'outils logiciels dédiés à la M&S

L'ensemble des étapes de l'activité de M&S présentée Figure 2.2 est coûteux en temps et en moyens. C'est pourquoi, l'un des enjeux actuels en M&S est le développement d'outils et de logiciels facilitant le développement des modèles, la réalisation des expériences virtuelles (les simulations) et l'analyse des résultats.

2.4.1 Des outils variés

Un grand nombre de logiciels de M&S spécialisés ont pu être implémentés grâce au fait qu'un même paradigme et un même formalisme peuvent être utilisés pour construire un grand panel de modèles qui répondent à des questions similaires. Tous fournissent des solutions à des besoins précis en matière de domaine d'étude, d'utilisation d'un formalisme particulier ou d'autres facteurs comme l'analyse et la visualisation des résultats.

Un outil de M&S peut aller de la bibliothèque logicielle proposant des éléments utiles pour la simulation, à un environnement de développement complet doté de nombreuses fonctionnalités.

2.4.2 Exemples d'outils logiciels dédiés à la M&S

Nous citons ci-dessous quelques outils de M&S et leurs fonctionnalités.

ns-3

ns-3 [Henderson et al., 2006] est un simulateur libre de réseaux de télécommunication qui contient une bibliothèque de modèles implémentés en C++.

OMNet++

OMNet++ [Varga and Hornig, 2008] est un autre simulateur qui permet de créer et de simuler des modèles de réseaux de télécommunication. Cet outil contient lui-aussi une bibliothèque de modèles/composants dédiés à l'étude de réseaux ainsi qu'un simulateur pour analyser leurs dynamiques, mais il fournit en plus un environnement graphique de modélisation.

MatLab/Simulink

MatLab est à l'origine un langage de programmation optimisé pour le calcul matriciel, associé à un environnement logiciel. Son extension Simulink permet la réalisation de modèles de systèmes multi-physiques en associant des composants de calcul. MatLab/Simulink dispose d'un environnement graphique de modélisation, et propose aussi des fonctionnalités avancées pour le calcul.

Modelica

Modelica [Fritzson and Bunus, 2002] est un langage dédié à la M&S de systèmes équationnels. Il permet d'écrire des modèles multi-physiques dans un format très proche des mathématiques (des équations différentielles), et de les simuler en utilisant des algorithmes de résolution approchée (Euler, Runge-Kutta...). Ce langage est accompagné d'une blbliothèque de modèles et est utilisé dans plusieurs outils comme JModelica [Andersson, 2017], OpenModelica [Fritzson et al., 2005], ou Dymola [Brück et al., 2002]. Ces deux derniers proposent en plus un environnement graphique de modélisation, des outils pour la visualisation etc.

Et beaucoup d'autres

Il est impossible d'être exhaustif et d'énumérer tous les logiciels de M&S existants. Cependant, pour appréhender leur diversité nous pouvons énoncer brièvement GAMA [Grignard et al., 2013], NetLogo [Wilensky, 1999] et Repast [North et al., 2013] pour les simulations orientées multi-agents ; Gazebo [Koenig and Howard, 2004] pour les simulations en robotique ; EMTP-RV [1] pour la simulation de systèmes électriques...

2.4.3 Éléments de développement d'outils de M&S

L'implémentation d'un outil de M&S nécessite de réfléchir préalablement aux types de modèles que nous souhaitons pouvoir créer, à la manière de les représenter formellement, de les implémenter... Cela requiert de se poser des questions aux quatre niveaux de l'activité de M&S. Les choix effectués vont alors délimiter les modèles qu'il sera possible de réaliser (en imposant des hypothèses, des abstractions, un ou plusieurs formalismes et simulateurs etc).

Plus un logiciel de M&S est spécialisé, plus il est contraint, mais cela lui permet aussi de proposer plus de fonctionnalités et donc de faire gagner du temps aux modélisateurs. Par exemple, les contraintes de l'environnement de développement permettent de limiter les erreurs lors de l'implémentation des modèles.

En fonction des possibilités qu'ils offrent, les logiciels de M&S peuvent couvrir différents niveaux de l'activité de M&S. Par exemple, si nous nous plaçons dans le domaine de la thermique des bâtiments, l'environnement OpenModelica associé à la bibliothèque BuildSyspro [Plessis et al., 2014] nous fournit un moyen de passer d'une écriture proche du modèle formel (des équations différentielles), à une simulation et aux résultats. Cela peut se voir schématiquement comme suit (voir Figure 2.6).

Un dernier élément à prendre en considération est que le développement d'un logiciel de M&S ne peut s'abstraire des techniques d'architecture logicielle [Sarjoughian and Singh, 2004], notamment dans le but de fournir des solutions pour l'observation, l'analyse de résultat, la V&V, la reproductibilité et l'ensemble des fonctionnalités liées à l'activité de M&S. Un choix d'architecture logicielle judicieux est fondamental pour le maintien de l'outil et son extensibilité, i.e. sa capacité à proposer de nouvelles fonctionnalités et à répondre à de nouveaux problèmes ou enjeux.

2.4.4 Bilan

Le développement de nouveaux outils logiciels ou l'amélioration de ceux existants occupe une place importante dans le domaine de la M&S. Ils ont pour objectif de diminuer les temps de

1. https://www.emtp-software.com/page/overview

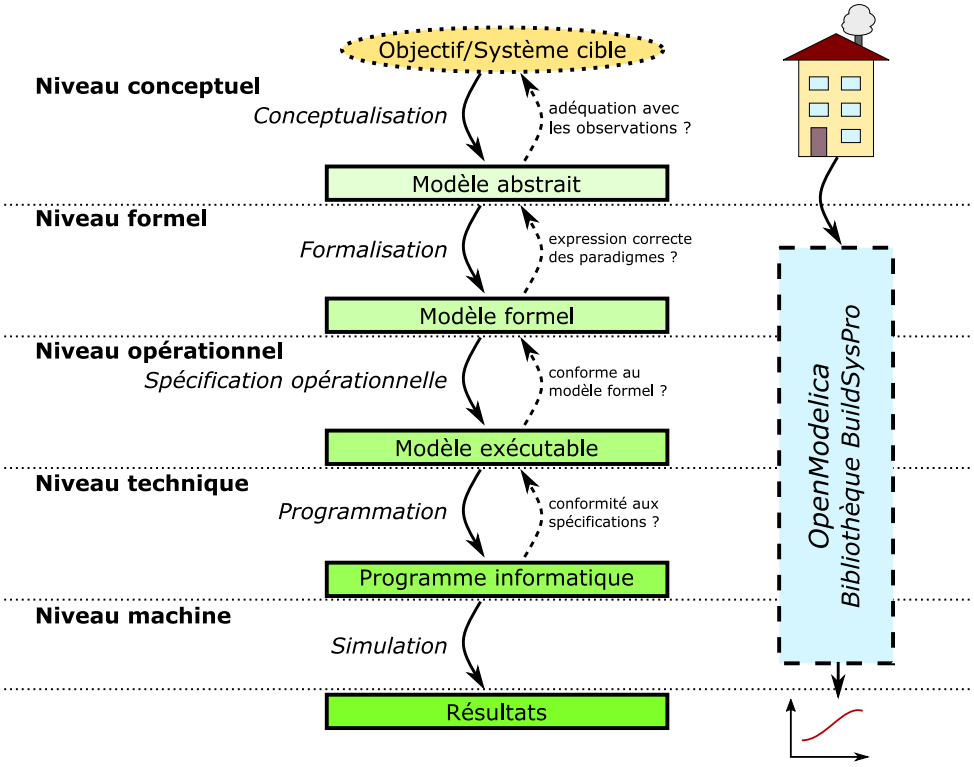


FIGURE 2.6 – Exemple de positionnement d'un logiciel de M&S spécialisé (OpenModelica) dans les étapes de l'activité de M&S

développement ainsi que les coûts en proposant à des degrés divers des fonctionnalités avancées de développement, de simulation et d'analyse.

L'utilisation d'un outil est donc d'une grande aide pour les modélisateurs, mais cela peut aussi devenir une contrainte car en ajoutant une couche d'abstraction nécessaire à l'amélioration du développement de modèles, les logiciels de M&S deviennent en partie ou complètement responsables :

1. de la vérification du modèle (par exemple s'il est généré à partir d'une interface graphique, c'est la génération du code qui doit être vérifiée),
2. de sa reproductibilité (certains choix d'implémentation du logiciel peuvent-ils empêcher ou complexifier la reproduction d'une expérience ? Les spécifications du logiciel sont-elles accessibles ?),
3. de sa réutilisabilité (le modèle peut-il être réutilisé en collaboration avec d'autres modèles ? Dans l'outil ? En dehors de l'outil ?).

Ainsi, le développement de nos modèles dans un outil spécialisé nous contraint généralement à cet environnement logiciel particulier et aux outils qu'il propose. Cela signifie que si nous voulons prendre en compte de nouveaux éléments du système cible dans notre modèle (pour relaxer une hypothèse, gagner en précision ou explorer un aspect qui avait été négligé à priori), ceux-ci devront se conformer aux hypothèses du logiciel actuel. Si c'est impossible, il faudra alors choisir un nouvel environnement de modélisation plus permissif dans lequel ré-implémenter le modèle et y intégrer les nouveaux éléments.

Les environnements de M&S permettent de capitaliser les connaissances et les expériences sur un domaine pour gagner en temps, en qualité et diminuer les coûts, mais ceci peut se payer

par une perte de souplesse et l'impossibilité de prendre en compte des éléments sortant du cadre pré-établi.

2.5 Conclusion

Ce premier chapitre nous a permis de faire un tour d'horizon des définitions et des concepts importants dans le domaine de la Modélisation et Simulation.

Nous avons notamment vu que la M&S est un domaine où on cherche à étudier des systèmes naturels ou artificiels, existants ou encore en conception. Cette étude est guidée par un ou plusieurs objectifs précis et s'effectue au travers de la réalisation d'un modèle, une représentation abstraite du système, sur lequel seront effectuées des simulations. Ces simulations sont des expériences virtuelles visant à obtenir un tracé de l'évolution du comportement du modèle au cours du temps.

Le passage d'un système cible à un modèle puis à sa simulation doit suivre une démarche rigoureuse. En tant que science expérimentale, plusieurs notions fondamentales ne doivent pas être oubliées. Tout d'abord, la vérification qui consiste à contrôler que le travail d'implémentation, et plus généralement le modèle numérique, respecte bien le modèle conceptuel. Ensuite la validation, qui consiste à vérifier la cohérence entre le modèle et le système réel. Enfin, la reproductibilité est un concept essentiel puisqu'il est impossible de tirer de conclusion d'une expérience que nous ne pouvons pas reproduire. Ces notions permettent d'augmenter la crédibilité du modèle.

L'ensemble des étapes de l'activité de M&S est coûteux en temps et en moyens. C'est ce qui a conduit au développement de nombreux outils de M&S répondant à des besoins précis et allant de la simple bibliothèque logicielle à un environnement de développement complet. Ces outils, en ajoutant une couche d'abstraction nécessaire à l'amélioration du développement de modèles, deviennent, en partie ou complètement, garants :

1. de la vérification du modèle,
2. de sa reproductibilité, et
3. de sa réutilisabilité.

Nous avons présenté dans ce chapitre les éléments généraux qui nous permettent de construire un modèle. Ces éléments sont suffisants lorsque nous nous intéressons à un domaine d'étude ou à un formalisme particulier. De fait, le cloisonnement des sciences en disciplines spécialisées entraine la création d'outils théoriques comme logiciels de plus en plus performants sur des domaines de plus en plus restreints. Dans le prochain chapitre, nous nous intéresserons à la notion de systèmes complexes et aux challenges que pose leur modélisation qui requiert de coupler plusieurs modèles issus de disciplines, formalismes et même outils logiciels variés.

3

Le cas des Systèmes Complexes : Multi-modélisation, Co-simulation et Composition

Ce second chapitre introduit la notion de systèmes complexes et les défis liés à leur modélisation. Ces systèmes sont composés d'un grand nombre d'éléments en interaction parfois difficiles à identifier. Ils nécessitent de combiner différentes approches, perspectives et disciplines ce qui se traduit par l'utilisation de plusieurs modèles en interaction qu'il faut organiser et hiérarchiser afin de comprendre et de représenter le système dans sa globalité.

Cela nous amènera à la multi-modélisation et à la co-simulation qui cherchent à répondre à ces besoins. Les difficultés sont alors liées aux hétérogénéités des différentes approches et outils utilisés, c'est un défi d'*interopérabilité*[2] à tous les niveaux de l'activité de M&S.

Modéliser des systèmes complexes à l'aide d'une approche multi-perspective basée sur la multi-modélisation et la co-simulation nécessite alors :

1. de trouver une solution générique au problème d'*interopérabilité*,
2. tout en permettant la nécessaire hiérarchisation des modèles, et
3. d'y associer un environnement logiciel fournissant des propriétés avancées de modélisation.

Nous commencerons dans ce chapitre par présenter les différents besoins liés à la modélisation des systèmes complexes. Nous verrons ensuite comment la multi-modélisation et la co-simulation tentent de répondre à ces besoins et les problématiques d'*interopérabilité* qu'elles entraînent avant de faire un récapitulatif des challenges.

3.1 La complexité en M&S

3.1.1 La notion de complexité

Dans le langage courant, le terme "complexe" renvoit à un problème compliqué, difficile à résoudre ou difficile à comprendre. Dans le cadre de la modélisation et simulation, la complexité fait référence à un ensemble d'éléments hétérogènes reliés entre eux, interagissant de manière à rendre difficile l'identification de chacun des éléments, de leurs comportements et des relations d'action et de rétroaction qui les lient [Morin, 2015]. De cette vision de la complexité découle la Définition 11 d'un système complexe.

2. Le terme "*interopérabilité*" a ici un sens particulier détaillé en 3.3

Définition 11 (Système complexe)**.** Un système complexe est un système composé d'un grand nombre d'entités hétérogènes en interaction et dont le comportement global émerge de l'interaction entre ses composants. [Ramat, 2006]

Une colonie de fourmis est un exemple de système complexe biologique où le comportement global de la colonie émerge du comportement de chaque fourmi individuelle et des interactions entre elles et avec leur environnement. Les systèmes urbains sont aussi des systèmes complexes composés d'entités issues de différents niveaux d'organisation (villes, quartiers, habitations) en interaction [Gil-Quijano et al., 2012].

3.1.2 Les systèmes cyber-physiques

Les systèmes cyber-physiques (CPS), tels que les réseaux électriques intelligents sont un autre exemple de systèmes complexes. Un système cyber-physique est un système composé d'un ou plusieurs systèmes physiques supervisés par des systèmes informatiques [Rajkumar et al., 2010]. Les réseaux électriques intelligents sont des exemples de CPS où un réseau électrique est supervisé par des systèmes informatiques de contrôle et de décision dans le but d'optimiser la consommation énergétique [Vaubourg, 2017]. Cela implique notamment la présence de capteurs et d'effecteurs qui permettront respectivement de capter l'état du système physique et d'agir sur lui. L'ensemble des informations et des commandes transitent alors via les réseaux de télécommunications [Vaubourg et al., 2016].

3.1.3 Étudier les systèmes complexes

Nous détaillons ci-dessous deux besoins associés à l'étude des systèmes complexes : la nécessité d'associer plusieurs points de vue pour comprendre la dynamique globale, et la nécessité de hiérarchiser les éléments du système pour l'appréhender.

Nécessité d'associer différents points de vue

Pour reprendre l'exemple des CPS, l'étude de leur dynamique globale nécessite de prendre en compte plusieurs perspectives amenées par différents experts. Dans le cas des réseaux électriques intelligents, l'expert en génie électrique pourra par exemple s'intéresser aux différents éléments composant le réseau, à la présence de sources d'énergie ponctuelles, à la gestion des pics de consommation et à leur répartition etc. L'expert en réseaux de télécommunication étudiera de son côté les types de protocole à mettre en place pour transférer les informations des capteurs aux systèmes de décision et des systèmes de décision aux effecteurs, vérifiera les délais, la charge nécessaire... Enfin l'expert en système de décision se positionnera sur la manière de contrôler et de s'adapter aux évolutions du réseau électrique et aux propriétés qu'il est possible d'espérer. L'étude du réseau électrique intelligent va donc se faire selon trois perspectives différentes mais interconnectées. En outre, chacun devra pouvoir tester différentes alternatives sur son domaine d'étude et étudier son impact sur le système global.

Il est nécessaire de combiner ces différents points de vue pour étudier le système dans sa globalité. Plus généralement l'étude de systèmes complexes nécessite une modélisation multi-perspective [Seck and Honig, 2012], i.e. prenant en compte plusieurs niveaux de détail [Gaud et al., 2008] et plusieurs domaines. Cette nécessité d'étudier les systèmes selon plusieurs points de vue amène la nécessité de les organiser de différentes façons pour appréhender leur complexité.

Nécessité de hiérarchiser

La notion de complexité est intimement liée à la nécessité de hiérarchiser [Herbert A., 1962]. Face à un système complexe, composé de multiple entités reliées entre elles par des relations hétérogènes parfois difficiles à identifier, il apparait nécessaire de regrouper, d'organiser, de hiérarchiser ces entités pour pouvoir appréhender le système global. Ces organisations sont nécessaires pour clarifier les phénomènes et les rendre intelligibles [Morin, 2015].

Il est possible d'organiser des éléments entre eux de différentes manières. Plusieurs d'entre elles sont présentées dans [Lane, 2006] dont :

1. La hiérarchie d'ordre : Consiste à ordonner des éléments suivant un indice. Par exemple, nous pouvons ordonner des étudiants en fonction de leur note.
2. La hiérarchie d'inclusion : Les poupées russes (*matriochkas*), où chaque poupée contient une autre poupée qui contient une autre poupée etc, sont un bon exemple de hiérarchie d'inclusion.
3. La hiérarchie de contrôle : Hiérarchie de type militaire où un élément de niveau supérieur contrôle les éléments de niveaux inférieurs, ses subordonnés.
4. La hiérarchie de niveau : Hiérarchie qui consiste à organiser les éléments par niveaux, spatio-temporel par exemple. Pour les systèmes biologiques, nous pouvons nous placer au niveau cellulaire et identifier différents types de cellules et leurs interactions. Nous pouvons nous placer au niveau des tissus composés de cellules et étudier leurs types et leurs interactions. Et enfin nous pouvons nous placer au niveau des organes composés de tissus et identifier leurs types et interactions etc [Herbert A., 1962].

Chaque organisation vue précédemment se base sur un attribut spécifique des éléments, un rôle, une fonction ou un lien structurel, et permet de mettre en lumière différentes relations entre les entités. Le choix du type de hiérarchie se fera en fonction du type de système étudié, il est évidemment possible de combiner des organisations. Il est également possible, voire même nécessaire, de proposer plusieurs manières d'organiser les systèmes, chaque organisation mettant en lumière des types d'interaction différents [Seck and Honig, 2012].

Mais la notion de hiérarchie ne se limite pas à la manière dont nous modélisons le système et sa structure interne. La notion d'organisation apparait à chaque étape de l'activité de M&S (comment les informations constitutives du modèle formel sont organisées, comment le programme résultant est organisé). L'identification des différentes étapes de l'activité de M&S est elle-même issue d'un processus d'organisation. Il faut donc garder à l'esprit que la capacité à organiser de manière hiérarchique est fondamentale dans l'étude des systèmes complexes. Elle doit donc se retrouver au niveau de la démarche de modélisation adoptée et au niveau des outils utilisés.

3.1.4 Complexité et courants de pensées en science

Dans [Ramat, 2006] est effectué un parallèle entre plusieurs courants de pensées (le globalisme, le réductionnisme et le holisme) et la vision que nous avons actuellement de la complexité, des systèmes et de la démarche de M&S. Ce parallèle intéressant permet de prendre du recul sur la démarche de modélisation et sur la notion de complexité. Ces quelques courants de pensées permettent de prendre de la hauteur sur notre manière d'appréhender, de concevoir ou d'étudier un système, il nous semble donc important de les présenter, même brièvement :

— Le **globalisme** voit le système comme un tout. Ce courant se focalise sur la fonction du système (son rôle, son influence sur l'environnement) sans se soucier d'identifier sa structure interne. Le but est d'étudier le système dans sa globalité en se basant sur sa relation à son environnement.

— Le **réductionnisme** voit le système comme un ensemble d'éléments. "*Toute réalité se réduit en fin de compte à des constituants élémentaires*", la compréhension du système passe alors par l'étude de sa structure interne.
— le **holisme** voit le système comme un Tout qui dépasse la somme de ces éléments constitutifs. "*le Tout est plus que la somme des parties*", ce courant de pensées est lié à la notion d'émergence et a particulièrement influencé le domaine des systèmes multi-agents.

Nous pouvons reconnaître dans ces courants des idées présentées précédemment. Par exemple, les visions des réductionnistes et des holistes sont évidemment fondamentales lorsque nous définissons un système complexe comme un ensemble d'éléments en interaction.

D'autre part, l'idée globaliste d'un système plongé dans un environnement se retrouve comme l'un des 4 préceptes du nouveau discours de la méthode [Le Moigne, 2006]. Ces 4 préceptes permettent de guider la démarche scientifique de modélisation. Il est important de les énoncer car ils mettent en perspective des notions essentielles du domaine de la M&S. Nous présentons ces 4 préceptes ci-dessous en les mettant en perspective par rapport aux notions introduites au chapitre 2 :

— le **précepte de pertinence** : "*Convenir que tout objet que nous considérerons se définit par rapport aux intentions implicites ou explicites du modélisateur. Ne jamais s'interdire de mettre en doute cette définition si, nos intentions se modifiant, la perception que nous avions de cet objet se modifie*". Dans l'activité de M&S définie au chapitre 2, ce précepte nous rappelle que le modèle est développé dans un but précis (l'intention du modélisateur) qui conditionne ce qui est pris ou non en compte dans le système étudié. Cela peut alors être rapproché des notions de cadre expérimental puis de validité du modèle.
— le **précepte du globalisme** : "*Considérer toujours l'objet à connaître par notre intelligence comme une partie immergée et active au sein d'un plus grand tout. Le percevoir d'abord globalement, dans sa relation fonctionnelle avec son environnement sans se soucier outre mesure d'établir une image fidèle de sa structure interne, dont l'existence et l'unicité ne seront jamais tenues pour acquises*". Ce précepte souligne l'importance de l'environnement dans lequel le système est plongé, ce qui implique de nouveau l'importance de déterminer le cadre expérimental. Cela mène aussi à la notion de "système ouvert" et à l'idée que le système ne peut être défini sans son environnement [Morin, 2015].
— **le précepte téléologique** : "*Interpréter l'objet non pas en lui-même, mais par son comportement, sans chercher à expliquer a priori ce comportement par quelque loi impliquée dans une éventuelle structure. Comprendre en revanche ce comportement et les ressources qu'il mobilise par rapport aux projets que, librement, le modélisateur attribue à l'objet. Tenir l'identification de ces hypothétiques projets pour un acte rationnel de l'intelligence et convenir que leur démonstration sera bien rarement possible*". Dit autrement, il faut commencer par observer objectivement l'objet, ses comportements et sa relation à l'environnement sans faire d'hypothèses ni sur sa structure interne, ni sur ses finalités. C'est seulement dans un second temps que des hypothèses et des interprétations peuvent être effectuées pour expliquer les comportements observés.
— **le précepte d'agrégativité** : "*Convenir que toute représentation est partisane, non pas par oubli du modélisateur, mais délibérément. Chercher en conséquence quelques recettes susceptibles de guider la sélection d'agrégats tenus pour pertinents et exclure l'illusoire objectivité d'un recensement exhaustif des éléments à considérer*". Ce précepte incite à la modestie du modélisateur qui ne doit pas chercher à tout expliquer du système qu'il étudie. Dit autrement, il faut accepter qu'un modèle ne peut pas être une représentation exhaustive du système étudié, c'est une représentation partielle choisie par le modélisateur et dépendante de la question posée.

3.1.5 Bilan

Nous avons détaillé dans cette section la notion de complexité en M&S, notamment en prenant l'exemple des systèmes cyber-physiques, et les besoins que cela engendre sur l'activité de M&S : la nécessité d'associer différents points de vue pouvant provenir d'experts de domaines variés, et la nécessité d'organiser les éléments entre eux selon une ou plusieurs hiérarchies pour pouvoir appréhender le système dans sa globalité.

Dans l'approche par modélisation et simulation présentée au chapitre 2, le système est modélisé selon un angle de vue unique et mono-domaine. C'était alors aux experts domaines de réaliser les différentes étapes de l'activité de M&S, en utilisant ou développant des outils formels et logiciels dédiés.

Dans le cas des systèmes complexes, il faut maintenant permettre aux différents experts de travailler ensemble dans une démarche commune. Il faut donc trouver un moyen de **réconcilier les différents points de vue au sein d'une démarche rigoureuse de modélisation et simulation** [Camus, 2015]. Il faut concevoir de nouvelles approches et développer les outils pour supporter leur mise en place.

3.2 Multi-modéliser et co-simuler

Cette section va nous permettre d'introduire la multi-modélisation et la co-simulation, deux approches actuellement en expansion pour la modélisation et simulation de systèmes complexes.

3.2.1 Multi-modélisation et modélisation multi-formalisme

La multi-modélisation est une approche prometteuse pour modéliser des systèmes complexes. Comme nous l'avons vu précédemment, face à la complexité il est nécessaire d'organiser, de hiérarchiser, de réduire le problème en sous parties plus faciles à résoudre.

L'activité de modélisation est guidée par une interrogation qui permet de définir un système cible à modéliser (voir chapitre 2). Si le système cible est complexe, cela revient à considérer que l'interrogation initiale est complexe et qu'il est nécessaire de la décomposer. Nous obtenons alors plusieurs questions, chacune d'elles traitant un aspect particulier du système cible global et conduisant à la construction d'un modèle particulier. Les modèles répondent alors chacun à une partie du problème, et mis ensemble, ils permettent d'étudier le système complexe dans sa globalité. C'est le principe de la multi-modélisation.

Cela signifie que lorsque nous voulons répondre à plusieurs questions au sujet d'un système cible, i.e. l'étudier sous plusieurs aspects, plutôt que de créer un seul modèle pour répondre à ces questions, l'idée de la multi-modélisation est de créer plusieurs modèles, chacun adapté à la représentation d'un aspect spécifique du système. Le système complet est alors représenté par un multi-modèle (Définition 12) qui répond aux différentes questions posées [Fishwick and Zeigler, 1992].

Définition 12 (Multi-modèle)**.** 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]

Cette définition regroupe en réalité une multitude de cas selon le nombre de sous-modèles, s'ils sont simulés séquentiellement ou en même temps... Une classification est présentée dans [Yilmaz and Ören, 2004] (voir Figure 3.1).

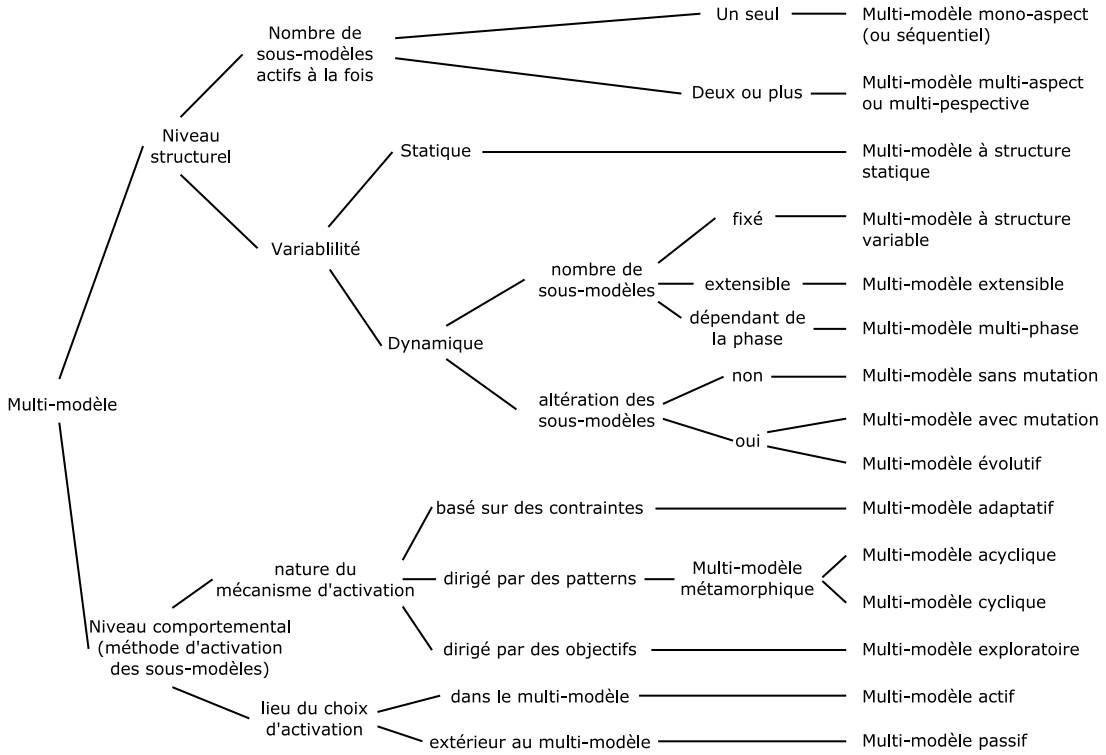


FIGURE 3.1 – Classification des multi-modèles. Traduit de [Yilmaz and Ören, 2004]

Les systèmes complexes sont composés de multiples entités que nous modélisons séparément, et l'interaction de leurs modèles au sein du multi-modèle doit permettre de reproduire le comportement global du système complexe. Il est entendu que chaque entité doit être modélisée en utilisant le formalisme le plus adéquat [Vangheluwe et al., 2002], leur intégration en un multi-modèle nécessite donc de gérer le couplage entre les formalismes. De ce fait, la multi-modélisation est généralement associée à une modélisation multi-formalisme.

3.2.2 Co-simulation

Dans un multi-modèle, chaque modèle est défini dans le formalisme le plus adapté, ce qui signifie que les dynamiques de simulation des modèles peuvent être différentes, i.e. il faut utiliser des simulateurs différents pour dérouler leurs comportements. Nous parlons alors de co-simulation (Définition 13) d'un multi-modèle lorsque chaque modèle est simulé avec son propre simulateur et que les simulateurs interagissent en cours de simulation. La co-simulation consiste à assurer la synchronisation et l'échange de données entre les simulateurs pendant leur simulation.

Définition 13 (Co-simulation)**.** La co-simulation consiste à permettre une simulation globale d'un système couplé (i.e. composé de plusieurs sous-systèmes) via le couplage de plusieurs simulateurs [Gomes et al., 2018].

D'un point de vue logiciel, la co-simulation permet notamment de réutiliser des modèles issus de différents logiciels de M&S et de les faire interagir. Pour la modélisation d'un système complexe composé d'entité de différents domaines comme les systèmes cyber-physiques, l'intérêt est de permettre à chaque expert de modéliser les entités correspondant à son domaine en utilisant l'outil le plus adapté pour cela.

3.2.3 Bilan

Une approche de multi-modélisation et co-simulation permet à la fois de combiner plusieurs perspectives d'experts différents, et de les laisser utiliser leurs outils dédiés. C'est donc une approche de choix pour la M&S de systèmes complexes. Cependant, l'utilisation d'une approche de multi-modélisation et de co-simulation soulève des challenges au niveau de l'*interopérabilité* des modèles.

D'un point de vue pratique, la construction d'un multi-modèle nécessite de pouvoir assembler des modèles de manière modulaire, i.e. en permettant l'ajout et la suppression d'un modèle composant ou son échange avec un autre. Cet assemblage modulaire fait appel à la notion de composition de modèles.

Ces notions d'*interopérabilité* entre modèles et de composition de modèles sont présentées dans la section suivante.

3.3 *Interopérabilité* et composition de modèles

La multi-modélisation et la co-simulation impliquent que les modèles et les simulateurs puissent échanger de l'information entre eux. Cette capacité à échanger de l'information fait référence à la notion d'*interopérabilité* dont la définition usuelle est donnée ci-dessous (Définition 14).

Définition 14 (Interopérabilité)**.** L'*interopérabilité* est la capacité permettant à deux systèmes ou plus d'échanger de l'information et d'utiliser l'information échangée. [Diallo et al., 2011] tiré de [Society, 1990]

Dans le cadre de la théorie de la M&S, rendre deux (ou plus) modèles interopérables au sein d'un multi-modèle et d'une co-simulation, nécessite de se poser plusieurs niveaux de questions[Tolk et al., 2007]. La définition usuelle ne rend pas compte de tous ces niveaux ce qui nous amène à adopter une autre définition (Définition 15).

Définition 15 (Interopérabilité)**.** L'*interopérabilité* est la capacité pour différentes simulations, connectées en un système de simulations distribuées, de collaborer de manière **pertinente** pour simuler un scénario commun ou un monde virtuel. [Petty and Weisel, 2003a]

Le point important de cette définition est la notion de pertinence de la collaboration entre plusieurs modèles et simulations. La construction des modèles est basée sur des hypothèses. Pour connecter deux modèles de manière pertinente, il faut s'assurer que leurs hypothèses de constructions soient compatibles. Le problème n'est plus seulement la capacité à échanger et utiliser de l'information, il faut aussi que cet échange ait du sens.

Cet aspect est approfondi dans la suite où nous détaillons les différents niveaux d'*interopérabilité* et la notion de composition.

3.3.1 Les niveaux d'*interopérabilité*

L'échelle LCIM (*Level of Conceptual Interoperability Model*), introduites dans [Tolk and Muguira, 2003] et améliorée depuis, présente les différents niveaux d'*interopérabilité* atteignables entre deux systèmes (voir Figure 3.2).

Cette échelle présente 6 niveaux :

0. **Pas d'*interopérabilité*** : Cela correspond à un système isolé ne possédant aucune capacité d'échange d'information.

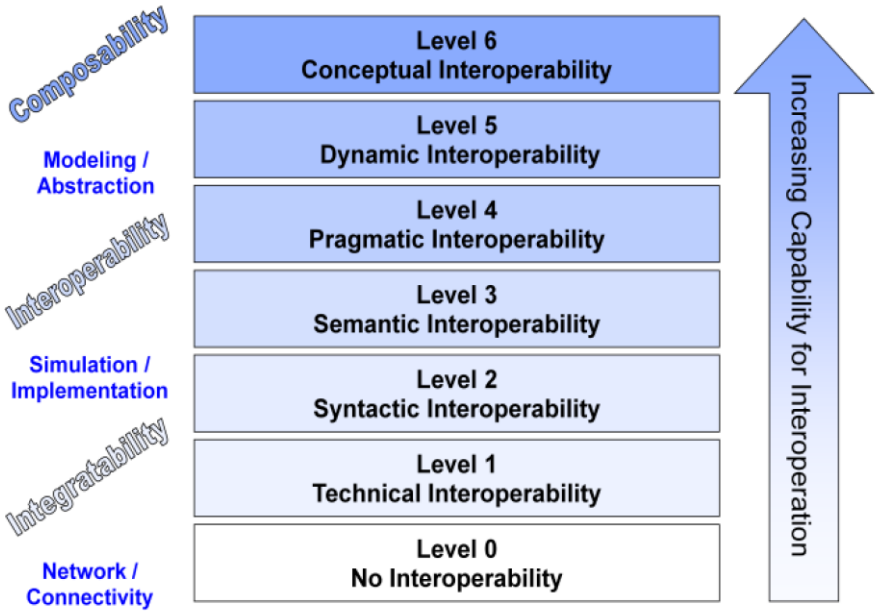


FIGURE 3.2 – Level of Conceptual Interoperability Model. [Tolk et al., 2007]

1. ***Interopérabilité* technique** : Elle correspond à l'établissement d'un protocole de communication sur une solution technique déterminée pour échanger des données.
2. ***Interopérabilité* syntaxique** : À partir de ce niveau, les données sont structurées permettant à chaque système participant à l'échange de les traiter. Cela revient par exemple à déterminer le type des données échangées (réel, entiers...), la représentation de ces types est non ambigüe.
3. ***Interopérabilité* sémantique** : En plus d'un format commun, les données échangées sont associées à un sens, une signification commune à chaque participant de l'échange. La signification du contenu des messages échangés est partagée.
4. ***Interopérabilité* pragmatique** : Ce niveau ajoute la connaissance du contexte d'échange de la donnée et de l'utilisation qui en sera faite (quelle méthode sera utilisée, sans connaissance de l'effet que cela produira). Chaque participant a la connaissance de l'interface (méthodes et procédures) du système avec lequel il interagit.
5. ***Interopérabilité* dynamique** : Ajoute la connaissance de l'effet de l'échange. Les systèmes voient leur état interne évoluer au fur et à mesure des échanges successifs. L'utilisation du message et l'effet qu'il produira sur la structure interne du système recevant la donnée sont définis.
6. ***Interopérabilité* conceptuelle** : Dernier niveau d'*interopérabilité* ajoutant la spécification des hypothèses et des contraintes liées à l'échange. En M&S, cela implique une parfaite documentation du modèle conceptuel.

Nous pouvons faire un parallèle entre ces différents niveaux d'*interopérabilité* et le schéma Figure 2.2 qui présente les étapes de la M&S. Lors du développement de plusieurs modèles devant faire partie d'un multi-modèle, la mise en place de l'*interopérabilité* technique et syntaxique doit se faire au niveau de la spécification opérationnelle et de l'implémentation du modèle. Les niveaux d'*interopérabilité* sémantique et pragmatique peuvent être atteints au niveau de la formalisation

du modèle, i.e. au moment où sont définies les méthodes liées à la dynamique du modèle. Enfin, pour mettre en place une *interopérabilité* dynamique et conceptuelle entre deux modèles, il faut s'intéresser à la première étape de l'activité de M&S, l'étape de conceptualisation où sont définis les évènements pouvant influencer les modèles ainsi que les hypothèses et simplifications effectuées.

L'intérêt de cette échelle est qu'elle présente une hiérarchisation générique des différents problèmes à résoudre pour rendre deux systèmes interopérables. Cette hiérarchie peut se résumer en 3 grandes étapes : l'intégrabilité (niveaux 1 et 2), l'*interopérabilité* (niveaux 3 et 4) et la composabilité (niveaux 5 et 6). La notion de composabilité est vue ici comme le plus haut niveau d'*interopérabilité*, i.e. deux modèles interopérables au plus haut niveau sont composables ce qui nous conduit à la notion de composition.

3.3.2 La composition de modèles

La composabilité (Définition 16), correspond à la faculté à assembler des modèles pour former des multi-modèles valides.

Définition 16 (Composabilité)**.** La composabilité est la capacité à sélectionner et assembler des composants de simulation selon différentes combinaisons pour former un système de simulation valide qui satisfait les besoins spécifiques de l'utilisateur. [Petty and Weisel, 2003a]

Cette vision de la composabilité implique qu'un mécanisme permette au modélisateur de coupler les modèles avant le lancement de la simulation. Elle implique aussi que les modèles à composer soient effectivement interopérables au plus haut niveau, ce qui fait le lien avec la composabilité vue Figure 3.2.

Le point difficile ajouté par cette définition est la notion de validité du multi-modèle résultant. Cette notion de validité ne dépend pas d'un seul modèle, mais d'un ensemble de modèles en interaction, chaque modèle représentant une partie, un aspect du système complexe cible. Intuitivement, cela impose que chaque modèle soit valide vis-à-vis de la partie qu'il représente. Mais il faut aussi étudier la validité des interactions entre modèles, leur cohérence par rapport au système réel.

Cela amène à l'identification de deux types de composition, la **composition syntaxique** qui correspond à la capacité à coupler des modèles, et la **composition sémantique** qui consiste à savoir si le multi-modèle résultant a du sens, i.e. s'il est valide.

Rappelons que la validation (Définition 10) est une propriété mettant en relation le système cible, le modèle et le cadre expérimental. Dans un multi-modèle, les cadres expérimentaux qui ont permis la construction de chaque modèle composant doivent aussi être cohérents entre eux. Il faut noter que même si nous considérons les modèles composants comme valides dans un cadre expérimental donné, il est très difficile d'affirmer qu'un multi-modèle sera valide à priori, i.e. avant sa simulation [Petty and Weisel, 2003b].

Le problème de composabilité des modèles est un challenge en M&S [Taylor et al., 2013]. Il nécessite de prendre en compte chaque niveau de développement des modèles (conceptuel, formel et technique), ce qui le différencie du problème de composition de logiciels [Davis and Anderson, 2004]. De plus, aux dimensions syntaxique et sémantique de la composition peut être ajouté l'aspect pratique. Ainsi, dans [Morse, 2004], plusieurs conseils sont donnés sur les fonctionnalités à associer à la composition de modèles. Ces fonctionnalités sont liées à la possibilité de modéliser les systèmes de manière hiérarchique grâce à la réutilisation de multi-modèles, au support des processus de vérification et validation grâce à l'automatisation de tâches, au passage à l'échelle...

La réutilisation des modèles couplés (modèles composés de sous-modèles en interaction) sous forme de hiérarchie est liée à la propriété de fermeture par couplage (Définition 17), appelée aussi fermeture par composition [Zeigler et al., 2000].

Définition 17 (Fermeture par couplage)**.** Un formalisme F est clos par couplage si un modèle M, résultat d'un couplage de modèles issus de F, est bien défini et appartient lui-même à F. [Zeigler et al., 2000]

Cela a plusieurs significations :
- Du point de vue de la spécification : la spécification de la structure d'un modèle couplé est équivalente à celle d'un modèle atomique.
- Du point de vue de l'utilisation : il est possible d'utiliser un modèle couplé comme si c'était un modèle atomique, cela autorise la construction hiérarchique.
- Du point de vue du comportement : nous pouvons écrire un modèle atomique dont le comportement dynamique est identique à celui du modèle couplé.

Autrement dit, la propriété de fermeture par couplage signifie qu'un modèle couplé est strictement équivalent à un modèle atomique.

Il y a une distinction entre la notion de composition au niveau des formalismes, où le modèle composé doit être bien défini et appartenir au formalisme considéré, et la notion de composition de modèles au sens de la composabilité définie en 16, qui ajoute en plus la notion de validité du modèle couplé résultant.

Dans le cadre de la co-simulation, deux modèles sont composables lorsqu'il est possible de les coupler et de former un multi-modèle valide vis à vis d'un système cible, sans supposer que ce nouveau système sera lui-même réutilisable pour modéliser hiérarchiquement un système d'ordre supérieur. Lorsque nous voudrons exprimer cette capacité à réutiliser un multi-modèle comme si c'était un modèle atomique, nous utiliserons plutôt le terme de "fermeture par couplage". La modélisation par composition implique la capacité à composer les modèles au sens de l'*interopérabilité* mais sous-entend généralement aussi la capacité à construire hiérarchiquement les multi-modèles.

3.4 Les challenges

Cette section va nous permettre d'insister sur les challenges liés à la multi-modélisation et co-simulation de systèmes complexes.

3.4.1 L'*interopérabilité*

Nous avons vu que la modélisation d'un système complexe nécessite l'agrégation de plusieurs points de vue, domaines, expertises et la collaboration d'experts aux outils conceptuels et logiciels différents.

La multi-modélisation et la co-simulation sont des approches prometteuses. Elles permettent à chaque expert de se focaliser sur une sous-partie du système cible appartenant à son domaine d'expertise, ceci pour développer un *modèle composant* qui sera intégré au multi-modèle global. La co-simulation permet à chaque *modèle composant* d'être simulé séparément, autorisant les experts à utiliser leurs propres outils logiciels. L'utilisation de logiciel de M&S existants permet de capitaliser leur savoir faire mais soulève la question de leur intégration dans la démarche. **Le problème est alors moins la création des modèles que le développement des moyens pour les faire interagir de manière systématique et cohérente**.

C'est donc en premier lieu un challenge d'*interopérabilité* à tous les niveaux :

— au niveau conceptuel, il faut pouvoir assurer la cohérence des interactions
— au niveau formel, il faut pouvoir intégrer les différents formalismes et coupler leur différentes dynamiques
— au niveau opérationnel, il faut que les spécifications des logiciels permettent leurs interactions
— au niveau technique, il faut que les différents programmes puissent échanger de l'information

La difficulté est alors de fournir un ensemble de solutions génériques à chacun de ces niveaux pour aboutir à une approche rigoureuse de multi-modélisation et co-simulation.

3.4.2 Propriétés attendues

Le deuxième niveau de challenge est lié aux propriétés que nous pouvons attendre d'une approche par multi-modélisation et co-simulation. Pour mettre en valeur ces propriétés, nous commencerons par nous intéresser à l'évaluation des performances d'un seul modèle.

Attention, le terme "performance" est utilisé ici au sens de [Brooks and Tobias, 1996] et regroupe des caractéristiques liées au développement et à la simulation d'un modèle. À ne pas confondre avec les performances d'un programme en terme de temps de calcul.

Les performances d'un modèle

L'activité de modélisation et simulation consiste en la création d'un modèle pour répondre à une question particulière sur un système cible. Le modèle sert alors de substitut au système cible pour la réalisation d'expériences qui permettront de répondre à l'interrogation du modélisateur. L'évaluation d'un modèle peut donc porter sur deux aspects : son développement et son utilisation.

Dans [Brooks and Tobias, 1996], les performances d'un modèle sont classées en 4 catégories :

— Résultats
  1. l'adéquation des résultats par rapport à la question posée,
  2. la précision des résultats,
  3. la facilité de compréhension du modèle et de ses résultats,
— Utilisation future
  4. la facilité avec laquelle le modèle peut être réutilisé et couplé à d'autres modèles,
— Vérification et Validation
  5. la vérification de la cohérence entre le modèle numérique et le modèle conceptuel,
  6. la validation des résultats produits,
  7. la crédibilité, i.e. la force des bases théoriques utilisées et la qualité des données d'entrée,
— Ressources requises
  8. temps et coût de développement (dont récolte de données, vérification et validation),
  9. temps et coût de simulation,
  10. temps et coût de l'analyse des résultats,
  11. coût du matériel requis pour la simulation.

Les quatre premiers items peuvent être rapprochés de l'utilisation du modèle tandis que les suivants peuvent être associés à son développement. L'intérêt de cette classification est qu'elle met en lumière des aspects souvent oubliés lors de la présentation des performances d'un modèle, comme la facilité avec laquelle il peut être réutilisé et ses coûts de développement.

Les performances d'une approche

Notons que dans cette partie, pour évaluer une approche de multi-modélisation et co-simulation nous supposons que celle-ci n'est pas seulement théorique mais que sa mise en œuvre est supportée par des outils logiciels.

Bien que les critères de performance précédents soient pensés pour évaluer un modèle indépendant, ils peuvent servir de base pour l'évaluation d'un multi-modèle. Il est alors pertinent d'évaluer une approche de multi-modélisation et co-simulation par rapport à sa capacité à maximiser ces critères pour les multi-modèles qui en sont issus.

En reprenant les critères précédents, voici les éléments qu'une approche de multi-modélisation et co-simulation devrait proposer :

- Faciliter la compréhension des multi-modèles et des interactions qu'ils contiennent.
- Rendre possible la réutilisation des modèles et des multi-modèles.
- Faciliter la vérification des modèles et des multi-modèles.
- Faciliter la validation des modèles et des multi-modèles.
- Augmenter la crédibilité du multi-modèle en reposant sur des bases théoriques solides.
- Réduire les temps et coûts de développement, notamment proposer des solutions pour l'intégration des différents outils.
- Réduire les temps de co-simulation, notamment réduire l'impact des communications entre les modèles.

Nous n'avons repris ci-dessus que les critères que nous considérons impactés par le passage d'un modèle simple à un multi-modèle. Nous pouvons ajouter à ces éléments le fait de faciliter la répétabilité et la reproductibilité des expériences, deux aspects essentiels en science expérimentale (voir 2). Il ne faut pas oublier non plus que la multi-modélisation doit permettre de faire collaborer des experts de différents domaines, l'approche proposée doit donc aussi être suffisamment générique.

Au cours de ce manuscrit, nous nous focaliserons sur trois propriétés qui impactent fortement les performances d'une approche de multi-modélisation et co-simulation :

- La **modularité** : La modularité est la capacité à remplacer facilement un composant par un autre, ou en ajouter/supprimer. Dans nos multi-modèles, avoir des *modèles composants* modulaires permet de tester facilement différentes alternatives. Cette propriété augmente la réutilisabilité des modèles qui pourront être intégrés facilement dans différents multi-modèles. Elle diminue ainsi les coûts de développement. Pour être effective, la modularité doit être pensée sur chacune des étapes de l'activité de M&S, i.e. elle doit être présente du niveau conceptuel (modèles conceptuels modulaires) jusqu'au niveau technique (modèles numériques modulaires).
- La **capacité de hiérarchisation** : Elle correspond à la faculté d'organiser les modèles au sein du multi-modèle. Si la taille du multi-modèle est importante et que celui-ci met en jeu beaucoup de domaines et d'interactions, il faut permettre au modélisateur d'organiser et de hiérarchiser pour avoir une vision claire de son travail. Si nous prenons l'exemple d'une hiérarchie d'inclusion, un multi-modèle peut être composé de plusieurs autres multi-modèles eux-même composés de multi-modèles etc (c'est le type de hiérar-

chie permis dans un formalisme fermé par couplage, voir exemple Figure 3.3). Cela permet une multi-modélisation hiérarchique en regroupant les composants par domaine ou selon d'autres critères. Comme nous l'avons vu, la capacité à hiérarchiser est essentielle face à la complexité. Cette propriété facilite la compréhension du multi-modèle, et permet aussi la diminution du temps de développement (d'autant plus si les multi-modèles sont eux-mêmes des composants modulaires).

— La **facilité d'utilisation** : Pour être intéressante, une approche de multi-modélisation et co-simulation doit pouvoir être utilisée par différents experts collaborant sur un multi-modèle. Cette propriété va être importante au niveau de l'outil logiciel qui sera proposé pour construire les multi-modèles et les co-simuler. Un environnement logiciel adapté permet de diminuer le temps de développement (notamment s'il permet d'automatiser les processus de vérification).

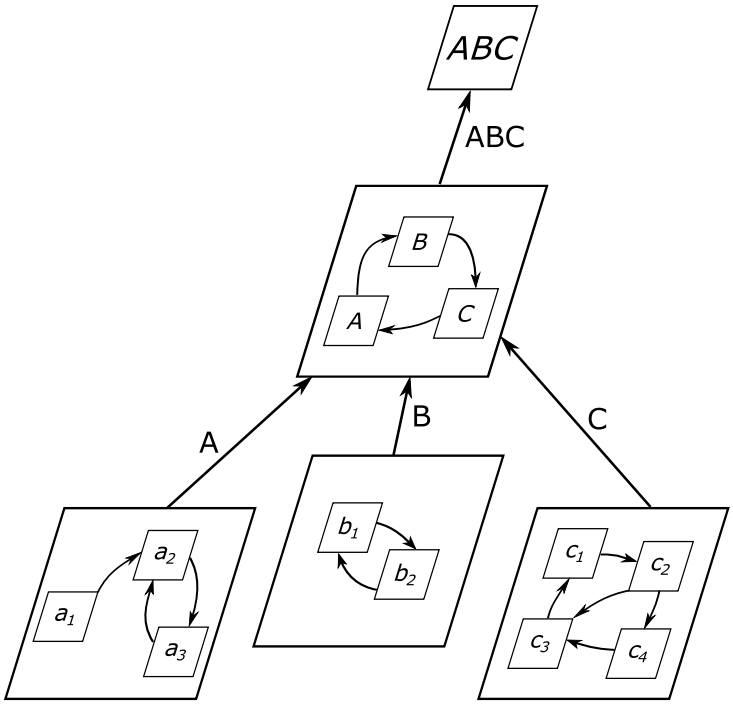


FIGURE 3.3 – Exemple de hiérarchie d'inclusion : construction d'un multi-modèle ABC à partir des multi-modèles A, B, C eux-mêmes composés de sous-modèles.

Le développement de la multi-modélisation et co-simulation dépasse le cadre de la gestion des hétérogénéités à tous les niveaux (challenge d'*interopérabilité*). **Le challenge est aussi de fournir une approche générique rigoureuse dotée de fonctionnalités de modélisation avancées comme la modularité, la capacité à hiérarchiser et la facilité d'utilisation**.

3.5 Conclusion

L'objectif de cette partie a été de présenter les challenges liés à la modélisation et simulation de systèmes complexes. Face à la complexité, la première étape est de décomposer le problème en sous-problèmes plus faciles à résoudre, puis de mettre en relation les différentes solutions pour aboutir à une résolution globale. C'est le principe de "diviser pour régner".

Dans le cadre de la modélisation d'un système complexe, l'approche par multi-modélisation consiste à identifier différents sous-systèmes, de leur créer des modèles, puis de faire interagir ces modèles au sein d'un multi-modèle représentant le système complexe. La co-simulation consiste à autoriser la simulation autonome de chaque modèle ce qui rend possible la réutilisation de ceux déjà implémentés dans différents outils de M&S. Cela permet de capitaliser les ressources logicielles existantes, mais implique aussi de gérer l'hétérogénéité logicielle.

La multi-modélisation et la co-simulation soulèvent de nombreuses questions sur l'interopérabilité des modèles. En effet, nous avons vu en prenant l'exemple des CPS que chaque entité peut être modélisée par des experts de différents domaines utilisant des outils conceptuels, formels et logiciels différents. Il faut donc s'assurer de la possibilité de les utiliser en collaboration. Nous faisons face alors à un challenge d'interopérabilité lié à l'intégration de ces outils au sein d'une même démarche de multi-modélisation et co-simulation. C'est un challenge sur trois niveaux :

— le niveau conceptuel que nous associons à la composabilité,
— le niveau formel associé à l'interopérabilité,
— le niveau technique associé à l'intégrabilité.

Mais l'intégration rigoureuse de différents outils n'est pas le seul challenge, il faut aussi associer à cette approche des fonctionnalités avancées de modélisation qui permettent de faciliter le développement et d'augmenter la valeur des multi-modèles. Nous nous focaliserons dans ce manuscrit sur les trois propriétés décrites précédemment (la **modularité**, la **capacité à hiérarchiser** et la **facilité d'utilisation**). Nous insistons sur le fait que la capacité à hiérarchiser et à organiser les différents éléments de résolution est fondamentale dans l'étude des systèmes complexes. Elle doit donc être présente dans la démarche de multi-modélisation et co-simulation comme dans les outils logiciels permettant sa mise en œuvre.

Le prochain chapitre sera dédié à la présentation de différentes approches de multi-modélisation et co-simulation, à la comparaison des solutions mises en place pour répondre aux challenges posés, et au positionnement de notre travail.

4

Approches existantes et positionnement

Le développement de la pensée complexe et les nouveaux besoins liés à la M&S de systèmes cyber-physiques nécessitent de nouvelles méthodes, comme la multi-modélisation et la co-simulation, qui dépassent le cadre d'un domaine spécifique.

Nous présenterons dans ce chapitre quelques approches de modélisation multi-formalismes et de co-simulation. Elles ont chacune des conceptions différentes sur la manière de construire un multi-modèle et de le simuler mais proposent toutes des solutions pour coupler des modèles hétérogènes et bénéficier des propriétés qui nous intéressent, i.e. la modularité, la capacité à hiérarchiser et la facilité d'utilisation. Nous introduisons notamment MECSYCO et rappelons sa place dans la littérature afin de positionner notre contribution.

4.1 Introduction

Ce chapitre positionne notre contribution en présentant différents travaux de la littérature s'attaquant aux problèmes de la multi-modélisation et co-simulation.

Nous n'insistons pas sur les approches ad-hocs, i.e. les approches dédiées à un problème particulier et qui ne proposent pas de solutions génériques d'interopérabilité. Elles sont généralement limitées à 2 ou 3 simulateurs [Gomes et al., 2017] sans possibilité d'en ajouter de nouveaux (ou au prix d'un effort important et d'une refonte du code). C'est la cas par exemple dans [Bollinger et al., 2016], où une co-simulation est effectuée en utilisant NetLogo [Wilensky, 1999] et Matpower, une bibliothèque MatLab. Le but était d'avoir dans un même multi-modèle des perspectives sociales et techniques sur les réseaux de transports d'électricité pour étudier les liens entre le marché de l'énergie et les infrastructures techniques. Ne pouvant pas être exhaustif, plusieurs solutions présentes dans la littérature ne sont pas non plus traitées ici. Nous pouvons juste évoquer Anylogic [Borshchev and Filippov, 2004] qui propose d'associer trois perspectives de modélisation différentes (système dynamique, évènement discret et multi-agent) au sein d'une même plateforme mais qui se limite à ces trois perspectives, et Mosaik [Schütte et al., 2011] qui propose d'intégrer des composants de simulation hétérogènes mais en se focalisant sur l'étude de réseaux électriques intelligents.

Nous présentons des travaux qui proposent des méthodes variées pour répondre au challenge d'*interopérabilité*, chacune de ces méthodes attaquant le problème selon des angles différents. Nous distinguons deux types d'approche : les approches d'interopérabilité logicielle, qui se basent sur des solutions techniques, et les approches multi-formalismes dont la réflexion se concentre sur l'intégration de formalismes hétérogènes.

4.2 Approches d'interopérabilité logicielle

Cette section est dédiée à des travaux qui s'appuient principalement sur des solutions techniques pour faire interagir différents logiciels de M&S.

4.2.1 *High Level Architecture*

Présentation générale

HLA (*High Level Architecture*) est une spécification d'architecture logicielle proposée par le DMSO (*Defense Modeling and Simulation Office*) du Département de la Défense américain [Dahmann et al., 1997][Dahmann, 1999]. Le but de cette spécification est de permettre la réutilisation et l'interopérabilité des composants de simulation développés au sein du DMSO, notamment pour réaliser des simulations distribuées (puis des co-simulations) mettant en relation des composants de simulation développés par différentes équipes.

Architecture

Dans HLA, les composants de la simulation distribuée sont appelés des **fédérés**. Ces fédérés peuvent être des simulations mais aussi des composants de visualisation ou des outils d'interaction avec un opérateur humain. Les fédérés interagissent au sein d'une **fédération**, une fédération désigne donc une simulation distribuée. HLA propose une approche de coordination centralisée autour de la **RTI** (*Run-Time Infrastructure*), c'est l'organe central de l'architecture, qui sert d'interface d'interaction entre chaque fédéré (Figure 4.1).

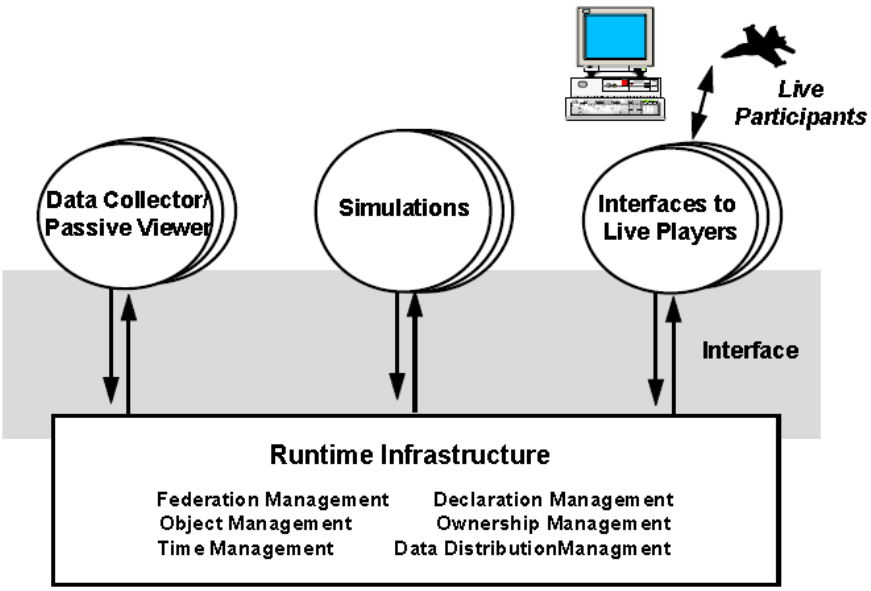


FIGURE 4.1 – Schéma d'une fédération HLA, tiré de [Dahmann et al., 1997]

Nous avons donc une fédération composée de fédérés en interaction grâce à un bus central appelé la RTI qui va gérer la co-simulation. Tous les fédérés peuvent être issus de logiciels, de langages de programmation et de plateformes d'exécution différents.

Pour assurer cette interopérabilité au sein d'une fédération, HLA spécifie plusieurs éléments :

— un standard commun de description (*Object Model Template)* utilisé pour :

  - documenter les fédérations (*Federation Object Model*),
  - documenter les fédérés (*Simulation Object Model*),
- les interfaces et leur utilisation :
  - interface des fédérés,
  - interface de la RTI,
  - protocole d'interaction avec les interfaces (gestion du temps, des données...),
- des règles, i.e. le principe de fonctionnement de chaque entité,
  - règles de fédérations,
  - règles de fédérés.

Propriétés et limites

Le standard HLA est une évolution par rapport à ces prédécesseurs (DIS [3] et ALSP [4]) qui a permis plus de flexibilité pour le développement de simulations distribuées [Tolk et al., 2012], au prix d'une complexité accrue. L'approche souffre cependant de plusieurs limites comme la spécification de la RTI, qui n'assure pas l'interopérabilité de deux implémentations différentes de cette dernière. La complexité de la norme fait que chaque implémentation est différente. Cela signifie qu'un modèle et un simulateur compatible avec une implémentation de la RTI ne sera pas compatible avec une autre implémentation. HLA n'est basé sur aucun formalisme et ne propose pas de structure globale réutilisable pour ses co-simulations, il n'est donc pas possible de réutiliser une fédération HLA au sein d'une autre fédération, ce qui empêche la réalisation de hiérarchies.

4.2.2 *Functional Mockup Interface*

Présentation générale

La *Functional Mockup Interface* (FMI) est une norme issue du monde industriel (initialement développée par Daimler AG) qui a pour but de faciliter l'échange de modèles de simulation [Blochwitz et al., 2014]. L'objectif de la norme est avant tout pratique : simplifier la collaboration entre partenaires industriels en leur donnant les moyens d'échanger leurs modèles.

L'échange de modèle du standard FMI repose sur des principes du génie logiciel, notamment l'utilisation de composants logiciels. Les modèles sont vus comme des boites noires appelées FMU (*Functionnal Mockup Unit*) contenant du code compilé (des librairies dynamiques). Ces FMU peuvent être données à un collaborateur qui pourra les exploiter au sein de co-simulations en employant l'interface d'interaction spécifiée par la norme. Le standard FMI définit comment exporter les modèles sous forme de composants logiciels, et quelles méthodes employer pour interagir avec ces composants pour effectuer des co-simulations. Il ne spécifie par contre pas l'algorithme de co-simulation (appelé aussi orchestrateur ou master de co-simulation) qui doit être utilisé. La Figure 4.2 présente les éléments d'une co-simulation basée sur FMI.

Deux stratégies d'échange

FMI a été pensé pour partager des modèles de systèmes continus ou hybrides qui sont représentés comme des systèmes d'équations différentielles ordinaires (ODE pour *Ordinary Differential Equation*) et/ou algébriques (DAE pour *Differential Algebraic Equation*). Dans ce formalisme :

- la structure des modèles est décrite par des variables d'état associées à des ODE et des DAE qui mettent en relations ces variables entre elles et avec le temps, et

3. *Distributed Interactive Simulation*
4. *Aggregate Level Simulation Protocol*

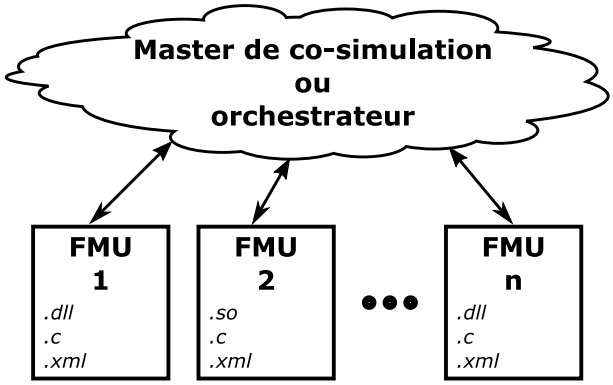


FIGURE 4.2 – Schéma d'une co-simulation basée sur FMI

— l'exécution des modèles est effectuée par des algorithmes d'approximation (ou solveurs) qui calculent pas à pas les différents états successifs du système.

Deux stratégies d'échange de modèle découlent de ce formalisme et sont proposées par FMI. Nous avons donc deux types de FMU :

— les FMU *Model-Exchange* (ME) contiennent la structure du modèle mais pas de solveur. Dans ce cas, celui qui reçoit la FMU doit implémenter lui-même un solveur qui doit faire évoluer toutes les variables d'état. La FMU se contentent d'évaluer les valeurs actuelles en fonction du système d'équation.
— les FMU *Co-simulation* (CS) contiennent la structure du modèle ainsi qu'un solveur pour le faire évoluer au cours du temps. C'est le solveur intégré qui fera évoluer les variables d'états du système en fonction des équations. Seules certaines variables identifiées comme étant des entrées du système peuvent être modifiées.

Architecture

Une FMU est un dossier zip contenant :

— Un fichier "modelDescription.xml", contenant une description des variables (paramètres, entrées, sorties) utilisées dans le modèle.
— Des fichiers C compilés (bibliothèques ".dll" ou ".so" suivant les systèmes) avec lesquels il est possible d'interagir via une interface standard.
— Les fichiers sources et les fichiers d'en-tête peuvent être présents.

Le point intéressant dans cette architecture n'est pas l'utilisation d'un fichier XML pour transmettre de l'information sur une bibliothèque logicielle (approche générique de génie logiciel), mais la définition des méthodes d'interaction avec le composant logiciel et la structuration du fichier XML (éléments spécifiques à l'échange de modèles de systèmes dynamiques et à la co-simulation).

Propriétés et limites

La norme FMI se concentre sur l'API logicielle et ne propose pas de solution d'intégration formelle. Aussi, la norme a été pensée pour les systèmes dynamiques écrits sous forme d'équations et n'est pas adaptée aux modèles évènementiels. Cette limite pour la gestion des évènements a déjà été soulevée et des solutions techniques d'amélioration du standard sont proposées [Tavella et al., 2016].

À l'heure de la rédaction de ce manuscrit, la norme FMI connait un fort engouement et plus de 80 outils se revendiquent compatibles (à l'export ou à l'import). Nous pouvons citer par exemple Daccosim [Galtier et al., 2015], Into-CPS [Larsen et al., 2016] ou MECSYCO [Camus et al., 2018]. Tous ces outils ne présentent pas les mêmes capacités en terme d'environnement de M&S. Parmi eux, nous choisissons de présenter ci-dessous l'outil Into-CPS pour les solutions qu'il propose concernant l'ensemble du cycle de M&S.

Exemple d'environnement compatible : Into-CPS

Into-CPS [Larsen et al., 2016][Lausdahl et al., 2017] est un consortium dont le but est de fournir une suite de logiciels pour la modélisation et simulation de systèmes cyber-physiques.

L'idée principale est de fournir un outil adapté pour chaque étape de l'activité de M&S, de la conception du multi-modèle, jusqu'à la co-simulation des différents composants :

- La partie conception est assurée via l'outil Modelio [5] qui utilise SysML (*System Modeling Language*) pour réaliser des descriptions de l'architecture des multi-modèles. Il faut noter que des tests peuvent être automatiquement générés à partir des modèles UML/SysML grâce à l'outil RT-Tester [6].
- La construction des modèles est séparée en plusieurs outils en fonction des spécialités, le point commun étant l'utilisation de la norme FMI pour les co-simulations :
  - Les systèmes physiques (partie continue) sont développés sur 20-sim [7], un environnement de M&S doté d'un langage similaire à Modelica. D'autres outils comme OpenModelica peuvent être utilisés du moment qu'ils peuvent exporter leurs modèles en FMU.
  - Les systèmes de contrôle commande (partie discrète) sont eux modélisés via Overture [8], un outil libre spécialisé dans l'analyse et la construction de systèmes informatiques à l'aide de la VDM-RT (*Vienna Development Method-Real Time*).
- Les co-simulations sont configurées et effectuées repectivement grâce aux outils Crescendo[Lausdahl et al., 2017] et TWT Engine [9].

L'un des intérêts de Into-CPS est l'identification des étapes successives permettant de modéliser et de co-simuler un CPS (conception du multi-modèle, implémentation des composants, co-simulation, test) et le fait qu'il propose des solutions adaptées à chacune d'elles.

La propriété de modularité est assurée par l'utilisation de FMI, il est très facile de remplacer une FMU par une autre du moment qu'elles possèdent les mêmes ports. Les multi-modèles étant construits sur Modelio, l'organisation des modèles et la hiérarchisation sont directement obtenues via SysML. Enfin, l'utilisation d'outil dédié à chaque étape du développement garantit la facilité d'utilisation.

Cependant, comme il est basé sur FMI, il hérite des mêmes limites concernant la gestion des évènements et n'a pas vocation à intégrer d'autres types de modèles.

4.2.3 Bilan

Au regard des deux solutions d'interopérabilité logicielle étudiées dans cette partie, nous pouvons identifier plusieurs grands principes.

5. https://www.modelio.org/
6. https://www.verified.de/products/rt-tester/
7. http://www.20sim.com/
8. http://overturetool.org/
9. https://www.twt-gmbh.de/en/produkte/co-simulation/cosimlab.html

Spécification des interfaces des composants de la co-simulation

Pour assurer l'interopérabilité entre les composants de la co-simulation, la première étape est de spécifier les interfaces d'interaction de ces composants. Cela s'observe dans HLA au travers de la spécification des fédérés, et dans le standard FMI avec la spécification de l'interface des FMU. La spécification des interfaces est dépendante de la dynamique que nous pourrons appliquer sur les modèles (i.e. leur simulateur). Bien que les approches présentées dans cette section ne se situent pas au niveau formel, il faut garder en tête que la spécification des interfaces est directement liée au formalisme dans lequel les modèles pourront être écrits. C'est aussi cela qui permet la création d'un algorithme de co-simulation (ou orchestrateur) qui mettra en relation les différents composants (peu importe le modèle du moment que son simulateur respecte l'interface standard).

La documentation des composants de la co-simulation

La spécification des interfaces des composants s'associe à une documentation. Cette documentation permet la réutilisation des éléments en rassemblant les informations qui les rendent spécifiques (i.e. noms de ports, paramètres, ...). Cette documentation suit une structure précise. Dans HLA, c'est le *Simulation Model Template* qui sert de base à la documentation des fédérés. Dans FMI, c'est le fichier "modelDescription.xml" dont la structure est prédéfinie qui permet de décrire les FMU. **C'est en associant la spécification des interfaces des simulateurs et leur documentation que nous obtenons des composants interopérables et réutilisables**.

La spécification de l'orchestrateur

Ce point est l'un des éléments divergent entre HLA et FMI. HLA a une vision centralisée et spécifie une structure centrale, la RTI, pour gérer la simulation distribuée des composants de simulation (dont l'interface est connue et qui fournissent une documentation pour être utilisés). À l'inverse, la norme FMI ne fournit pas de spécification du master de co-simulation, il doit seulement utiliser l'interface fournie par les FMU. La norme peut néanmoins se reposer sur les travaux existants concernant la résolution de systèmes équationnels. Plusieurs masters de co-simulation aux propriétés différentes sont proposés dans la littérature (par exemple dans [Bastian et al., 2011]).

Nous pouvons faire un parallèle entre la séparation modèle/simulateur préconisée dans la théorie de la modélisation et simulation et la séparation des spécifications des composants de simulations (fédérés, FMU, ...) de celles de l'orchestrateur (la RTI, le master de co-simulation, ...).

4.3 Approches multi-formalismes

Dans cette section nous nous intéressons aux approches qui se focalisent davantage sur les aspects formels de l'intégration de simulateurs hétérogènes et mettent les aspects techniques au second plan. Elles s'intéressent à comment rendre interopérables des simulateurs aux dynamiques différentes.

4.3.1 $ATOM^3$

Présentation

$ATOM^3$ (*A Tool for Multi-formalism, Meta-Modeling*) [De Lara and Vangheluwe, 2002] est un logiciel de M&S de systèmes complexes basé sur la transformation de formalisme. L'idée générale est que dans la modélisation d'un système complexe, chaque partie doit être modélisée avec le formalisme le plus adéquat. Pour faire fonctionner l'ensemble des modèles, ceux-ci doivent être réécrits, de préférence automatiquement, dans le formalisme commun le plus proche indiqué sur l'arbre de transformation des formalismes (voir Figure 4.3). Une fois ceux-ci réécrits dans un formalisme commun il suffit d'utiliser sa dynamique pour simuler l'ensemble des modèles.

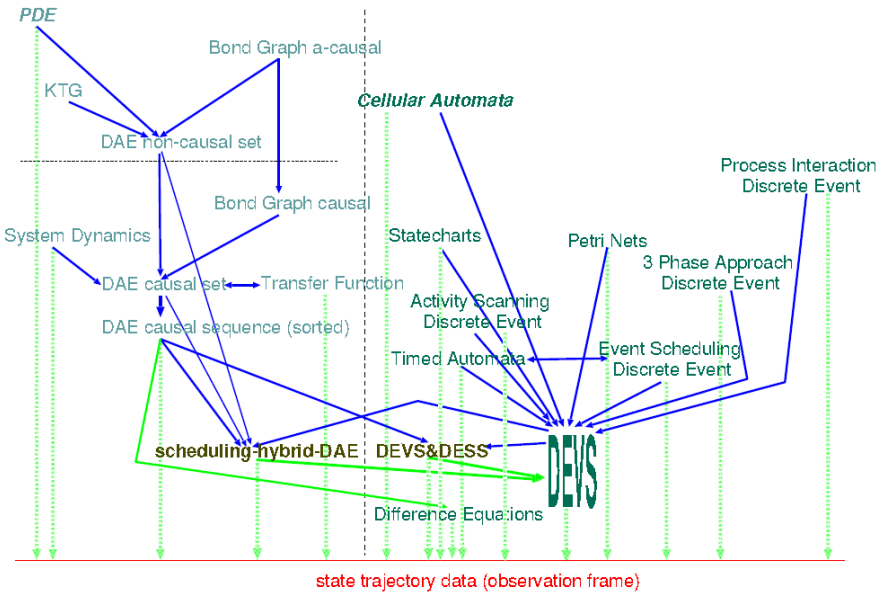


FIGURE 4.3 – Graphe de transformation des formalismes, tiré de [Vangheluwe et al., 2002]

Pour mettre en place cette stratégie, $ATOM^3$ repose sur la méta-modélisation, i.e. les formalismes et les opérations de transformation entre ceux-ci sont modélisés explicitement. La méta-modélisation consiste à modéliser les langages de modélisation eux-mêmes. Dans $ATOM^3$ cette étape est faite en UML associé à OCL (*Object Constraint Language*). Le méta-modèle permet ensuite de générer automatiquement un outil de M&S graphique pour le formalisme considéré. Les opérations de transformation de formalisme définies permettent (sans être limitées à cela) de passer d'un formalisme à l'autre.

Pour modéliser un système complexe dans $ATOM^3$ il faut :

— sélectionner l'ensemble des formalismes dont nous aurons besoin puis les modéliser s'ils ne sont pas encore disponibles,
— trouver le formalisme commun le plus proche et le modéliser s'il n'est pas disponible,
— trouver et modéliser s'il y a lieu les transformations nécessaires, puis
— modéliser les modèles voulus et les rassembler au sein du formalisme commun pour effectuer la simulation.

Propriétés et limites

$AToM^3$ est un outil de multi-modélisation dont l'approche est très pertinente car elle met au cœur de l'activité de multi-modélisation le choix et la spécification des formalismes à employer. De plus, le multi-modèle obtenu est homogène et peut être étudié avec des outils formels. En outre, il est possible de construire le multi-modèle hiérarchiquement. D'un point de vue pratique, l'outil fournit une interface graphique adaptée aux formalismes utilisés qui permet de passer automatiquement à la simulation. Le désavantage de l'approche est qu'elle ne permet pas la réutilisation des modèles existants dans d'autres logiciels, il est nécessaire de les réécrire dans $AToM^3$.

4.3.2 Ptolemy II

Présentation

Ptolemy II [Eker et al., 2003] est un logiciel de M&S de systèmes complexes basé sur le concept de hiérarchie hétérogène. Ptolemy II repose sur les notions d'acteurs et de Modèles de Calcul appelés MoC (*Model of Computation*). Les acteurs sont des processus concurrents dotés de ports d'entrée/sortie qui se synchronisent par échange de messages, ils représentent les modèles. Les MoC sont les règles de calcul qui déterminent comment sont exécutés les acteurs, i.e. les MoC déterminent les dynamiques de simulation des acteurs.

L'idée générale est que chaque niveau de la hiérarchie concerne un formalisme, un type de modèle particulier, i.e. chaque niveau de la hiérarchie est associé à un MoC.

La Figure 4.4 illustre les éléments d'une hiérarchie hétérogène dans Ptolemy II. Chaque niveau contient :

— des acteurs en interaction qui peuvent être atomiques ou composites,
— des récepteurs placés dans les ports d'entrée et qui gèrent les communications selon la sémantique du MoC, et
— un directeur qui définit le modèle de calcul à appliquer sur les acteurs de son niveau, il implémente un MoC.

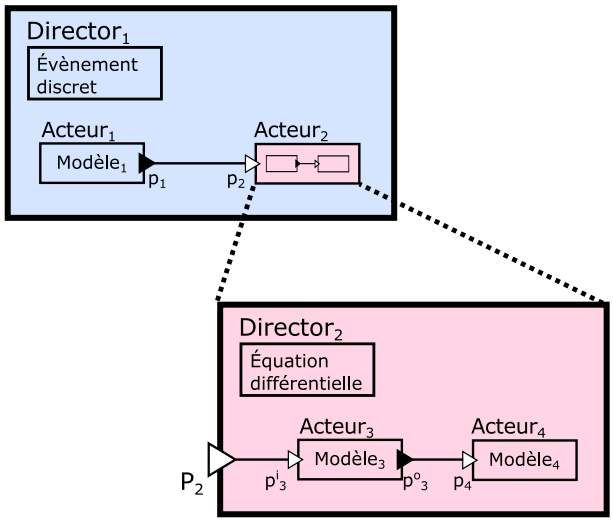


FIGURE 4.4 – Schéma de hiérarchie hétérogène dans Ptolemy II. Inspiré de [Eker et al., 2003]

Ces MoC sont exprimés selon une sémantique abstraite commune, celle des acteurs [Lee,

2010]. Les acteurs, qu'ils soient atomiques ou composites, respectent la même interface abstraite ce qui permet de les manipuler de manière homogène et de faire des hiérarchies.

Ptolemy II repose sur une syntaxe abstraite commune pour définir les multi-modèles, et sur la sémantique abstraite des acteurs pour définir les MoC. L'implémentation d'un MoC (constitué d'un directeur et des récepteurs) est appelé un domaine. De nombreux domaines sont disponibles (Évènement Discret, Automate à état fini...). Cependant, il n'est pas possible de combiner tous les types de formalismes [Goderis et al., 2007].

Propriétés et limites

L'approche de Ptolemy II soulève plusieurs intérêts. Tout d'abord, elle fournit un cadre d'intégration des formalismes hétérogènes via la sémantique abstraite des acteurs. Ensuite, elle permet la réalisation de hiérarchies et l'intégration de modèles issus d'autres logiciels [Müller and Widl, 2013]. L'utilisation d'acteurs dotés de ports assure une modularité au moins au sein d'un même domaine, et une interface graphique est disponible pour faciliter le travail des modélisateurs [Eker et al., 2003].

4.3.3 Le formalisme DEVS

Présentation

DEVS est un formalisme évènementiel proposé par Bernard P. Zeigler en 1976 [Zeigler et al., 2000]. Inspiré de la théorie des systèmes, le formalisme DEVS présente plusieurs propriétés intéressantes [Zeigler et al., 2012].

Nous pouvons citer par exemple les propriétés d'universalité et d'unicité pour la représentation de systèmes évènementiels. Ces propriétés signifient respectivement que tout système évènementiel est représentable par un modèle DEVS et que cette représentation est unique, i.e. il n'existe pas deux modèles DEVS spécifiant le même système évènementiel.

La propriété d'universalité signifie que DEVS est capable de représenter tous les autres formalismes évènementiels. Il est par exemple montré dans [Jacques and Wainer, 2002] l'intégration des Réseaux de Petri dans DEVS. Cependant, cette capacité à représenter d'autres formalismes ne se limite pas aux formalismes évènementiels. Il a été montré notamment dans [Zeigler, 2006] la capacité de DEVS à représenter le formalisme hybride DEV&DESS qui spécifie l'interaction entre modèles discrets et continus. La capacité d'intégration de DEVS le place comme un formalisme pivot pour l'intégration de formalismes hétérogènes et donc pour le développement de multi-modèles multi-formalismes [Vangheluwe, 2000]. Cette capacité d'intégration peut s'observer sur la Figure 4.3 qui représente le graphe de transformation des formalismes utilisé dans $AToM^3$.

Spécification formelle

DEVS définit deux types de modèles : les modèles atomiques et les modèles couplés (modèles composés de plusieurs sous-modèles DEVS). À ces deux spécifications, DEVS ajoute celle d'un simulateur abstrait qui permet de dérouler les comportements des modèles DEVS. La différenciation très claire entre la spécification des modèles et celle du simulateur est l'un des avantages de DEVS, cela permet une séparation claire entre l'activité de modélisation et celle de simulation.

Un modèle atomique DEVS (appelé aussi modèle comportemental) M_i est défini comme suit :

$$M_i = (X_i, Y_i, S, \delta_{ext}, \delta_{in}, \lambda, ta)$$

Où :

- $X_i = \{(p, v) | p \in InPorts_i, v \in V_{X_i}\}$ est l'ensemble des ports d'entrée ($InPorts_i$) et de leurs valeurs admissibles (V_{X_i}).
- $Y_i = \{(p, v) | p \in OutPorts_i, v \in V_{Y_i}\}$ est l'ensemble des ports de sortie ($OutPorts_i$) et de leurs valeurs admissibles (V_{Y_i}).
- S est l'ensemble des états du modèle.
- $\delta_{ext} : Q \times X_i \rightarrow S$ est la fonction de transition externe qui décrit comment le modèle évolue à la réception d'un évènement externe.
  - $Q = (s, e) | s \in S, 0 \leq e \leq ta(s)$ correspond à un état total du modèle (un état de l'ensemble S associé au temps e passé dans cet état).
  - e correspond au temps écoulé depuis la dernière transition d'état.
- $\delta_{in} : S \rightarrow S$ est la fonction de transition interne qui décrit la dynamique interne du modèle (i.e. comment le système change d'état lorsque le temps e passé dans l'état courant atteint la limite).
- $\lambda : S \rightarrow Yi$ est la fonction de sortie qui décrit l'évènement de sortie du modèle selon son état courant. Elle permet de récupérer les évènements de sortie du modèle.
- $ta : S \rightarrow \mathbb{R}^+_{0,\infty}$ est une fonction donnant le temps maximum que le modèle peut passer dans un état. Elle permet notamment de donner le prochain temps auquel le modèle génèrera un évènement et a un rôle crucial dans la synchronisation des modèles DEVS lors de la simulation.

Un modèle atomique M peut donc se voir comme une boite présentant des ports d'entrée et des ports de sortie (voir Figure 4.5).

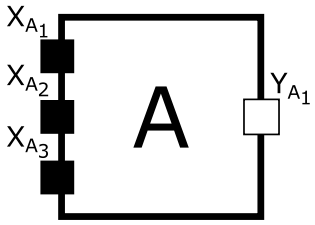


FIGURE 4.5 – Un modèle atomique A

La structure d'un modèle DEVS couplé spécifie un ensemble de sous-modèles, leurs interactions au sein du multi-modèle, et les interactions qu'il est encore possible de créer (i.e. les ports d'entrée/sortie du multi-modèle qui permettront de le réutiliser). Le comportement du multi-modèle DEVS ne dépend que du comportement et de l'interaction de ses sous-modèles.

La spécification du modèle couplé est présentée ci-dessous :

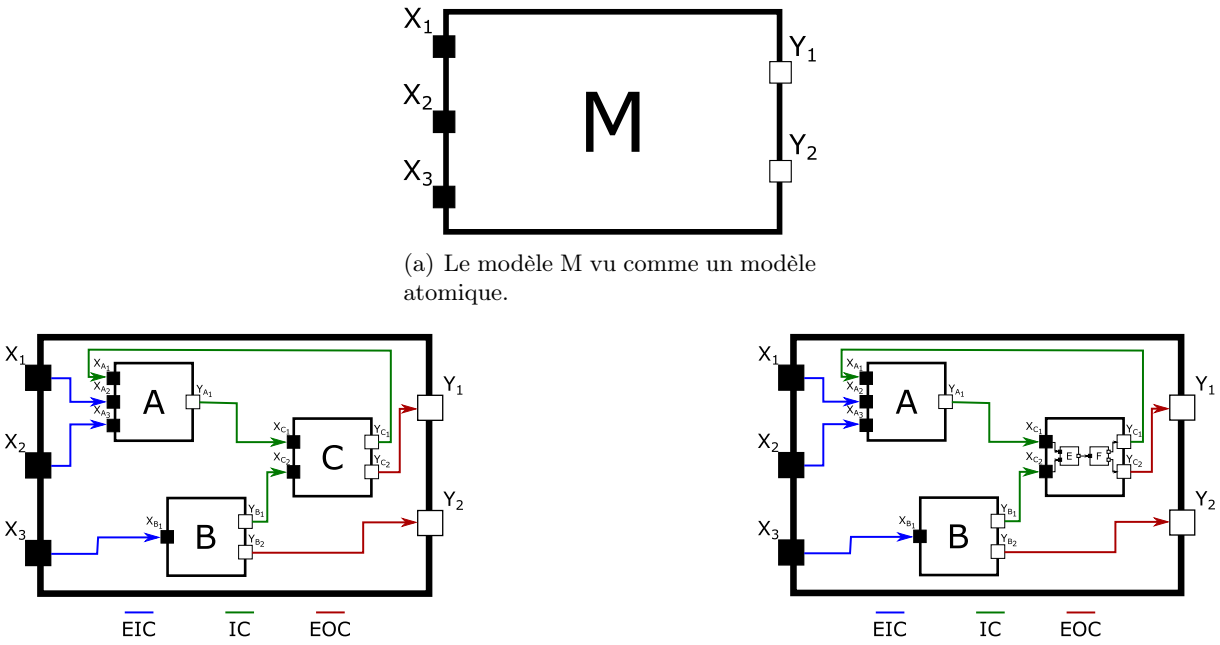
$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select)$$

Avec :

- D est l'ensemble des noms des sous-modèles.
- $X = \{(p, v) \| p \in IPorts, v \in X_p\}$ est l'ensemble des ports d'entrée, i.e. des couples (port, valeur d'entrée). $IPort$ correspond à l'ensemble des ports d'entrée et X_p est l'ensemble des valeurs acceptables sur le port p.
- $Y = \{(p, v) \| p \in OPorts, v \in Y_p\}$ est l'ensemble des sorties, i.e. des couples (port, valeur de sortie). $OPort$ correspond à l'ensemble des ports de sortie et Y_p à l'ensemble des valeurs possibles pour le port p.

— $\{M_d | d \in D\}$ est l'ensemble des sous-modèles DEVS (qui peuvent être atomiques ou couplés).
— $EIC = \{((N, ip_N), (d, ip_d)) \| ip_N \in IPorts, d \in D, ip_d \in IPorts_d\}$ (*External Input Coupling*) est l'ensemble des connexions entre les ports d'entrée du multi-modèles et les ports d'entrée de ses sous-modèles.
— $EOC = \{((d, op_d), (N, op_N)) \| op_N \in OPorts, d \in D, op_d \in OPorts_d\}$ (*External Output Coupling*) est l'ensemble des connexions entre les ports de sortie des sous-modèles et les ports de sortie du multi-modèle.
— $IC = \{((a, op_a), (b, ip_b)) \| a, b \in D, op_a \in OPorts_a, ip_b \in IPorts_b\}$ (*Internal Coupling*), est l'ensemble des couplages internes entre les sous-modèles.
— $Select : 2^D - > D$ est une fonction permettant de résoudre les conflits lorsque plusieurs modèles souhaitent exécuter leurs évènements internes au même temps. La fonction détermine l'ordre d'exécution.

La spécification d'un modèle couplé DEVS nous permet de réutiliser le modèle atomique A précédent et de l'utiliser au sein d'un multi-modèle M (voir Figure 4.6).



(a) Le modèle M vu comme un modèle atomique.

(b) Le modèle M vu comme un modèle couplé.

(c) Le modèle M contenant un autre modèle couplé C.

FIGURE 4.6 – Un modèle couplé M composé de trois sous-modèles A, B, C.

La définition du modèle couplé permet de l'utiliser de la même manière qu'un modèle atomique, i.e. il est possible d'utiliser un modèle couplé comme composant d'un modèle atomique. Cette propriété de fermeture par couplage signifie que du point de vue du simulateur abstrait DEVS, un modèle atomique est identique à un modèle couplé. DEVS assure aussi que le choix de la structure n'impacte pas les résultats de simulation ce qui est fondamentale pour garantir une exécution correcte.

Simulateur abstrait

DEVS fournit des spécifications pour les modèles atomiques et pour les modèles couplés, mais il fournit aussi un algorithme de simulation abstrait pour dérouler leur comportement. Ce simulateur est basé sur cinq méthodes :

— *initialize* : initialise le modèle.
— *processInternalEvent* : exécute un évènement interne au modèle.
— *processExternalEvent* : exécute un évènement externe entrant dans le modèle.
— *getOutputEvent* : renvoi un évènement de sortie du modèle pour l'envoyer à un autre.
— *getNextInternalEventTime* : renvoi le temps du prochain évènement interne.

Hormis la fonction d'initialisation, les autres font directement référence aux fonctions δ_{in}, δ_{ext}, λ et tn des modèles atomiques.

Note : Lors d'une co-simulation basée sur le simulateur DEVS, ces 5 méthodes correspondent à l'interface d'interaction avec les modèles.

Intégration de formalisme

La capacité d'intégration de DEVS le positionne comme une solution de choix pour la modélisation multi-formalisme [Vangheluwe, 2000]. Deux approches sont envisageables [Quesnel et al., 2005] :

— ***mapping***. Le *mapping* consiste à trouver une équivalence totale entre deux formalismes, i.e. de trouver une bijection permettant de passer de l'un à l'autre. C'est ce qui est réalisé lors de l'intégration des Réseaux de Petri dans DEVS [Jacques and Wainer, 2002]. C'est aussi ce qui permet des transformations de formalisme comme vu dans $AToM^3$.
— ***wrapping***. Le *wrapping* consiste à créer une interface fonctionnelle autour d'un modèle écrit dans un formalisme particulier pour pouvoir le manipuler comme s'il était du formalisme encapsulateur.

Le *mapping* peut se concrétiser par une approche de transformation de formalisme tandis que le *wrapping* peut se concrétiser par une approche d'encapsulation formelle (et logicielle).

Propriétés et limites

Les propriétés de DEVS en font un formalisme pivot pour la multi-modélisation multi-formalisme [Vangheluwe, 2000]. Son simulateur abstrait, sa capacité d'encapsulation et ses multiples extensions en font aussi un outil formel de choix pour la co-simulation. De plus, la propriété de fermeture par couplage a deux intérêts pratiques :

1. il est possible de réutiliser modèles atomiques et couplés pour construire incrémentalement un multi-modèle de système complexe.
2. Il est possible de remplacer un modèle atomique par un modèle couplé, i.e. il est possible de spécialiser, de décomposer, les modèles atomiques en sous-modèles en interaction.

Ces deux propriétés de couplage et de décomposition sont au cœur de *System Entity Structure* (SES), un système de représentation des hiérarchies. SES permet de représenter un modèle de système en spécifiant ses sous-composants, leurs interactions et les alternatives possibles. En utilisant un formalisme doté de la propriété de fermeture par couplage, il devient possible de créer des bibliothèques de modèles atomiques et couplés réutilisables. Une structure comme SES permet alors, dans un second temps, d'organiser ces composants.

Le formalisme DEVS et ses extensions ont été implémentés dans de nombreux logiciels de M&S. Rien que parmi la communauté francophone travaillant sur DEVS, nous pouvons citer [10] VLE, PythonPDEVS et DEVSimPy, Quartz, fwkDEVS et enfin ProDEVS. La plupart de ces outils se sont spécialisés sur une extension particulière de DEVS (par exemple *Parallel DEVS* dans le cas de PythonPDEVS) et sur une implémentation dans un langage particulier.

Il faut noter que les différents logiciels de M&S basés sur DEVS ne sont pas interopérables et ne peuvent généralement pas être utilisés en collaboration au sein d'une co-simulation [Wainer et al., 2010b]. Cela est dû aux hétérogénéités logicielles et à l'absence de méthode de développement standard pour partager les modèles respectant DEVS.

Chaque logiciel basé sur DEVS propose des fonctionnalités de développement spécifiques qui dépendent, entre autre, de l'état d'avancement du projet. Nous présentons ci-après le logiciel VLE qui est l'une des implémentations de DEVS les plus abouties.

L'exemple de VLE

VLE (*Virtual Laboratory Environment*) [Quesnel et al., 2007] est un environnement de M&S basé sur le formalisme DEVS. Développé pour l'écriture de modèles de systèmes complexes, il a notamment été utilisé dans le domaine de l'écologie marine (par exemple pour le modèle DynFish [Versmisse et al., 2006]).

Pour permettre le développement multi-formalisme, VLE se repose sur DEVS et sur les formalismes qui y sont déjà intégrés (e.g. *Dynamic DEVS*, *CellDEVS*, *Quantified State System*...). Le cœur du simulateur est basé sur PDEVS (*Parallel DEVS*) ce qui permet de facilement distribuer les calculs.

L'intérêt de VLE est qu'il propose un ensemble de solutions pour l'ensemble du cycle de M&S [Quesnel et al., 2009] :

- Pour la modélisation, il propose GVLE, une interface graphique facilitant la modélisation et évitant les erreurs d'implémentation. Cette interface permet aussi de créer des plans d'expérience.
- L'observation est géré via un outil dédie, EOV (*Eye Of VLE*).
- L'analyse peut être faite par un outil dédié, AVLE (*Analysis for VLE*), qui peut être lié au logiciel de statistique R.

L'approche utilisée, grâce à la base formelle DEVS permet la modularité et la construction hiérarchique de multi-modèles. Il faut noter que les hiérarchies dans VLE ne sont que des hiérarchies de construction, du point de vue de la simulation le multi-modèle est exécuté "à plat" [Quesnel, 2006] pour des gains en temps de simulation.

4.3.4 MECSYCO

Présentation

MECSYCO [11] (*Multi-agent Environment for Complex SYstem CO-simulation*) [Camus, 2015] est un intergiciel de co-simulation développé au sein du LORIA/Inria et qui a fait l'objet d'un partenariat avec EDF R&D pour la M&S de réseaux électriques intelligents [Vaubourg et al., 2015]. L'objectif principal de cette plateforme est l'intégration de modèles issus de différents logiciels pour pouvoir les coupler en multi-modèles et effectuer des co-simulations.

10. Outils listés dans https://devs-network.org/node/16
11. http://mecsyco.com/fr/

MECSYCO permet d'intégrer des modèles issus de NetLogo [Paris et al., 2017a], de ns-3 et d'Omnet++/Inet [Vaubourg et al., 2016][Vaubourg, 2017], ou sous forme de FMU pour la co-simulation [Vaubourg et al., 2015] comme pour l'échange de modèle [Camus et al., 2018].

Pour faire face au challenge d'interopérabilité lié à la co-simulation, MECSYCO repose :

1. au niveau technique sur une stratégie d'encapsulation logicielle, et
2. au niveau formel sur une stratégie d'encapsulation basée sur DEVS.

De plus, l'architecture de la plateforme se base sur le paradigme AA4MM (*Agent and Artifact for Multi-Modeling* [Siebert, 2011] inspiré par l'approche *Agent and Artifact* [Ricci et al., 2007] et de la notion d'agent modèle proposée dans [Bonneaud, 2008]. Le choix d'une architecture basée sur les systèmes multi-agents est motivé par le lien entre les systèmes complexes (ensemble d'entités hétérogènes en interaction), les co-simulations (ensemble de simulateurs en interaction) et les systèmes multi-agents (ensemble d'agents en interaction). Cela permet d'une part de voir une co-simulation comme une société d'agents en interaction, et d'autre part de réifier les concepts liés à la multi-modélisation et à la co-simulation (voir 5.2.1).

Propriétés et limites

MECSYCO offre un cadre rigoureux pour résoudre les problèmes d'hétérogénéité logicielle, formelle, mais aussi de représentation (du temps et des données) afin de connecter plusieurs modèles hétérogènes issus de logiciels de M&S différents. Nous pouvons ainsi construire de manière modulaire des multi-modèles à partir de modèles issus d'autres logiciels. Les co-simulations sont ensuite effectuées de manière décentralisée et parallèle, les calculs peuvent donc être distribués sur plusieurs machines. Cependant, MECSYCO ne permet pas la réutilisation des multi-modèles développés en son sein. Il n'est donc pas possible de construite hiérarchiquement des multi-modèles. Il faudrait pour cela associer une démarche de multi-modélisation hiérarchique à MECSYCO.

Une approche d'Ingénierie Dirigée par les Modèles a été proposée dans [Camus, 2015] pour faciliter l'utilisation de MECSYCO et automatiser le passage du multi-modèle, exprimé avec AA4MM, à la co-simulation. Cependant, celle-ci est peu extensible, i.e. elle permet l'utilisation des modèles déjà intégrés dans MECSYCO mais il est difficile d'en ajouter de nouveaux tout comme il est difficile de la faire évoluer pour y intégrer des hiérarchies.

4.4 Conclusion

4.4.1 Constats sur le couplage de modèles hétérogènes

Le premier constat est que le problème n'est pas la capacité à effectuer des co-simulations, il est toujours possible d'effectuer des co-simulations de manière ad-hoc. Le problème, c'est la généralisation des co-simulations par le développement de démarches, méthodes et outils qui permettent d'obtenir des propriétés telles que :

— la modularité des composants,
— la construction hiérarchique,
— la simplicité d'utilisation, mais aussi
— l'extensibilité,
— la vérification des modèles et multi-modèles,
— la validation des modèles et des multi-modèles,
— la réutilisation des composants,
— la répétabilité des expériences,

— la reproductibilité des expériences,
— le passage à l'échelle,
— l'ouverture vis-à-vis des autres approches existantes,
— ...

La plupart de ces propriétés sont décomposables en plusieurs aspects tout comme la notion d'interopérabilité est décomposée dans l'échelle LCIM (voir Figure 3.2). Certaines caractéristiques sont même interconnectées (par exemple l'interopérabilité et l'extensibilité d'un composant sont liées à sa faculté à être réutilisé).

Le deuxième constat est qu'il n'y a pas, dans la littérature, d'approches qui permettent l'intégration et la composition de modèles hétérogènes dans un cadre générique. Au niveau de la composition syntaxique, des solutions existent pour connecter des modèles hétérogènes et effectuer des co-simulations, mais le niveau sémantique (prouver à priori que le couplage de deux modèles hétérogènes est valide) est difficile à atteindre.

Notons tout de même que d'un point de vue pratique, qui concerne les fonctionnalités et propriétés gravitant autour de la composition et de la réutilisation de modèles existants, de nombreux travaux soulignent l'importance de documenter les modèles et d'utiliser des méta-données pour permettre la réutilisation des éléments et l'automatisation de tâches [Morse, 2004][Petty and Weisel, 2003b][Breunese et al., 1998].

4.4.2 Multiples approches

Beaucoup d'approches développées ces dernières années tentent de résoudre les problèmes d'hétérogénéité des modèles en se plaçant à différents niveaux de question. Les approches purement logicielles comme HLA et FMI, mettent en placent des règles et des standards permettant à des logiciels différents d'effectuer des co-simulations. Cependant, ils ne proposent pas de guide quant à l'intégration formelle des modèles. Les approches basées sur l'étude des formalismes proposent des opérations (comme la transformation ou l'encapsulation de formalisme) qui autorisent des approches multi-formalismes mais qui ne sont généralement pas pensées pour réutiliser des modèles existants et issus d'autres logiciels.

Ce qu'il faut retenir de ces travaux, c'est la multiplicité des solutions possibles pour rendre des systèmes interopérables, que ce soit aux niveaux technique ou formel. Bien que les approches soient variées, cela ne signifie pas qu'elles ne peuvent pas être utilisées en collaboration, directement ou après quelques évolutions. Nous pouvons citer comme exemples l'utilisation conjointe de HLA et FMI [Awais et al., 2013], DEVS/HLA [Zacharewicz, 2006][Zacharewicz et al., 2008], DEVS et FMI [Camus et al., 2018] et Ptolemy II et FMI [Müller and Widl, 2013].

4.4.3 Positionnement

MECSYCO a plusieurs particularités par rapport aux autres solutions de M&S de systèmes complexes :

— Contrairement aux approches d'interopérabilité logicielle, MECSYCO se base sur le formalisme DEVS pour effectuer l'intégration et gérer la co-simulation. Il bénéficie donc de la rigueur de ce formalisme, de ses propriétés et de son abondante littérature.
— Vis-à-vis des approches multi-formalismes et des autres approches DEVS telles VLE, MECSYCO a la spécificité de ne pas chercher à développer ses propres modèles atomiques (même si cela est possible) mais plutôt à intégrer ceux issus d'autres logiciels et à gérer les hétérogénéités.

Ces particularités se traduisent par le développement d'un intergiciel de co-simulation, l'adoption d'une stratégie d'encapsulation logicielle et formelle basée sur DEVS et l'utilisation du méta-modèle AA4MM qui permet de résoudre les problèmes d'hétérogénéités de représentation pouvant être présents dans les modèles intégrés. C'est l'ensemble de ces éléments qui fait l'originalité de MECSYCO dans la littérature [Camus, 2015].

Cependant, MECSYCO souffre de plusieurs limites concernant le cycle complet de modélisation et simulation, à commencer par l'impossibilité de hiérarchiser les multi-modèles (propriété fondamentale pour gérer la complexité). Notre travail poursuit les efforts autour de l'approche MECSYCO pour repousser ses limites.

La Figure 4.7 propose une synthèse du positionnement de MECSYCO par rapport aux autres solutions présentées. Nous y identifions, de par notre expérience de MECSYCO et de l'étude des autres approches existantes, cinq étapes nécessaires à la simulation d'un multi-modèle formé de composants hétérogènes. La première fait écho à l'étape de conceptualisation en M&S, elle correspond à l'identification des différents éléments à modéliser pour représenter le système complexe. Ces différents composants seront modélisés à l'étape de modélisation atomique puis seront intégrés pour former un ensemble de modèles composables (au niveau logiciel et syntaxique). Ensuite les modèles composants sont utilisés pour construire le multi-modèle qui sera lui-même exploité pour effectuer la simulation.

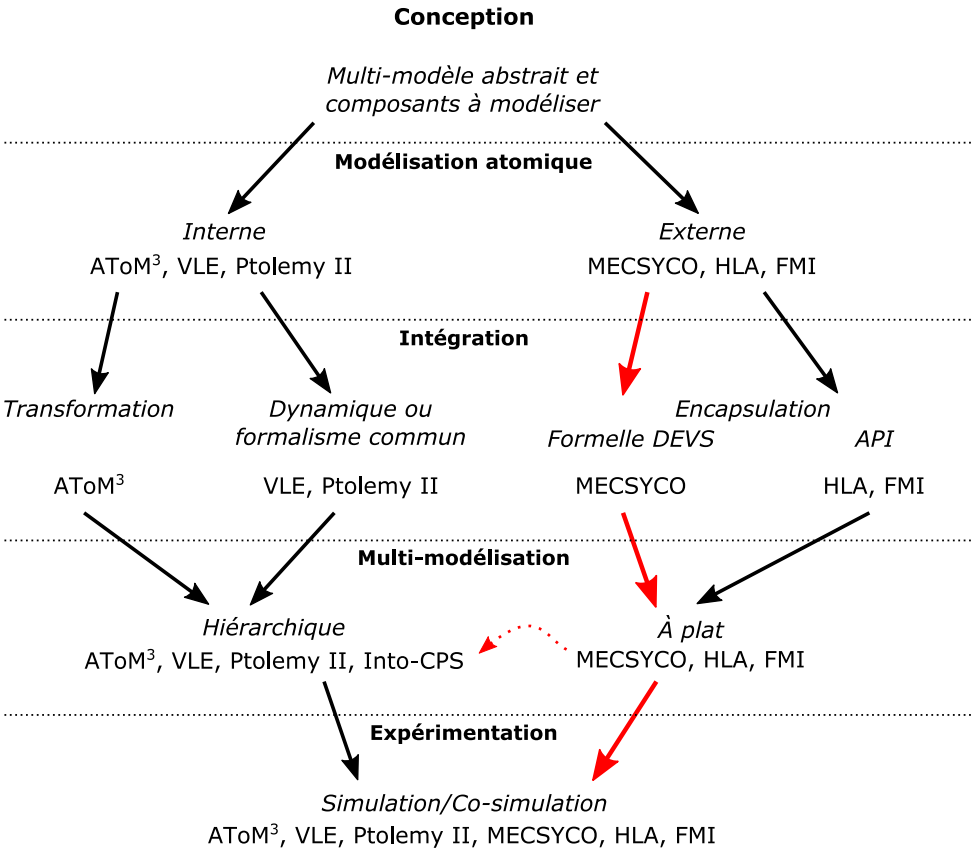


FIGURE 4.7 – Synthèse du positionnement de MECSYCO (en rouge les points impactés par notre travail).

Notes pour la Figure 4.7 :

— FMI désigne la plupart des approches basées sur la co-simulation de FMU (dont Into-

CPS). Nous faisons cependant la séparation au niveau de la multi-modélisation parce que CPS, via l'utilisation de SysML, propose une modélisation hiérarchique contrairement à la plupart des autres approches de type FMI (e.g. Daccosim [Galtier et al., 2015]).

— Nous identifions en rouge les étapes impactées par nos contributions, i.e. les étapes d'intégration, de multi-modélisation (qui permettront de passer à une version hiérarchique) et d'expérimentation.

Nous détaillons dans le prochain chapitre les particularités de l'approche MECSYCO ce qui nous amènera à notre proposition.

5

Cadre des travaux et proposition

Ce chapitre est dédié à la présentation de notre proposition. Étant donné que celle-ci reprend et étend le travail effectué sur l'intergiciel MECSYCO, nous commençons par rappeler la problématique de multi-modélisation et co-simulation ainsi que le positionnement de MECSYCO au sein de celle-ci. Nous détaillons et critiquons ensuite l'intergiciel, d'une part pour présenter les éléments nécessaires à la compréhension et d'autre part pour mettre en avant ce que nous apportons.

Ces éléments nous amènent ensuite à notre proposition qui commence par l'identification des 5 étapes pour la multi-modélisation et co-simulation. Nous proposons (1) d'associer à MECSYCO une démarche descriptive, basée sur les étapes précédentes, pour permettre la multi-modélisation hiérarchique et la co-simulation de modèles hétérogènes et (2) d'y ajouter un environnement de M&S basé sur des langages dédiés pour soutenir l'approche et faciliter son utilisation.

5.1 Rappel de la problématique

Le contexte général de nos travaux est la modélisation et simulation de systèmes complexes par une approche de multi-modélisation et co-simulation. Comme nous l'avons vu, la difficulté de cette approche est la définition de méthodes génériques associée au développement d'outils qui assurent des propriétés telles la modularité et la modélisation hiérarchique.

Dans ce contexte, MECSYCO se positionne comme une approche intégrative. L'accent est mis sur l'intégration de modèles issus de différents logiciels pour construire des multi-modèles et les co-simuler. L'idée sous-jacente est qu'il est pertinent et profitable de réutiliser les simulateurs existants qui proposent déjà de nombreuses solutions dans des domaines particuliers. L'intégration repose sur une stratégie d'encapsulation logicielle et formelle basée sur DEVS. Une fois intégrés, les modèles sont vus comme des modèles DEVS atomiques ce qui nous fait profiter de la modularité de DEVS. Les multi-modèles formés sont co-simulés de manière décentralisée, permettant un certain passage à l'échelle.

Cependant, MECSYCO ne permet pas la réutilisation des multi-modèles développés en son sein. Il est donc impossible de construire des multi-modèles de manière hiérarchique, i.e. en composant des multi-modèles MECSYCO qui seront réutilisables comme des modèles atomiques (propriété de fermeture par couplage). L'objectif de MECSYCO étant la multi-modélisation et co-simulation de systèmes complexes, cette capacité de hiérarchisation des multi-modèles est fondamentale et doit être ajoutée tout en conservant les propriétés actuelles de la plateforme.

Notre problématique est donc la suivante : dans le cadre du projet MECSYCO, proposant une solution d'intégration de modèles basée sur l'encapsulation en DEVS et qui a pour propriétés la

modularité, la décentralisation et le passage à l'échelle en plus de la généricité de l'encapsulation DEVS, **comment permettre à MECSYCO de définir hiérarchiquement des multi-modèles à partir de modèles hétérogènes pour les co-simuler tout en conservant les propriétés initiales de la plateforme ?**

La question précédente est double, elle regroupe (1) le besoin d'une démarche qui guide l'activité de multi-modélisation et co-simulation dans MECSYCO et qui permette une construction hiérarchique, et (2) la nécessité d'accompagner cette démarche avec un outil logiciel permettant sa mise en pratique et compatible avec les propriétés de MECSYCO.

5.2 AA4MM et MECSYCO

La contribution de la thèse reprend et peaufine les bases conceptuelles, formelles et logicielles posées par les travaux sur MECSYCO. Cette section va donc rappeler les éléments essentiels à la compréhension des contributions et de la réflexion associée.

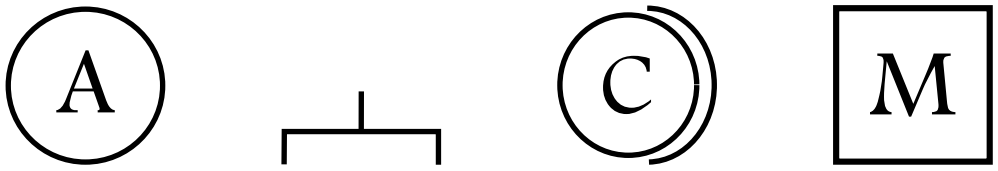
5.2.1 Méta-modèle AA4MM

Cette section sert d'introduction aux différentes entités appartenant au méta-modèle (voir la Définition 18 dans le cadre du paradigme multi-agent) AA4MM [Siebert, 2011] et qui servent de base à la représentation des co-simulations dans MECSYCO.

Définition 18 (Méta-modèle)**.** Un méta-modèle multi-agent définit l'ensemble des concepts qui composent un système multi-agent, leurs relations et leurs rôles. [Siebert, 2011]

Principe général

L'idée générale du méta-modèle AA4MM était d'utiliser des principes issus de la communauté multi-agent pour réifier et proposer des solutions aux problèmes d'hétérogénéités posés par la multi-modélisation et le couplage de simulateurs. Il repose notamment sur le paradigme A&A (*Agent&Artifact*) [Ricci et al., 2007], qui consiste à représenter l'environnement des agents, i.e. l'ensemble des éléments avec lesquels ils peuvent interagir, au moyen d'entités passives appelées "artéfact". Cela signifie que chacune des actions des agents (communication, action sur l'environnement) sera réifiée grâce à un artéfact particulier. La Figure 5.1 présente les entités utilisées pour construire un multi-modèle.



(a) Agent modèle (b) Artéfact de modèle (c) Artéfact de couplage (d) Modèle

FIGURE 5.1 – Les entités du méta-modèle AA4MM

Dans AA4MM, un multi-modèle est vu comme une "société de modèles en interaction" (expression tirée de [Bonneaud, 2008]). Chaque multi-modèle est représenté intuitivement comme une société d'agent en interaction, et est implémenté au niveau logiciel comme un ensemble de programmes en interaction. Chaque couple (modèle, simulateur) est géré par un agent qui va se charger de son exécution et de sa synchronisation avec les autres couples (modèle, simulateur),

i.e. les agents conduisent la co-simulation. Ils interagissent avec les modèles issus d'autres logiciels grâce à des artéfacts de modèle et interagissent entre eux grâce à des artéfacts de couplage. La Figure 5.1 présente les entités qui sont utilisées pour construire intuitivement un multi-modèle AA4MM (voir Figure 5.2).

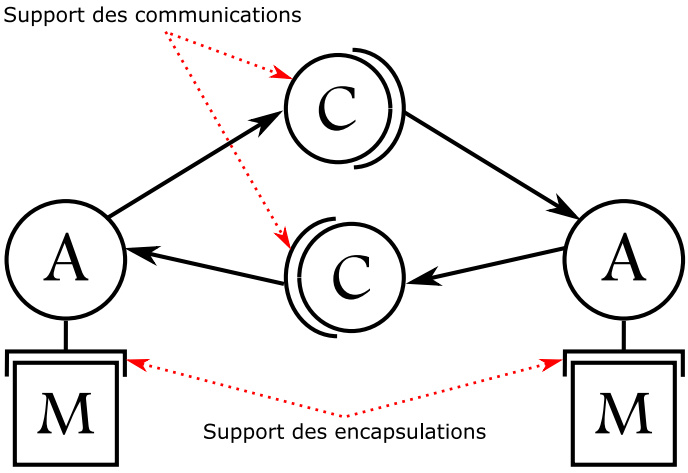


FIGURE 5.2 – Représentation intuitive d'un multi-modèle via AA4MM

Détail des entités

Agent modèle :

Les agents modèles, que nous nommerons aussi seulement "agents" ou "m-agents" (voir Figure 5.1(a)), sont les entités dynamiques de AA4MM. Chaque m-agent va interagir avec un modèle issu d'un autre logiciel au moyen d'un artéfact de modèle. Par l'intermédiaire de ces artéfacts de modèle, les agents sont capables de piloter la simulation des modèles. Les agents modèles interagissent entre eux grâce à des artéfacts de couplage, ces interactions leur permettent de synchroniser les simulations de leur modèle et d'échanger des données. La co-simulation d'un multi-modèle correspond alors à la dynamique d'un système multi-agent.

Artéfact de modèle :

Les artéfacts de modèle (Figure 5.1(b)) sont des entités passives manipulées par des agents modèles. Ce sont les supports de l'encapsulation logicielle et formelle qui permettent aux m-agents d'interagir sur les modèles. Les artéfacts de modèle n'ont pas vocation à être spécifiques à un modèle, ils sont généralement compatibles avec un ensemble de modèles tirés d'un logiciel de M&S particulier et simulables avec un simulateur spécifique. Par exemple, l'artéfact de modèle FMI est compatible avec n'importe quelle FMU de type co-simulation.

Au niveau formel, l'encapsulation des modèles est basée sur DEVS. Cela signifie qu'après encapsulation, les modèles intégrés sont vus comme des modèles atomiques DEVS et sont manipulables grâce aux cinq méthodes du simulateur abstrait DEVS (voir 4.3.3).

Artéfact de couplage :

Les artéfacts de couplage (Figure 5.1(c)) sont les entités servant de support à la communication entre agents. Ils sont orientés, ce qui signifie que chaque artéfact de couplage est lié à un agent émetteur qui dépose des évènements et à un agent receveur qui lit les évènements. Le fonctionnement des artéfacts de couplage est similaire à celui d'une boite aux lettres, i.e. ils contiennent une pile ordonnée des évènements devant être consommés par l'agent receveur.

Il faut noter qu'en tant que support des communications, les artéfacts de couplage sont aussi le support d'opérations destinées à résoudre les problèmes d'hétérogénéité de représentation. Il y a deux types d'opérations :

- les opérations de données qui permettent d'effectuer des transformation sur la donnée échangée au cours de l'évènement (par exemple si la donnée est une distance exprimée en km par l'agent envoyeur, mais qu'une distance en m est attendue par l'agent récepteur).
- les opérations de temps qui permettent de modifier systématiquement l'étiquette temporelle des évènements transitant par l'artéfact de couplage. Cela permet de résoudre les problèmes d'hétérogénéité temporelle, comme par exemple lorsqu'un agent a un temps de simulation exprimé en s, alors qu'un autre est en ns.

Modèle :

Les entités "modèles" (voir Figure 5.1(d)) de AA4MM sont des modèles issus d'autres logiciels de M&S et intégrés à la plateforme. Ils représentent en réalité un couple (modèle, simulateur), i.e. ils comprennent la partie statique du modèle (ses variables d'état) et la partie dynamique (son algorithme de simulation).

Synthèse

L'avantage de AA4MM est sa capacité à représenter les différents types d'interaction liés à la co-simulation, i.e. les interactions entre l'intergiciel (les agents) et les logiciels intégrés (les modèles) et les interactions au sein de l'intergiciel (les communications entre agents). La réification de ces interactions permet de spécifier où les problèmes d'hétérogénéité vont être résolus. **Les artéfacts de modèle sont les supports pour la résolution des hétérogénéités logicielle et formelle tandis que les artéfacts de couplage sont le lieu de résolution des hétérogénéités de représentation des données et du temps**.

5.2.2 Spécifications MECSYCO

L'intergiciel MECSYCO [Camus, 2015] est un environnement de co-simulation issu de la volonté d'associer l'expressivité du méta-modèle AA4MM avec les avantages formels de DEVS. MECSYCO utilise DEVS comme pivot pour l'intégration de formalismes hétérogènes, ce qui permet une démarche d'intégration rigoureuse alimentée par l'abondante littérature autour de ce formalisme.

Concernant son architecture logicielle, celle-ci est basée sur AA4MM pour bénéficier de son expressivité. Une interface graphique basée sur les principes de l'Ingénierie Dirigée par les Modèles (IDM) a été développée en complément. Elle automatise le passage d'un schéma de multi-modèle AA4MM au programme et à la co-simulation [Camus, 2015], évitant ainsi les erreurs de programmation des utilisateurs. Cependant, cette démarche d'IDM n'est pas extensible, et n'a pas pu être conservée lors du développement de nouveaux outils dans MECSYCO.

Enfin, en complément des spécifications héritées de DEVS, les travaux de [Camus, 2015] ajoutent l'algorithme de Chandy-Misra-Bryant [Chandy and Misra, 1979][Bryant, 1979] comme master de co-simulation. Le choix de cet algorithme décentralisé et conservatif permet d'être le plus générique possible en n'imposant pas de mécanisme spécifique tel que le *roll-back* lors de l'intégration des simulateurs [Camus et al., 2018].

MECSYCO existe en deux versions : Java et C++. La version Java, issue de la thèse de Benjamin Camus [Camus, 2015], est celle sur laquelle nous nous concentrons pour notre travail. La version C++ a été développée au cours de la thèse de Julien Vaubourg [Vaubourg, 2017] pour l'intégration des simulateurs ns-3 et Omnet++/Inet.

5.2.3 Étapes du développement dans MECSYCO

Nous allons maintenant dérouler les différentes étapes nécessaires au développement d'un multi-modèle dans MECSYCO. Ces différentes étapes serviront de base pour la démarche de multi-modélisation proposée en 6.1.

Il faut noter que ces étapes sont liées à l'ordre d'implémentation des éléments et non à la démarche de conception choisie (*bottom-up* ou *top-down*).

L'intégration des simulateurs

La construction d'un multi-modèle MECSYCO commence par l'encapsulation logicielle et formelle des environnements de M&S que nous souhaitons utiliser pour développer les modèles atomiques. Si nous nous replaçons sur l'échelle LCIM, cette étape correspond à la résolution des problèmes d'intégrabilité et d'interopérabilité. Dans MECSYCO, cela passe par le développement d'artéfacts de modèle spécifiques s'ils n'existent pas encore.

À cette étape, l'encapsulation peut être guidée par l'un des nombreux travaux concernant l'intégration de formalisme dans DEVS. Cette intégration formelle entraine au niveau logiciel, que l'artéfact de modèle implémente les cinq méthodes héritées du simulateur abstrait DEVS.

Une fois cette étape réalisée, nous pouvons utiliser les modèles issus des environnements de M&S intégrés comme si c'était des modèles DEVS pour effectuer nos co-simulations.

Construction du multi-modèle

Une fois les artéfacts de modèles développés, il faut décrire le multi-modèle. Cela consiste à écrire un programme Java qui instancie les artéfacts de modèle associés à leur modèle, qui les connecte entre eux suivant les interactions voulues, et qui décrit l'initialisation des paramètres des modèles et de la co-simulation.

C'est lors de l'étape de connexion des modèles que sont choisies (et développées si nécessaire) les opérations sur les évènements échangés qui servent à résoudre les problèmes d'hétérogénéité de représentation.

À la fin de cette étape, nous avons donc une classe Java qui représente la co-simulation du multi-modèle dans MECSYCO.

Des exemples de construction de multi-modèles MECSYCO sont disponibles sur `http://www.mecsyco.com/dev/`, e.g. La rubrique "*Getting started 2*" donne le code pour la construction et la co-simulation d'un multi-modèle représentant le système dynamique de Lorenz. Celui-ci est aussi disponible en annexe A.

Co-simulation et expérimentation

Le lancement de la co-simulation correspond alors à l'exécution de la classe Java développée juste avant, avec le jeu de paramètres voulu. Notons que nous pouvons identifier 3 catégories de paramètres :

1. Les paramètres des modèles qui correspondent au cadre expérimental, modifiant donc le sens de ce qui est simulé.
2. Les paramètres des simulateurs des modèles, qui ne modifient que la dynamique des modèles et pas ce qui est représenté.
3. Les paramètres de l'algorithme de co-simulation, l'algorithme de Chandy-Misra-Bryant dans notre cas. Ils impacteront la manière dont les agents sont synchronisés (e.g. le délai minimum de propagation des évènements qui influence la synchronisation des modèles).

5.2.4 Bilan

Nous avons présenté dans cette section les bases conceptuelles, formelles et logicielles de MECSYCO. Nous en faisons ci-dessous un bilan où nous énonçons les critiques et les points d'amélioration que nous avons relevés. Ce bilan sert de base pour notre proposition.

Expression du multi-modèle

Au cours des différentes étapes de développement d'une co-simulation MECSYCO, la structure du multi-modèle n'est jamais exprimée en dehors du code. Il en résulte qu'il n'y a pas de représentation du multi-modèle permettant d'effectuer des traitements automatiques (analyse, vérification de propriétés, ...). Exprimer la structure du multi-modèle sera une étape nécessaire pour permettre sa réutilisation et donc la construction hiérarchique de multi-modèles à partir de ceux déjà développés.

Limite dans l'intégration des modèles atomiques

Au cours du développement, bien qu'il y ait une étape d'encapsulation des simulateurs, il n'apparait pas d'étape dédiée à l'intégration des modèles atomiques. En effet, l'intégration d'un modèle issu d'un autre logiciel ne se limite pas au développement d'un artéfact de modèle. Cet artéfact de modèle est destiné à l'intégration du simulateur de l'environnement de M&S que nous souhaitons utiliser. L'artéfact de modèle ne doit pas être spécifique à un modèle particulier mais à un simulateur. Il reste donc à spécifier les éléments spécifiques à chaque modèle, e.g. les noms des ports d'entrée/sortie et les valeurs des évènements admissibles, les noms et valeurs possibles des paramètres, les hypothèses de fonctionnement (ou de validité) associées. Dans la démarche précédente, il est sous-entendu que le programmeur doit connaitre ces différents éléments particuliers aux modèles lorsqu'il code le multi-modèle. Comme ils ne sont jamais exprimés sous une forme concrète, il n'est pas possible d'automatiser des traitements ou d'effectuer des vérifications. La réutilisabilité des modèles s'en trouve atténuée : Comment un autre modélisateur peut-il réutiliser les modèles sans connaître leur interface ?

Il faut noter que certains outils documentent déjà l'interface de leur modèle pour pouvoir les échanger et effectuer des co-simulations, c'est le cas de ceux respectant la norme FMI.

Notons que cette limite dans l'intégration des modèles entraine que pour les modèles issus de NetLogo [Wilensky, 1999], un simulateur multi-agents, il n'existe pas d'artéfact de modèle générique. Dit autrement, il y a un m-artéfact pour chaque modèle NetLogo que nous utilisons dans MECSYCO.

Spécification incomplète des m-artéfacts

L'utilisation d'artéfacts de modèle permet d'avoir une vision homogène des modèles au niveau de la dynamique de co-simulation (ils sont tous vus comme des modèles DEVS grâce à l'encapsulation formelle et sont simulés comme tels). Cependant au niveau logiciel, la manière d'instancier ces artéfacts de modèle n'est pas standardisée et dépend des choix des programmeurs. Dit autrement, au niveau du code, les signatures des méthodes liées à la dynamique de co-simulation sont clairement spécifiées et communes à chaque m-artéfact, tandis que les signatures des constructeurs ne sont pas définies et sont donc particulières à chaque m-artéfact. En conséquence, bien qu'au niveau de l'exécution de la co-simulation, les artéfacts de modèles soient homogènes, leur instanciation ne l'est pas.

De même, il n'y a pas d'indication sur la manière d'utiliser la méthode d'initialisation et donc sur comment intégrer les paramètres des modèles ou ceux des algorithmes de simulation internes aux modèles.

Ce manque de spécification logicielle fait qu'il n'est pas possible de manipuler des m-artéfacts de manière standardisée dans un environnement. C'est en partie pour cela qu'une approche d'IDM telle que présentée dans [Camus, 2015] permet de construire des multi-modèles AA4MM conformes au méta-modèle, mais seulement avec les modèles déjà intégrés dans MECSYCO. L'utilisation et surtout l'extensibilité de l'approche s'en trouvent limitées.

Manque d'utilisabilité

L'un des objectifs de MECSYCO est de permettre à des experts domaines de travailler ensemble sur la modélisation de système complexe en connectant leurs logiciels de M&S. Or pour l'instant, il est nécessaire d'avoir des connaissance en programmation pour utiliser la plateforme ce qui empêche son utilisation par des non-informaticiens.

5.3 Proposition

Notre contribution s'inscrit sur deux axes : **la définition d'une démarche de multi-modélisation et co-simulation autorisant la création hiérarchique de multi-modèles**, et **la création d'un environnement de développement permettant sa mise en pratique sur MECSYCO**.

Par le terme "démarche", nous entendons identifier les différentes étapes permettant de construire incrémentalement un multi-modèle à partir de composants hétérogènes et de mener des expériences sur ce multi-modèle. Notre travail sur MECSYCO nous a amené à identifier cinq niveaux dans l'activité de multi-modélisation et co-simulation :

1. **La conception** qui consiste à identifier le système cible global et à définir un multi-modèle abstrait ainsi que les composants à modéliser.
2. **La modélisation atomique** qui concerne le développement des modèles atomiques, hétérogènes ou non, devant être utilisés pour construire les multi-modèles. Dans le cas de MECSYCO, cette étape doit s'effectuer dans des logiciels de M&S tiers. Notons que ce niveau regroupe l'ensemble des étapes de l'activité de M&S que nous avons vues Figure 2.2 au chapitre 2.
3. **L'intégration** qui concerne l'encapsulation logicielle et formelle des modèles atomiques ainsi que la définition de leurs interfaces. Cette étape doit aboutir à la création de modèles composants réutilisables.
4. **La multi-modélisation** qui consiste à coupler les différents modèles atomiques précédents pour en faire des multi-modèles. C'est à cette étape que nous formalisons la notion de multi-modèle MECSYCO en nous inspirant de DEVS pour assurer la propriété de fermeture par couplage. Cette propriété nous autorise la réutilisation des multi-modèles et donc la construction hiérarchique. Cette étape doit permettre de mettre en relation des modèles composants et d'en produire de nouveaux à partir de ceux existants.
5. **La co-simulation**, étape durant laquelle nous définissons les expériences que nous voulons réaliser avec nos modèles/multi-modèles composants choisis dans la bibliothèque de modèles que nous avons construite aux niveaux précédents.

Ces cinq niveaux ne sont pas spécifiques à MECSYCO mais sont généralisables à toute approche par co-simulation.

L'identification des trois niveaux effectués dans MECSYCO (l'intégration, la multi-modélisation et l'expérimentation) se concrétise par l'utilisation de documents de description spécifiques à chacun d'eux. Dit autrement, **nous choisissons de mettre en place une démarche descriptive où chaque élément (modèle, multi-modèle, co-simulation) produit lors des activités précédentes fait l'objet d'une description qui permet de le manipuler, de le réutiliser et de l'analyser**.

Pour la mise en pratique de notre démarche, nous y associons un environnement de développement basé sur l'utilisation de langages dédiés. Chaque niveau de l'activité de M&S est supporté par un langage qui facilite la création des documents de description. Cet environnement peut être vu comme une surcouche autour de MECSYCO qui facilite son utilisation. Nous développons ensuite les mécanismes qui permettent d'automatiser le lancement d'une co-simulation à partir de sa description, limitant ainsi les erreurs d'implémentation. L'environnement associé aux mécanismes de transformation s'inscrit dans une démarche d'Ingénierie Dirigée par les Modèles. Pour ne pas retomber dans le travers du manque d'extensibilité de l'environnement précédent, nous ajoutons des spécifications et un moyen pour intégrer facilement de nouveaux composants (artéfacts ou opérations). L'**extensibilité** sera donc l'un de nos critères d'évaluation de nos contributions au même titre que la **modularité**, la **capacité à hiérarchiser** et la **facilité d'utilisation**.

La démarche descriptive que nous proposons nous permet de construire des multi-modèles de manière hiérarchique et modulaire tandis que l'environnement se charge de faciliter les descriptions et d'automatiser le passage vers la co-simulation en utilisant le cœur MECSYCO.

Nous détaillons la démarche et l'environnement associé dans le prochain chapitre. Le chapitre 7 est consacré à plusieurs apports plus localisés sur différents aspects de l'activité de multi-modélisation et co-simulation et qui ont généré des développements dans MECSYCO. Enfin, le chapitre 8 présentera deux exemples pour illustrer et évaluer les contributions.

6

Démarche de multi-modélisation hiérarchique et co-simulation

Ce chapitre présente comment, à partir de notre expérience sur MECSYCO, nous avons travaillé sur la démarche de multi-modélisation et co-simulation basée sur l'intégration de modèles hétérogènes et comment nous y avons incorporé la réutilisation des multi-modèles.

Le chapitre est organisé comme suit. Nous détaillons tout d'abord les spécifications des entités produites à chaque étape de l'activité de multi-modélisation et co-simulation. Cela nous donne les moyens de définir des fichiers de description qui permettent de manipuler les dites entités, de les réutiliser et de les analyser. Nous présentons ensuite un environnement destiné à soutenir cette démarche par l'utilisation de langages dédiés.

6.1 Modélisation par composition dans MECSYCO

Rappelons que l'objectif de la thèse est de proposer une démarche de multi-modélisation et co-simulation basée sur l'intégration de modèles hétérogènes, tout en permettant une construction hiérarchique. Ce travail s'inscrit dans la poursuite des travaux autour de l'intergiciel MECSYCO qui fournit déjà un cadre pour l'intégration de modèles et leur co-simulation.

Nous détaillons par la suite la démarche de multi-modélisation et co-simulation que nous proposons, celle-ci est basée sur la nécessité de documentation des modèles et multi-modèles. Une fonctionnalité supplémentaire sera développée dans l'intergiciel pour permettre la démarche.

6.1.1 La nécessité de documentation

La documentation des modèles est un aspect très présent dans la littérature :

— Dans FMI, chaque FMU est associée à un document "modelDescription.xml" qui présente l'interface du modèle. Ce document est nécessaire à l'import des FMU.

— Dans Into-CPS, en plus de l'utilisation des descriptions du standard FMI, les multi-modèles sont décrits via SysML. Cette description est utilisée ensuite comme squelette pour créer le multi-modèle qui sera co-simulé.

— Dans HLA, la documentation est structurée via le *Object Model Template* et se présente sous la forme de *Federation Object Models* qui décrivent les fédérations et de *Simulation Object Models* qui décrivent les fédérés.

— Dans VLE, des structures de modèles couplés servent de descriptions aux multi-modèles et sont stockées sous forme de XML [Quesnel, 2006]. Les documents servent ensuite à

manipuler les modèles couplés.

Les structures descriptives citées ci-dessus sont concrétisées par des fichiers qui peuvent ensuite être manipulés, analysés, partagés. L'idée générale que nous voulons mettre en avant par ces exemples est que, pour permettre la réutilisation d'un élément dans différents contextes, il est nécessaire de séparer sa description de son utilisation, i.e. d'associer un document de description à son implémentation. Dit autrement, **il faut se donner les moyens d'exprimer ce que nous souhaitons manipuler**. Cela nous amène à nous demander : que décrire ? comment ? et sous quelle forme ?

Dans notre cas basé sur l'intégration de modèles hétérogènes, nous avons identifié 5 étapes successives pour l'activité de multi-modélisation et co-simulation : la conception du multi-modèle, le développement des modèles atomiques, leur intégration, la construction des multi-modèles et enfin l'expérimentation. Après l'étape de conception, chacune des 4 étapes suivantes produit des éléments qui devront être manipulés et réutilisés. Ce sont respectivement :

1. Les modèles implémentés qui sont ceux produits dans des logiciels de M&S spécialisés que nous souhaitons réutiliser pour nos co-simulations.
2. Les modèles intégrés/atomiques sont ceux qui sont déjà utilisables dans la plateforme de co-simulation, ils doivent servir à construire des multi-modèles.
3. Les multi-modèles construits dans la plateforme qui doivent pouvoir être réutilisés pour construire hiérarchiquement le multi-modèle du système cible ou pour lancer des expériences.
4. Les plans d'expérience qui doivent être utilisés pour exécuter ou ré-exécuter les co-simulations.

Les modèles implémentés sont généralement des fichiers enregistrés dans des formats spécifiques à chaque plateforme. Les modèles intégrés, les multi-modèles et les expériences sont les entités à manipuler pour effectuer la multi-modélisation et la co-simulation. Ce sont les entités que nous devons décrire.

En résumé, pour apporter à MECSYCO la capacité à hiérarchiser ses multi-modèles, nous pensons qu'il est nécessaire de définir des documents de description adaptés à chacune des étapes que nous avons identifiées pour la multi-modélisation et co-simulation. Ce sont ces documents qui permettront ensuite de manipuler les entités décrites, les modèles et les multi-modèles, pour conduire la multi-modélisation jusqu'à l'expérimentation. La manipulation des différents éléments via des descriptions permet en outre de s'abstraire du code, un aspect important pour que l'intergiciel puisse être utilisé par des experts domaines non-informaticiens.

Nous définissons trois niveaux de description :

1. **Description des modèles atomiques** qui pourront être utilisés dans MECSYCO. Le terme "modèle atomique" fait référence ici à un modèle (issu d'un logiciel de M&S existant) qui peut être utilisé dans MECSYCO, i.e. pour lequel un artéfact de modèle est (ou va être) implémenté. Ce sont ces modèles atomiques qui serviront de base pour la construction des multi-modèles.
2. **Description des multi-modèles** qui pourront servir à lancer des expériences ou être réutilisés au sein d'autres multi-modèles. La description des multi-modèles se base formellement sur la description des modèles couplés DEVS, c'est ce qui nous assure de la propriété de fermeture par couplage, i.e. de la capacité à pouvoir réutiliser les multi-modèles et à construire hiérarchiquement.

3. **Description des expériences** (des co-simulations) qui contiendront toutes les informations nécessaires pour lancer une co-simulation à partir d'un multi-modèle MECSYCO, notamment les jeux de paramètres, les systèmes d'observation à utiliser, les informations à récolter (log)...

Nous définissons dans la suite les informations que nous retenons pour chaque étape. Notons que nos documents de description prennent la forme de fichiers XML.

6.1.2 Description d'un modèle atomique

Nous allons détailler dans cette partie les informations que nous proposons de stocker pour permettre la réutilisation d'un modèle atomique de MECSYCO. Comme précisé en 5.2.4, l'intégration des modèles s'effectue via un pont logiciel et un pont formel (concrétisés par un m-artéfact dans MECSYCO), mais il faut aussi décrire les interfaces des modèles (un des manques identifiés dans la plateforme) .

Structure générale

La Figure 6.1 présente la structure générale utilisée pour la description d'un modèle atomique.

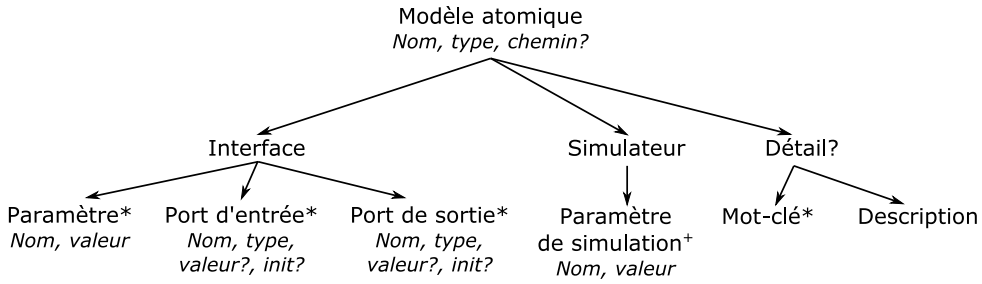


FIGURE 6.1 – Structure générale de description d'un modèle.

Nous représentons la structure générale sous forme d'arbre, un format qui représente bien la hiérarchie des éléments. Chaque nœud représente un concept et nous notons en dessous, en italique, les éléments qui le caractérisent. Le nœud racine, le concept de modèle, est donc caractérisé par :

— Un **nom** qui sera utilisé pour identifier le modèle
— Le **type** du modèle qui permet de choisir automatiquement une stratégie pour l'incorporer à la co-simulation. Le nombre de types différents acceptés sera amené à augmenter au fur et à mesure que des simulateurs sont intégrés à MECSYCO. À l'heure actuelle, MECSYCO dispose d'artéfacts de modèle pour les FMU 2.0 de type co-simulation, pour NetLogo (nous y reviendrons en 7.1) et pour les fichier CSV, les types acceptés correspondants sont "FMU CS 2", "NetLogo" et "CSV".
— Le **chemin** vers le modèle implémenté (s'il y a lieu).

Ensuite les sous-nœuds représentent les sous-concepts. Par exemple, pour décrire un modèle nous devons décrire son interface (paramètres et ports) et son simulateur, nous avons donc deux sous-concepts **Interface** et **Simulateur**. Le symbole " ?" indique qu'un élément n'est pas obligatoire, c'est le cas du sous-concept **Détail** qui permet d'attacher des renseignements supplémentaires sur le modèle. Enfin, le symbole "*" désigne un élément présent 0 ou N fois, ainsi l'interface d'un modèle peut contenir plusieurs paramètres ou pas du tout. De façon similaire,

"+" indique qu'un élément est répété (mais au moins une fois). C'est le cas des paramètres de simulation, nous imposons que le temps de fin de simulation soit toujours présent.

La suite détaille les trois sous-concepts **Interface**, **Simulateur** et **Détail**.

L'interface du modèle

La description de l'interface permet de répondre à trois question : Comment initialiser le modèle ? Comment envoyer des perturbations au modèle ? Comment récupérer des informations du modèle ? Elle comprend donc :

- Les **paramètres** du modèle associés à leur valeur par défaut (le type de donnée est inféré/déduit grâce à cette valeur).
- Les **ports d'entrée** du modèle ainsi que leur type, qui permettent de savoir comment connecter le modèle en entrée. Une valeur par défaut et une information sur l'initialisation sont aussi ajoutées mais sont optionnelles (détail plus loin).
- Les **ports de sortie** du modèle ainsi que leur type, qui permettent de savoir comment connecter le modèle en sortie. Une valeur par défaut et une information sur l'initialisation sont aussi ajoutées mais sont optionnelles (détail plus loin).

À ce niveau, il est important de faire un récapitulatif du lien entre la spécification du modèle atomique DEVS et le modèle atomique MECSYCO. Rappelons la structure d'un modèle atomique DEVS :

$$M_i = (X_i, Y_i, S, \delta_{ext}, \delta_{in}, \lambda, ta)$$

Les liens avec le modèles atomiques MECSYCO sont dispersés sur trois niveaux :

1. Concernant l'ensemble S des états du modèle, comme nous passons par une stratégie d'encapsulation logicielle et formelle, nous n'y avons pas directement accès. En effet, ceux-ci sont (ou non) spécifiés dans le modèle implémenté que nous intégrons.
2. La partie dynamique (δ_{in}, δ_{ext}, λ et ta) est contenue dans l'artéfact de modèle qui fait le lien entre le simulateur du modèle et le simulateur abstrait DEVS.
3. L'interface (les ensembles X et Y des ports d'entrée et de sortie) est décrite via notre description.

Un point intéressant à relever est que contrairement à notre description, la structure DEVS ne contient pas de paramètres. En effet, le formalisme DEVS ne définit pas comment un modèle peut être paramétré ni comment l'initialiser. C'est une lacune déjà identifiée dans [Van Tendeloo and Vangheluwe, 2018], où les auteurs proposent d'ajouter à la spécification DEVS un ensemble $q_{init} \in Q$ correspondant à l'état total dans lequel le modèle se trouve au début de la simulation. Cela donne l'ensemble :

$$M_i = (X_i, Y_i, S, \mathbf{q_{init}}, \delta_{ext}, \delta_{in}, \lambda, ta)$$

Dans MECSYCO, nous voyons les modèles comme des boites noires, sans avoir accès à leur état interne. La définition précédente de l'initialisation ne peut donc pas nous convenir.

La plupart des simulateurs proposent des moyens de paramétrer les modèles. Ces paramètres sont liés à l'état interne du modèle mais peuvent aussi conditionner son comportement. Si nous prenons l'exemple d'un modèle de feu de forêt, nous pouvons avoir comme paramètre la localisation du début d'incendie (liée à l'état interne) et la vitesse de propagation du feu (liée au comportement).

Le choix opéré dans MECSYCO (et que nous avons formalisé dans notre description) est plus pragmatique. Nous considérons les paramètres comme des ports d'entrée accessibles seulement

à l'appel de la méthode *init*() du simulateur. Nous voyons donc nos modèles atomiques comme des ensembles :

$$M_i = (X_i, Y_i, \mathbf{P_i}, S, \delta_{ext}, \delta_{in}, \lambda, ta)$$

Avec $P_i = \{(p, v) | p \in Param_i, v \in V_{P_i}\}$ l'ensemble des paramètres ($Param_i$) et de leurs valeurs admissibles (V_{P_i}).

Un autre aspect important à communiquer pour coupler un modèle concerne les valeurs initiales de ses ports. Ces informations seront nécessaires lorsque nous couplerons plusieurs modèles pour s'assurer par exemple que la valeur initiale d'un port de sortie d'un modèle est cohérente avec celle du port d'entrée auquel il est connecté. Si ce n'est pas le cas, il est important de savoir si les valeurs des ports peuvent être initialisées. Si c'est le cas, les valeurs initiales des ports peuvent être considérées comme des paramètres du modèles. Nous enrichissons donc la description des ports (voir Figure 6.1) avec une valeur par défaut *valeur* et un attribut *init* précisant si le port peut être initialisé comme un paramètre ou non.

Un dernier point important à éclaircir concerne la notion de *type* des ports d'entrée et de sortie, aspect qui n'est pas détaillé sur la Figure 6.1 pour ne pas alourdir le schéma. Le *type* caractérise la structure de donnée adaptée au port, il est en lien avec le niveau 2 d'*interopérabilité*, l'*interopérabilité* syntaxique, de l'échelle LCIM (voir Figure 3.2 en section 3.3). MECSYCO propose plusieurs types, qui peuvent en plus être composés. Pour éviter des structures trop complexes sans perdre en expressivité, nous choisissons de nous restreindre à :

— Des types simples (entiers, flottants, booléens, chaines de caractères).
— Des types complexes (des tuples de 1 à 5 éléments de types simples distincts, et des listes de tuples). Notons que les tuples de 1 élément correspondent en réalité à des types simples.

Cette restriction a notamment un impact lors de l'intégration de nouveaux simulateurs en limitant les choix du programmeur sur les évènements qui pourront sortir ou être intégrés.

Le simulateur

La branche **Simulateur** permet de distinguer les paramètres du modèle, qui impactent son comportement intrinsèque, et les paramètres de simulation et de co-simulation, qui impactent la manière dont ce comportement est déroulé.

Les paramètres de la co-simulation, qui dépendent de l'algorithme utilisé dans MECSYCO, sont définis au même endroit que les paramètres du simulateur du modèle. La branche *simulateur* contient donc seulement les noms des *paramètres de simulation* du modèle associés à leur valeur par défaut.

Notes :

— L'algorithme CMB utilisé dans MECSYCO dispose de 3 paramètres :

1. le temps de fin de simulation
2. le temps de début de simulation
3. le temps minimum de propagation des évènements

Du fait de sa nature décentralisée, chacun de ces paramètres est particulier à chaque modèle. Nous imposons donc des noms de paramètre de simulation particuliers pour identifier les paramètres liés à CMB pour chaque modèle. Nous imposons respectivement les noms "stopTime", "startTime" et "minPropagationDelay".

Documentation du modèle

En plus de l'interface du modèle et de son simulateur, nous avons décidé d'ajouter une branche **Détail** optionnelle pour permettre l'ajout d'une documentation informelle du modèle. Cette documentation est basée sur deux éléments :

1. Un ensemble de mots-clés pour identifier le modèle et son domaine. L'idée sous-jacente est que nos modèles, une fois intégrés dans MECSYCO et documentés, sont des composants modulaires qui forment une bibliothèque de composants. L'ajout de méta-données telles que des mots-clés permet d'effectuer des recherches dans la bibliothèque de modèles. Cette possibilité de découverte des modèles est l'une des recommandations de Katherine L. Morse dans [Taylor et al., 2013, Morse, 2004] pour le développement de framework de composition de modèle.
2. Un texte à remplir par le modélisateur pour expliquer son modèle. Nous avons fait le choix de n'imposer aucune contrainte sur cette description qui a pour rôle de faciliter la compréhension du modèle pour un modélisateur devant le réutiliser.

Cette documentation informelle est une ébauche qui doit être améliorée. Une spécification plus poussée nous permettrait de faire effectivement de la découverte de modèle. Nous pourrions aussi, en ajoutant des informations plus formalisées, calculer des propriétés sur nos modèles et automatiser des solutions (par exemple, spécifier la représentation du temps dans les modèles pour automatiser l'utilisation d'opérations temporelles lorsque nécessaire).

6.1.3 Description d'un modèle couplé

Nous allons décrire dans cette section les informations que nous devons stocker pour permettre 1) la co-simulation du multi-modèle et 2) sa réutilisation comme composant au sein d'un autre multi-modèle MECSYCO. Pour répondre à ces deux objectifs la structure que nous proposons se base sur le formalisme DEVS car celui-ci fournit une structure de modèle couplé fermée par couplage. Nous y intégrons ensuite les éléments du méta-modèle AA4MM qui fournit des mécanismes de gestion des hétérogénéités.

Structure générale

La Figure 6.2 présente la structuration de l'information décrivant un modèle couplé.

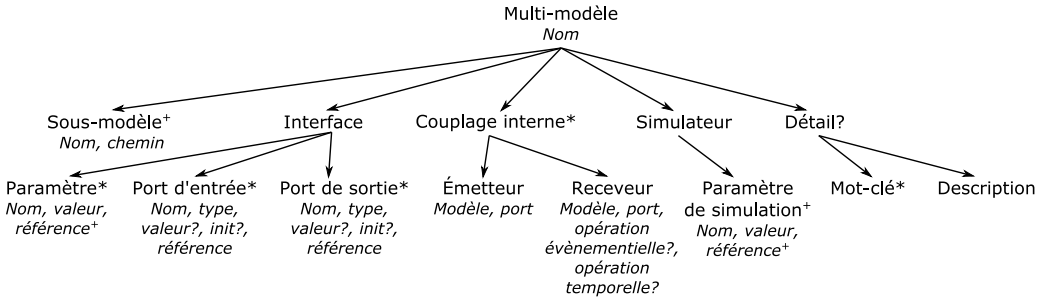


FIGURE 6.2 – Structure générale de description d'un multi-modèle

Les multi-modèles sont des modèles, donc nous retrouvons les concepts d'**Interface**, de **Simulateur** et de **Détail**. La branche **Détail** est d'ailleurs identique à celle du modèle.

Comme les modèles, nous identifions les multi-modèles par un nom. Leur structure est ensuite inspirée de la spécification du modèle couplée DEVS que nous rappelons :

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

Les multi-modèles sont composés d'un ensemble de modèles qui interagissent, c'est le concept de **Sous-Modèle**, il y en a au minimum un. Cela correspond à l'ensemble $\{M_d | d \in D\}$. Chaque sous-modèle est identifié par son nom (l'ensemble des noms forme l'ensemble D) et par le chemin menant à sa description (que ce soit un modèle atomique ou un multi-modèle, ce qui permet les hiérarchies).

Nous détaillons ensuite la spécificité de l'interface du multi-modèle et de son simulateur, puis son architecture interne (les couplages entre ses sous-modèles).

L'interface et le simulateur

Comme pour DEVS, nous retrouvons dans l'interface du multi-modèle : les ports d'entrée (X), les ports de sortie (Y), ainsi que les paramètres (P) que nous avons ajoutés .

Le multi-modèle MECYCO étant structurel, chaque port d'entrée, de sortie et chaque paramètre du multi-modèle fait référence respectivement à un port d'entrée, de sortie ou à un paramètre d'un sous-modèle. Pour chacun des ports et paramètres nous ajoutons un attribut *référence* qui fait le lien avec les sous-modèles. Ces références sont en fait les ensembles EIC (*External Input Couplings*) et EOC (*External Input Couplings*) de DEVS, auxquels nous ajoutons l'ensemble EPC (*External Parameter Couplings*) pour les paramètres.

Il est important de noter que contrairement aux ports, nous considérons qu'un paramètre de multi-modèle peut faire référence à plusieurs paramètres de sous-modèles. Cela permet de simplifier le paramétrage lorsqu'un même paramètre s'applique sur plusieurs modèles.

Pour la branche **Simulateur** qui contient l'ensemble les paramètres pour les simulateurs des sous-modèles et pour l'algorithme de co-simulation. Comme l'algorithme CMB n'impose pas de paramètre globaux, tous les paramètres de simulation du multi-modèle font référence à des paramètres des sous-modèles.

Donc de la même manière que pour les paramètres de modèle, chaque paramètre de simulation du multi-modèle peut faire référence à plusieurs paramètres de simulation des sous-modèles (c'est souvent le cas pour le paramètre de temps de fin de simulation).

Couplages internes

Les couplages internes correspondent à l'ensemble IC du modèle couplé DEVS. Chaque couplage est une relation entre deux modèles, i.e. c'est la connexion entre un port de sortie d'un modèle m_i vers un port d'entrée d'un modèle m_j.

Les couplages sont représentés par les noms du modèle émetteur et de son port et les noms du modèle receveur et de son port. Cependant, dans MECSYCO, les couplages peuvent être liés à des opérations qui permettent de corriger les hétérogénéités de représentation lors de la co-simulation de modèles d'origines différentes. C'est le but des éléments **opération évènementielle** et **opération temporelle** associés au receveur.

Précision sur les opérations

Dans [Camus et al., 2018], deux types d'opération sont distingués : les opérations évènementielles s'appliquant seulement sur les échanges de données (les évènements externes) et les

opérations temporelles s'appliquant sur toutes les communications (dont celles liées à la synchronisation). Les opérations sont effectuées par les artéfacts de couplage, chaque artéfact de couplage contient un ensemble ordonné d'opérations évènementielles et un ensemble ordonné d'opérations temporelles.

Les opérations évènementielles ont deux objectifs :

1. les changements d'échelle, par exemple pour passer de km à m, et
2. les changements de niveau de résolution, e.g. passer d'une vision individuelle (une voiture) à une vision agrégée (un flux de voitures).

Pour faire le lien avec la formalisation DEVS et les ensembles d'évènements admissibles, une opération évènementielle entre le port de sortie *out* d'un modèle m_i et le port d'entrée *in* d'un modèle m_j peut être représentée par la fonction :

$$O^{data}_{m_i^{out}->m_j^{in}} : V_{m_i^{out}} -> V_{m_j^{in}}$$

Avec :
— $V_{m_i^{out}}$ l'ensemble des valeurs admissibles du port de sortie *out* du modèle m_i.
— $V_{m_j^{in}}$ l'ensemble des valeurs admissibles du port d'entrée *in* du modèle m_j.

Étant donné le rôle des opérations, celles-ci sont vouées à être différentes pour chaque connexion entre modèles aux représentations hétérogènes. Sans perdre de généralité, nous choisissons de nous restreindre à une opération évènementielle par couplage interne.

Chaque opération est un morceau de code exécutable qui sera appelé durant la co-simulation. Pour éviter d'avoir à implémenter une opération pour chaque cas, nous choisissons de les rendre paramétrables. Cela permettra d'utiliser des opérations standards pour les cas les plus communs (par exemple une multiplication avec un paramètre réel pour changer d'échelle). Dans les autres cas, une opération particulière devra être implémentée.

Les opérations évènementielles étant des objets implémentés (typiquement en Java), nous les décrivons au sein d'un couplage interne comme :
— un chemin menant à leur implémentation, et
— des paramètres, i.e. des ensembles (identifiant, valeur de type simple).

De leur côté, les opérations temporelles visent à corriger les problèmes d'échelles temporelles (par exemple en convertissant des secondes en nanosecondes, des minutes en heures, ...). Étant donné que le temps est considéré comme un réel dans MECSYCO, une opération temporelle entre le port de sortie *out* d'un modèle m_i et le port d'entrée *in* d'un modèle m_j peut être représentée par une fonction :

$$O^{time}_{m_i^{out}->m_j^{in}} : \mathbb{R} -> \mathbb{R}$$

Dans le cas des opérations temporelles, nous choisissons de nous restreindre aux opérations d'addition (qui servent aussi pour les soustractions), de multiplication et de division associées à un paramètre réel. Ces opérations étant implémentées, nous les décrivons via un type (*add*, *multiply* ou *divide*) et une valeur réelle correspondant à l'opérande.

Note : Nous aurions pu, pour prendre en compte les opérations, nous donner les moyens d'exprimer les calculs nécessaires directement dans nos langages. Cela reviendrait à définir un langage de programmation dans nos langages de description. Par simplicité, nous nous contentons d'opérations sous forme de morceaux de code exécutable à appeler.

6.1.4 Description d'une co-simulation

Avec les deux sections précédentes, nous sommes capables d'exprimer un multi-modèle composé de sous-modèles (atomiques ou couplés) qui peut lui-même servir de composant pour former d'autres multi-modèles. Ces deux premiers types de description nous permettent de décrire les différents éléments que nous construisons au fur et à mesure de l'étape de modélisation. Par ailleurs, comme l'objectif est l'intégration de simulateurs hétérogènes, les descriptions contiennent aussi les paramètres spécifiques de chaque simulateur, et les paramètres liés à l'algorithme de co-simulation. Cet ensemble d'informations nous permet de manipuler les modèles lors de l'étape de modélisation mais aussi de lancer leurs simulations avec les paramètres par défaut. Cependant, un même multi-modèle est presque toujours utilisé dans plusieurs expériences pour tester différents paramètres et scénarios. Il est donc pertinent de documenter de manière séparée les multi-modèles et les expériences effectuées sur ces multi-modèles.

C'est l'objectif de cette dernière description qui doit exprimer ce qu'est une expérience MECSYCO. Cette description doit contenir les informations nécessaires pour exécuter ou ré-exécuter automatiquement une expérience/co-simulation MECSYCO.

Structure générale

Pour réaliser une expérience virtuelle il nous faut :
— un modèle à simuler,
— un jeu de paramètres pour ce modèle, et enfin
— une définition de ce qui doit être observé, des informations à récolter.

Nous retrouvons ces éléments sur la Figure 6.3 qui présente la structure générale d'une expérience MECSYCO.

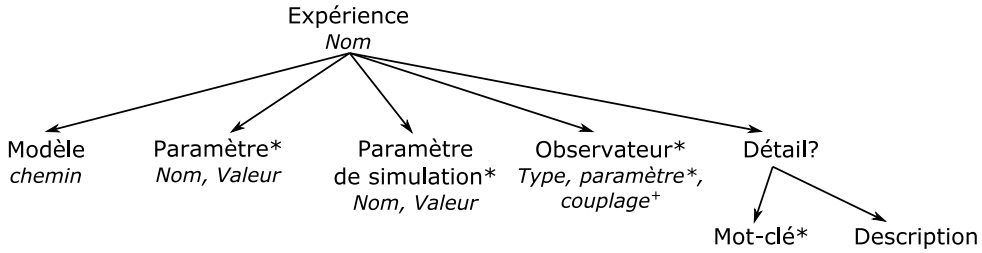


FIGURE 6.3 – Structure générale de description d'une expérience

Dans cette structure, nous identifions une expérience via un nom. La structure comprend ensuite un modèle/multi-modèle à simuler que nous récupérons via le chemin vers sa description, le choix des paramètres du modèle et des paramètres de simulation, et enfin un choix des mécanismes d'observation à utiliser. La branche optionnelle **Détail** est identique à celle des modèles atomiques et couplés.

Précision sur l'observation

Les branches **Modèle**, **Paramètres** et **Paramètres de simulation** rassemblent les informations pour pouvoir lancer une co-simulation (le choix du modèle et des paramètres), la branche **Observateurs** gère le dernier aspect essentiel pour l'expérience : définir ce qui doit être observé.

À l'heure actuelle, dans MECSYCO, l'observation est basée sur des m-agents associés à des artéfacts d'observation similaires aux artéfacts de modèle. Cela signifie que les observateurs sont

vus comme des modèles DEVS particuliers qui ne font que recevoir les évènements issus des autres modèles pour les afficher (ou les enregistrer). *Ces modèles d'observation n'ont donc pas de ports de sortie et peuvent avoir un nombre de ports d'entrée variable.*

Cela entraine que décrire un observateur dans MECSYCO revient à décrire d'une part le type de modèle d'observation choisi et d'autre part les interactions avec ce modèle d'observation.

Chaque observateur est dont identifié par :

— Son *type*, qui correspond soit à un mécanisme d'observation "classique" de MECSYCO ("TX" ou "3D" donnant respectivement des graphes 2D et 3D) soit à un chemin vers une implémentation d'un nouvel artéfact d'observation.
— Ses *paramètres* s'il en a.
— Ses *couplages* qui sont les liens avec les ports de sortie du modèle. Lors de la co-simulation, ces liens seront représentés par des artéfacts de couplage. Chaque couplage est représenté par un nom de port (utilisé côté graphique), le nom du port de sortie du modèle (celui qui enverra les évènements à observer) puis si besoin une opération évènementielle et/ou une opération temporelle.

6.1.5 Des descriptions à l'exécution

Les sections précédentes nous ont permis de détailler les structures de description que nous avons définies pour MECSYCO. Les informations sont stockées au format XML, il faut donc les lire et les traiter pour passer des descriptions aux co-simulations effectives via MECSYCO. Le but de la partie suivante est de montrer comment sont effectuées ces transformations et en quoi notre approche est extensible.

Avant les expériences MECSYCO étaient lancées à partir d'un code Java. Dorénavant, nous passons automatiquement d'une description de ces expériences à leur exécution.

Nous présentons dans la suite l'architecture des classes Java implémentées pour lire les fichiers XML et effectuer les traitements, puis nous détaillons comment l'extensibilité de notre approche est apportée.

Architecture générale

La Figure 6.4 présente un schéma UML reprenant l'architecture générale des classes utilisées pour manipuler les fichiers XML.

Nous avons choisi d'implémenter des classes Java pour chaque type de description. Nous en avons donc une pour les co-simulations, une pour les modèles couplés et une pour chaque type de modèle atomique.

Pour rassembler les traitements identiques, les informations communes aux modèles atomiques et couplés sont chargées dans une seule et même classe *ModelDescription*. Ensuite la distinction est faite entre les modèles atomiques (*AtomicModelDescription*) et les modèles couplés (*MultimodelDescription*). La première ajoute la méthode *getModelArtifact* renvoyant l'artéfact de modèle adapté en fonction des informations chargées, tandis que la seconde implémente *getGenericMMAgent* qui renvoie un *GenericMMAgent*, i.e. un agent de niveau supérieur contenant un ensemble de sous-agents devant prendre part à la co-simulation.

La classe *AtomicModelDescription* est abstraite, toutes ses classes filles sont adaptées à un artéfact de modèle particulier et implémentent la méthode *getModelArtifact* adéquate. Les spécificités de chaque simulateur peuvent être capturées via la méthode *importSpecificData*. Cette méthode est utile lorsqu'il est nécessaire de stocker plus d'informations pour décrire un modèle

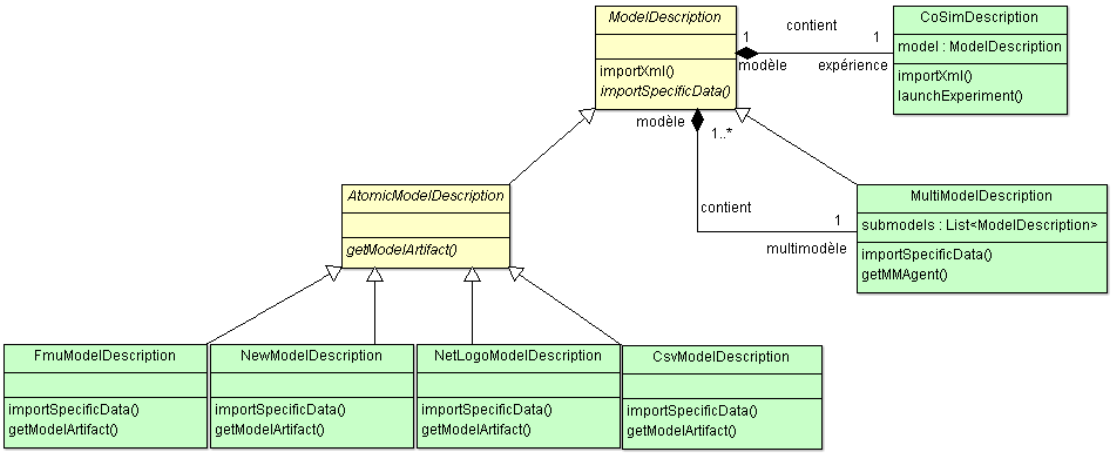


FIGURE 6.4 – Schéma résumé de l'organisation des classes chargées de lire les descriptions XML et de les manipuler. (Classes abstraites en jaune, classes concrètes en vert)

atomique issu d'un simulateur particulier. Elle est notamment utile dans *NetLogoModelDescription* pour les descriptions des modèles NetLogo (voir section 7.1) où des commandes doivent être associées aux ports et aux paramètres des modèles.

Pour l'instant, il y a des classes spécifiques pour chaque type de modèle atomique associé à des artéfacts de modèle existants : *FmuModelDescription* pour le type "FMU CS 2", *CsvModelDescription* pour le type "CSV" et *NetLogoModelDescription* pour le type "NetLogo". La classe *NewModelDescription* est une classe particulière adaptée à plusieurs types de modèles atomiques du moment que les artéfacts de modèle correspondants respectent certaines propriétés. Cela nous amène à l'extensibilité de notre approche et à l'intégration de nouveaux artéfacts de modèle.

Prise en compte des futurs artéfacts de modèle et opérations

Lors de l'activité de multi-modélisation et co-simulation, trois étapes majeures sont celles d'intégration de nouveaux simulateurs, de création d'opérations pour gérer les représentations hétérogènes et de développement de systèmes d'observation. Cela signifie que de nouveaux artéfacts de modèle, de nouvelles opérations et de nouveaux artéfacts d'observation sont amenés à être développés.

Pour pouvoir les intégrer facilement à l'environnement et au langage, nous avons choisi une stratégie de chargement à chaud (ou chargement dynamique) de classes Java. Cela consiste à charger, pendant l'exécution du programme, une classe implémentée dont nous connaissons l'interface, i.e. les signatures de ses constructeurs et de ses méthodes. Cette technique permet de reporter l'implémentation d'une classe dont nous connaissons l'interface, et de tester différentes implémentations sans avoir à recompiler l'ensemble du projet. Dans notre cas, cela nous offre la possibilité de charger de manière standard tout type d'artéfact de modèle. *Pour l'instant, cela ne marche qu'avec l'implémentation Java de MECSYCO, d'autres stratégies équivalentes devront être envisagées pour la version C++.*

Un utilisateur développant son propre artéfact de modèle/d'observation ou ses propres opérations d'évènement pourra les utiliser au sein de l'environnement, ceci en respectant des contraintes (héritage de la classe *GenericModelArtifact* de MECSYCO, signatures de constructeur particulières). Cela signifie que nous serons capable de charger, et d'utiliser, n'importe quel nouvel artéfact de modèle à partir de son fichier XML. L'attribut "*type*" de sa description aura alors

pour valeur le chemin vers la classe Java à charger.

C'est la classe *NewModelDescription* qui a pour fonction de récupérer les informations liées aux nouveaux artéfacts de modèle. Elle génère un *ModelArtifactLoader* dont le rôle est de charger dynamiquement les artéfacts de modèle nouvellement développés. Pour que cette stratégie fonctionne, nous n'autorisons que 5 types de constructeur pour les m-artéfacts :

— Un constructeurs à 6 arguments dont le nom et le chemin vers le modèle sous forme de chaines de caractères, deux listes de chaines de caractères pour les ports, et de deux dictionnaires pour les paramètres de modèles et les paramètres de simulation. Les clés sont les noms des paramètres sous forme de chaines de caractères, nous imposons que leur valeurs soient de type simple (entier, réel, booléen ou chaine de caractères). C'est le constructeur le plus complet (exemple ci-dessous), les autres sont des versions réduites.

```
MyModelArtifact(String aName, String aPath,
       List<String> inPorts, List<String> outPorts,
       Map<String, Object> modelParameters,
       Map<String, Object> simulParameters)
```

— Un constructeurs à deux arguments, le nom et le chemin vers le modèle.
— Un constructeur requérant un nom et un dictionnaire pour les paramètres (dans lequel sera fusionné les dictionnaires de paramètres de modèle et de simulation).
— Un constructeur avec seulement un nom.
— Le constructeur sans paramètre.

Cet aspect, bien que majoritairement technique, est intéressant car il permet d'augmenter l'extensibilité de l'environnement au prix d'une spécification plus poussée de l'implémentation des nouveaux artéfacts de modèle ou d'observation et des opérations sur les évènements.

Conservation de l'exécution décentralisée

Les descriptions de co-simulation au format XML sont directement utilisées pour lancer les expériences au travers de la méthode *launchExperiment* de la classe *CoSimDescription* (voir Figure 6.4).

Cette classe va charger la description de l'expérience et le modèle associé puis lancer la co-simulation. Il est important de noter que malgré une construction hiérarchique du multi-modèle, son exécution est toujours réalisée avec l'algorithme de Chandy-Misra-Bryant implémenté dans le coeur de MECSYCO [Camus, 2015]. Cela permet de conserver une exécution "à plat" et décentralisée.

6.1.6 Synthèse et discussion

La démarche de multi-modélisation et co-simulation proposée sépare l'activité en cinq étapes dont trois (de l'intégration à l'expérimentation) sont effectuées directement sur MECSYCO :

1. La conception du multi-modèle.
2. Le développement des modèles atomiques, réalisé dans des outils de M&S adaptés.
3. L'intégration, qui comprend le développement des artéfacts de modèle dans MECSYCO et la description des interfaces des modèles pour spécifier comment les utiliser.
4. La création du multi-modèle, en s'appuyant sur la structure du modèle DEVS couplé, ces descriptions peuvent servir de composants réutilisables au sein d'autres multi-modèles. C'est cette propriété qui nous permet de construire le multi-modèle final de manière hiérarchique.

5. L'expérimentation, i.e. la spécification des expériences qui sont réalisées sur le multi-modèle.

Un premier avantage de la démarche est d'avoir une distinction claire entre les différentes étapes et leur rôle. La contribution s'intéresse plus particulièrement aux étapes 3 à 5 que nous associons respectivement à trois niveaux de description : la description des modèles atomiques, celle des multi-modèles et celle des co-simulations. Les descriptions permettent d'augmenter la réutilisabilité des entités produites en exprimant de manière exploitable les éléments qu'elles contiennent. L'extensibilité de MECSYCO est aussi questionnée en prévoyant comment seront intégrés les futurs artéfacts de modèle (ou d'observation) et opérations sur les évènements.

La démarche que nous proposons se focalise sur l'intégration de modèles déjà implémentés pour les utiliser et construire incrémentalement des multi-modèles de systèmes complexes sur lesquels conduire des expériences. Contrairement à une méthode telle qu'ASPECS [Cossentino et al., 2010] qui permet de concevoir des modèles de systèmes complexes en suivant une approche orientée agent, nous n'abordons pas dans notre approche la phase de conceptualisation (nous partons du principe que cette phase a été réalisée au préalable). De plus, concernant les hiérarchies, notre démarche vise à permettre une construction incrémentale, i.e. nous avons une hiérarchie de construction du multi-modèle qui n'est pas forcément liée à une organisation identifiée dans les entités modélisées.

En résumé, la démarche présentée dans cette section a pour avantage d'être hiérarchique, d'augmenter la réutilisabilité des composants et d'être extensible tout en conservant les propriétés initiales de MECSYCO. En revanche, il est extrêmement fastidieux d'écrire nos descriptions directement en XML (ce qui serait valable pour n'importe quel format textuel). Le problème est que l'écriture d'une description dans un format spécifique sans éditeur dédié ni assistance (vérification de la syntaxe...) peut donner lieu à de nombreuses erreurs. Cela limite l'utilisabilité de l'outil. La section suivante présente une solution pour corriger ce défaut.

6.2 Environnement basé sur les DSL

L'objectif de cette partie est de présenter l'environnement de développement basé sur les DSL (*Domain Specific Language* ou langages dédiés) qui a été réalisé pour supporter la démarche de multi-modélisation et co-simulation décrite précédemment. Cet environnement s'inscrit dans une démarche d'Ingénierie Dirigée par les Modèles (IDM) dont le but est de faciliter l'utilisation de MECSYCO et l'écriture des descriptions, d'éviter les erreurs dans l'écriture et de permettre à des experts domaines non-informaticiens d'utiliser la plateforme.

Nous commencerons par décrire l'IDM, les DSL et leurs intérêts dans notre travail puis nous présenterons les langages que nous avons développés.

6.2.1 Ingénierie Dirigée par les Modèles et langages dédiés

L'Ingénierie Dirigée par les Modèles est une démarche d'ingénierie logicielle dont l'objectif est d'augmenter la vitesse de développement et la qualité des programmes [Sendall and Kozaczynski, 2003].

Son principe repose sur les deux étapes du développement d'un programme : la conception puis la programmation. Lors de la conception sont spécifiés les objectifs du programme et son organisation (par exemple à l'aide de diagramme UML). Un modèle du programme est créé (à différencier de la notion de modèle en M&S défini au chapitre 2). Ce modèle sert de guide au

programmeur lors de la deuxième étape qui consiste en la production du code. À la fin de l'étape de programmation, le modèle est ajouté à la documentation du logiciel.

Le problème est que des erreurs peuvent se glisser entre ces deux étapes, soit à cause d'un manque de spécification à l'étape de conception, soit à cause d'une erreur d'interprétation pendant la programmation.

L'idée de l'IDM est d'utiliser le modèle pour générer automatiquement le programme [Bézivin, 2004], évitant ainsi les erreurs d'interprétation et d'implémentation. Cela passe par la définition d'un méta-modèle (voir Définition 19) qui définit comment construire le modèle, et par le développement d'un processus de transformation qui permettra de passer du modèle au programme. En résumé, au lieu de coder directement les programmes, nous créons la transformation permettant de passer automatiquement des modèles aux codes.

Définition 19 (Méta-modèle)**.** Le méta-modèle d'un modèle M (ou d'un programme) est un modèle qui définit la syntaxe abstraite d'un langage utilisé pour décrire ce modèle M. [Voelter et al., 2013]

Le méta-modèle est une syntaxe abstraite permettant de spécifier un ensemble de modèles ; il définit les concepts, structures et notations qui permettront de décrire les modèles. Dans ce cadre là, **un modèle est dit valide s'il est conforme au méta-modèle**. Un processus de transformation doit alors être défini pour transformer automatiquement un modèle valide en un programme exécutable ou en un format textuel intermédiaire.

La démarche d'IDM peut être mise en place avec des langages graphiques (c'est ce qui avait été utilisé dans MECSYCO [Camus, 2015] avec le méta-modèle AA4MM [Siebert, 2011]), ou avec des langages textuels (l'option que nous choisissons) ou un mélange des deux (à l'image de ce qui est réalisé dans OpenModelica [Fritzson et al., 2005]). Le développement de langages dédiés fait donc partie d'une démarche d'Ingénierie Dirigée par les Modèles où le méta-modèle est la grammaire du langage, chaque script écrit dans le langage est un modèle, et chaque script subit un processus de transformation automatique pour effectuer une action ou générer un fichier dans un autre format. Cela nous amène à définir plus précisément la notion de DSL et leur développement.

Introduction aux DSL

Les DSL sont de petits langages adaptés à la résolution de problèmes précis (Définition 20).

Définition 20 (*Domain Specific Language*)**.** Un DSL est un langage de programmation ou un langage de spécification qui offre, grâce à des notations et abstractions appropriées, un pouvoir expressif concentré sur, et généralement limité à, un type de problème spécifique à un domaine. [Van Deursen et al., 2000]

Le langage HTML (*HyperText Markup Language*) est un exemple de DSL dédié à la représentation des pages web. Nous pouvons citer aussi Modelica [Fritzson and Bunus, 2002] dédié à la modélisation et simulation de systèmes équationnels. Les DSL sont généralement des langages déclaratifs qui peuvent être utilisés pour masquer les détails techniques liés à l'utilisation d'une bibliothèque logicielle ou d'un programme [Van Deursen et al., 2000].

Le terme "DSL" est souvent mis en opposition avec le terme "GPL" (*Global Programming Language*) qui désigne des langages tels C, Java et Python qui ne sont pas restreints à des problèmes particuliers. Il faut cependant garder à l'esprit que la différence entre un GPL et un DSL n'est pas clairement définie [Voelter et al., 2013]. L'objectif ici n'est pas de creuser cette

question. Nous nous contenterons d'une vision intuitive des DSL : **de petits langages définis pour répondre à un problème précis**.

Dans notre approche, les solutions à base de DSL ont de multiples avantages :

— **Niveau d'abstraction et expressivité** : Les DSL étant dédiés à des problèmes particuliers, les éléments du langage (mots-clés...) sont spécifiquement adaptés à la résolution de ce problème. En se focalisant sur un problème particulier ils fournissent le bon degré d'abstraction et sont expressifs.

— **Concision et facilité d'apprentissage** : Les DSL étant de petits langages, l'investissement pour les maîtriser est moindre.

— **Restrictions et vérifications** : La taille du DSL est limitée, en étant restreint à une certaine forme il limite le nombre d'erreurs (syntaxiques ou sémantiques) possibles et autorise la création de processus de vérification.

Il y a toutefois un inconvénient, le développement de langages dédiés est un processus difficile et coûteux en temps. Il y a néanmoins des outils spécialisés qui permettent de développer rapidement et relativement facilement des DSL. C'est le cas par exemple de XText [12], qui est l'outil que nous avons utilisé. Nous nous positionnons comme utilisateur d'outils dédiés au développement de DSL.

L'intérêt spécifique d'XText est qu'il propose nativement des fonctionnalités avancées pour l'environnement d'édition. Nous pouvons citer par exemple l'auto-complétion, la vérification et la coloration syntaxique etc. De plus, il autorise la génération de plugins Eclipse [13] permettant de fournir simplement un environnement de multi-modélisation et co-simulation. La simplicité d'utilisation de l'environnement ainsi généré a été empiriquement vérifiée en le faisant tester par des étudiants au cours de TP et de stages concernant la co-simulation.

Étapes du développement de DSL

Nous détaillons ci-dessous les différentes étapes de développement de DSL que nous avons suivies en partant des conseils présents dans [Bettini and Efftinge, 2016].

1. **Conception de la grammaire** : Cela correspond au développement du méta-modèle, i.e. à cette étape nous définissons ce que sera un script bien formé dans notre langage. En pratique, cela consiste à définir l'ensemble des concepts et règles qui feront notre langage et comment ils seront organisés. Dans notre cas, les DSL seront destinés à décrire respectivement un modèle, un multi-modèle et une co-simulation et permettront de générer les fichiers XML correspondants. Les concepts que nous définissons seront donc liés aux éléments que nous voulons décrire dans les fichiers de descriptions et qui ont déjà été présentés en 6.1.
2. **Développement de l'analyseur syntaxique** : Cet élément lit un script valide du langage pour le transformer en arbre syntaxique abstrait (AST pour *Abstract Syntax Tree*), un format plus adapté aux traitements ultérieurs car il permet de naviguer facilement entre les concepts utilisés dans le script. Dans XText, l'analyseur syntaxique est généré automatiquement à partir de la grammaire du langage. Nous ouvons faire un parallèle entre les arbres des Figures 6.1, 6.2 et 6.3 présentés précédemment et les AST que nous aurons dans nos langages dédiés.
3. **Développement d'un générateur** : Cette étape consiste à parcourir l'AST pour le transformer dans un autre format. Dans notre cas, nous le transformons en ficher XML

12. https://www.eclipse.org/Xtext/
13. https://www.eclipse.org

respectant l'une des structures définies précédemment.

4. **Déploiement** : Cela consiste à rendre les DSL utilisables dans un environnement. Dans notre cas, XText nous permet de générer simplement des plugins s'intégrant à l'environnement de développement Eclipse.

À la suite de ces étapes, d'autres éléments peuvent être développés pour faciliter l'utilisation du DSL : ajout de contraintes, gestion des erreurs, validation [14], proposition de corrections automatiques, indentation automatique... À l'heure où nous écrivons ces lignes, plusieurs éléments de gestions des erreurs et de validation ont été ajoutés aux langages pour faciliter leur utilisation. Ils seront détaillés lors des présentations des différents langages.

Proposer un ensemble de langages

Nous développons des DSL dédiés à chacun des niveaux de description que nous avons définis dans notre démarche. Ces DSL serviront d'interface utilisateur pour la création des documents de description XML définis plus tôt. Trois langages sont créés :

1. *m2xml* : Le langage dédié à la description de l'interface d'un modèle atomique.
2. *mm2xml* : Langage dédié à la description d'un multi-modèle, il reprend une partie des concepts définis dans *m2xml*, notamment pour décrire l'interface du multi-modèle. Avoir des concepts communs entre ces deux langages est naturel, nous utilisons en effet des éléments liés au formalismes DEVS qui permettent de manipuler un multi-modèle comme un modèle atomique lors de la phase de multi-modélisation. C'est ce qui autorise la hiérarchisation de la construction.
3. *cs2xml* : Ce dernier langage est dédié à la description des expériences. Il comprend plusieurs étapes allant de la sélection du modèle à simuler jusqu'au choix des dispositifs d'observation à mettre en place et des données à récolter.

Nous détaillons ci-dessous ces différents langages en donnant des éléments de leur syntaxe et des scripts d'exemple pour expliquer leur utilisation. Comme ces langages nous servent à exprimer les structures descriptives des Figures 6.1, 6.2 et 6.3 que nous avons déjà présentées, certains éléments sont redondants. Nous décrivons ensuite certains mécanismes de validation que nous avons développés pour guider les utilisateurs et les aider à vérifier leurs scripts. Ces mécanismes permettent d'avoir dans l'éditeur des vignettes d'erreurs, d'avertissement ou d'information directement sur les éléments concernés.

6.2.2 Le langage *m2xml* : description des modèles atomiques

Le langage *m2xml* est le premier DSL que nous proposons. Il fournit une syntaxe et des mots clés adaptés à la description de l'interface d'un modèle intégré dans MECSYCO via un m-artéfact. Le but est de fournir un média adapté qui guide les utilisateurs dans la description des modèles atomiques qu'ils veulent utiliser dans MECSYCO.

Éléments de syntaxe

La syntaxe de ce langage est le reflet de la structure de description d'un modèle atomique proposée Figure 6.1 et qui se retrouve dans les fichiers XML des modèles atomiques MECSYCO. les scripts *m2xml* sont transformés automatiquement en XML.

14. Le terme "validation" est à prendre ici dans le contexte du développement de langages dédiés et pas dans celui de la Théorie de la Modélisation et Simulation.

```
Modèle := Nom, TypeModèle, Chemin?, Interface, Information?, Simulateur;

Interface := Paramètre*, PortEntrée*, PortSortie*;

Paramètre := Nom, ValeurParDéfaut;
PortEntrée := Nom, PortType;
PortSortie := Nom, PortType;

Information := MotClé*, Description?;

Simulateur := ParamètreDeSimulation+;
ParamètreDeSimulation := Nom, ValeurParDéfaut;

//Type des ports
PortType := PortTypeSimple | PortTypeComplexe;
PortTypeSimple := TypeSimple, InitOption?, ValeurParDéfaut?;
PortTypeComplexe := Tuple | Vecteur;

// Terminaux
Nom := Identifiant;
TypeModèle := ChaineDeCaractères;
Chemin := ChaineDeCaractères;
ValeurParDéfaut := Entier | Booléen | Flottant | ChaineDeCaractères;
TypeSimple := "Int" | "Boolean" | "Real" | "String";
Tuple := (TypeSimple)1..5;
Vecteur := TypeSimple+;
MotClé := ChaineDeCaractères;
Description := ChainesDeCaractères;
```

FIGURE 6.5 – Syntaxe de *m2xml* en forme de Backus-Naur.

La Figure 6.5 présente la syntaxe du langage *m2xml* dans un format BNF (*Backus-Naur Form*). Ce format se rapproche fortement du code XText qui a été utilisé pour générer le langage, excepté que dans la version XText, des mots clés sont utilisés pour séparer les différents éléments. La Figure 6.6 présente un exemple contenant les différents éléments de syntaxe obtenus via XText, nous détaillons ces éléments ci-dessous.

Les premiers éléments demandés sont le nom du modèle sous forme d'identifiant, son type et son chemin (optionnel). Les identifiants sont des chaines de caractères respectant des règles de nommage, comme les noms de variables dans n'importe quel langage de programmation. Nous utilisons ces identifiants pour le nom du modèle mais aussi pour les noms des paramètres, des ports d'entrée et des ports de sortie.

Le type du modèle doit permettre de choisir l'artéfact de modèle à utiliser. Pour l'instant les types autorisés sont "CSV", "FMU CS 2" et les noms de classe Java du type "Mon.Package.MaClasseModelArtifact" (ces classes seront chargées à chaud lors du lancement des expériences pour les incorporer aux co-simulations). Le type "NetLogo" est traité dans un langage à part (voir section 7.1). Le chemin vers le modèle implémenté est un élément facultatif parce que tous les artéfacts de modèle n'en ont pas besoin.

Après ces premiers éléments, nous décrivons l'interface du modèle. Dans la version XText, les différents éléments sont identifiés par les mots-clés "*parameters*", "*inputs*" et "*outputs*". Chaque paramètre correspond à un nom suivi d'une valeur par défaut (entière, réelle, booléenne ou une chaine de caractères) tandis que les ports d'entrée/sortie correspondent à un nom suivi d'un

```
model MyModel type "CSV" path "./File.csv"
interface
  parameters
    Param1:13
    Param2:true
  inputs
    in:Double initOption no default value 10.
  outputs
    out:Integer
information
  keywords "Thesis" "Basic" "Example"
  description "Basic m2xml file."
simulator
  simulation variables
    stopTime:10.
```

FIGURE 6.6 – Exemple de description *m2xml*.

type simple ou complexe [15]. Seuls les types simples contiennent des éléments facultatifs comme la possibilité d'initialisation et/ou la valeur par défaut.

Les éléments de documentation du modèle, i.e. une documentation informelle et des mots-clés, sont respectivement introduits par les mots-clés *description* et *keywords*. Ce sont des chaines de caractères.

Enfin, nous plaçons en dernier les paramètres de simulation qui s'écrivent comme les autres paramètres. Toutefois, certains identifiants sont réservés pour les paramètres de l'algorithme de co-simulation de MECSYCO. Il s'agit des identifiants :

- *stopTime* qui correspond au temps de fin de simulation, ce paramètre est obligatoire et doit être réel.
- *startTime* qui correspond au temps de début de simulation, facultatif c'est un réel dont la valeur par défaut est 0.
- *minPropagationDelay* qui correspond au délai de réaction minimal d'un modèle lorsqu'il reçoit un évènement externe. Ce paramètre est lié à l'algorithme de synchronisation (Chandy-Misra-Bryant) utilisé dans MECSYCO [Camus, 2015]. Ce paramètre est facultatif. S'il n'est pas précisé, nous considérons que les évènements externes n'ont pas d'influence sur l'occurrence d'évènements dans le modèle.

Mécanismes de validation

L'utilisation du langage *m2xml* nous a amené à identifier plusieurs éléments de vérification importants à mettre en place pour guider les utilisateurs. Ces méthodes "*check*", définies dans la partie "*validation*" de XText , sont appelées directement lors de l'écriture des entités concernées dans le document. Cela signifie que des vignettes apparaissent dès que l'utilisateur finit d'écrire (dans l'éditeur dédié) un éléments concerné par une vérification.

Nous mettons en place plusieurs vignettes :

15. Les types des ports d'entrée et de sortie sont un ensemble restreint des types utilisables dans la plateforme MECSYCO. Nous nous limitons aux types simples (*Integer*, *Double*, *Boolean*, *String*) et à deux types complexes nommés respectivement *Tuple* et *Vector*. Les *Tuple* correspondent à des sacs de 1 à 5 éléments contenant des données hétérogènes tandis que le type *Vector* fait référence à une liste d'élément. Il est possible de composer les différents types (pour faire des *Vector* de *Vector* par exemple).

— Nous ajoutons une vignette d'avertissement lorsque les chemins définis ne mènent à aucun fichier. Cela permet d'éviter les erreurs de chargement.
— Plusieurs types d'artéfacts de modèle sont déjà définis. Pour identifier ceux nouvellement créés qui doivent être chargés à chaud, nous imposons que le nom de la classe Java se termine par "*ModelArtifact*". Nous explicitons ces informations par une vignette d'erreur apparaissant lorsque le type du modèle ne correspond à rien de reconnaissable et engendrerait une erreur au lancement de la co-simulation.
— Une vignette d'information est associée aux paramètres de simulation. En effet, comme nous l'avons précisé, plusieurs paramètres sont réservés pour l'algorithme de co-simulation de MECSYCO.
— Le paramètre "*stopTime*" étant obligatoire, nous générons une vignette d'erreur lorsqu'il n'est pas défini.

Par ce biais, les informations concernant ces paramètres de simulation particuliers sont directement affichées dans l'éditeur ce qui facilite l'apprentissage.

6.2.3 Le langage *mm2xml* : description des multi-modèles

La description des modèles couplés correspond à l'étape de multi-modélisation de notre démarche. Elle permet de mettre en relation différents modèles ou multi-modèles déjà décrits, de résoudre les problèmes d'hétérogénéités de représentation en précisant les opérations à appliquer sur les évènements et le temps, et d'obtenir une description XML de multi-modèle réutilisable au sein d'une autre description. La syntaxe de ce langage doit donc pouvoir exprimer cet ensemble de concepts.

Éléments de syntaxe

La syntaxe de description de l'interface du multi-modèle est similaire à celle présente dans *m2xml* au niveau de l'interface du modèle et des paramètres de simulation. Les points de divergence concernent les sous-modèles, leurs interconnexions et les liens avec l'interface du multi-modèle (les ensembles M, IC, EIC et EOC dans la définition du modèle DEVS couplé). À cela s'ajoute les opérations apportées par MECSYCO et AA4MM qui agissent sur le temps et les évènements.

La Figure 6.7 présente la syntaxe du langage *mm2xml* dans un format BNF. Comme pour les modèles, ce format se rapproche du code XText utilisé pour générer le langage (en retirant les mots clés). Un exemple de fichier *mm2xml* est donné Figure 6.8. Le script commence par le nom du multi-modèle. Viennent ensuite les noms des sous-modèles et les chemins qui mènent à leur description XML.

Nous demandons ensuite la description de l'interface du multi-modèle, similaire à celle d'un modèle excepté l'ajout de références (associées au mot-clé "*refers to*" dans XText), permettant de lier les paramètres et ports aux sous-modèles concernés. Un port du multi-modèle ne peut être connecté qu'à un port d'un sous-modèle, en revanche un paramètre de multi-modèle peut faire référence à plusieurs paramètres de différents sous-modèles. Cette connexion unique au niveau des ports découle directement du formalisme DEVS. Pour les paramètres (qui ne sont pas explicités dans DEVS), nous ajoutons la possibilité de faire référence à plusieurs sous-paramètres via un unique identifiant pour éviter l'explosion du nombre de paramètres dans les multi-modèles. Par exemple, le paramètre du temps de fin de simulation (destiné à l'algorithme de co-simulation de MECSYCO) est souvent commun à tous les sous-modèles s'ils partagent la même représentation du temps.

```
Multi-modèle := Nom, SousModèles, Interface, CouplageInternes, Information?, Simulateur;

SousModèles := SousModèle+
SousModèle := Nom, Chemin;

Interface := Paramètre*, PortEntrée*, PortSortie*;

Paramètre := Nom, ValeurParDéfaut, EPC+;
EPC := NomSousModèle, NomParamètre;
PortEntrée := Nom, PortType, EIC;
EIC := NomSousModèle, NomPort;
PortSortie := Nom, PortType, EOC;
EOC := NomSousModèle, NomPort;

CouplageInternes := CouplageInterne*;
CouplageInterne := Envoyeur, Receveur, OpérationTemporelle?, OpérationÉvènementielle?;
Envoyeur := NomSousModèle, NomPort;
Receveur := NomSousModèle, NomPort;
OpérationTemporelle := TypeOpérationTemporelle, Argument;
OpérationÉvènementielle := Chemin, ParamètresOpération?;
ParamètresOpération := ParamètreOpération+;
ParamètreOpération := Nom, ValeurParDéfaut;

Information := MotClé*, Description?;

Simulateur := ParamètreDeSimulation+;
ParamètreDeSimulation := Nom, ValeurParDéfaut, ESPC;
ESPC := NomSousModèle, NomParamètre;

//Type des ports
PortType := PortTypeSimple | PortTypeComplexe;
PortTypeSimple := TypeSimple, InitOption?, ValeurParDéfaut?;
PortTypeComplexe := Tuple | Vecteur;

// Terminaux
Nom := Identifiant;
NomParamètre := Identifiant;
NomPort := Identifiant;
NomSousModèle := [SousModèle]; //Référence à un SousModèle
Chemin := ChaineDeCaractères;
ValeurParDéfaut := Entier | Booléen | Flottant | ChaineDeCaractères;
TypeSimple := "Int" | "Boolean" | "Real" | "String";
Tuple := (TypeSimple)1..5;
Vecteur := TypeSimple+;
TypeOpérationTemporelle := ("add" | "multiply" | "divide");
Argument := Flottant;
MotClé := ChaineDeCaractères;
Description := ChainesDeCaractères;
```

FIGURE 6.7 – Syntaxe de *mm2xml* dans un format BNF.

Les couplages internes font le lien entre un sous-modèle émetteur et un sous-modèle receveur. Les opérations sur le temps ou les données peuvent ensuite être ajoutées. Les opérations temporelles sont limitées à l'addition, la multiplication et la division, représentées respectivement par les mots-clés *add*, *multiply* et *divide* suivi d'un réel utilisé en argument. Les opérations sur les données sont uniquement chargées à chaud en donnant le nom de la classe à charger et ses

```
multimodel MyMultimodel
  submodels
    name Model1 path "Library/Basic/MyModel.xml"
    name Model2 path "Library/Basic/OtherModel.xml"
  interface
    parameters
      Model1Param1:13 refers to (Model1:"Param1") (Model2:"p1")
      Model1Param2:true refers to (Model1:"Param2")
    inputs
      in:Double refers to (Model1:"in")
    outputs
      out:Double refers to (Model2:"out")
  internal couplings
    {Model1."out1"->Model2."in1"}
    {Model1."out2"->Model2."in2"
      time operation add 1.
      event operation "operation.MyOp" param=1.
    }
  information
    keywords "Multimodel" "Example" "Thesis"
    description "Example of mm2xml file."
  simulator
    simulation variables
      globalStopTime:10. refers to (Model1:"stopTime")(Model2:"stopTime")
```

FIGURE 6.8 – Exemple de description *mm2xml*.

paramètres. Dans le code de la classe, ces paramètres sont insérés sous forme de table de hachage (une *Map* Java).

Pour l'instant, l'interface des opérations (données en entrée, paramètres et données en sortie) n'est disponible que dans le code source ce qui limite la réutilisation. L'un des points d'amélioration concernant l'utilisation des opérations serait de fournir des descriptions de leur interface.

Enfin, la description informelle du modèle se fait de la même manière que dans *m2xml*.

Mécanismes de validation

Comme pour *m2xml*, l'utilisation de *mm2xml* nous a permis d'identifier plusieurs aspects de validation à ajouter à l'éditeur pour faciliter la descriptions des multi-modèles.

La première vérification que nous effectuons concerne l'écriture de chemin vers des modèles ou des opérations, nous nous assurons que les chemins désignent effectivement un fichier existant.

Nous mettons ensuite en place une vignette d'information concernant l'utilisation de FMU au sein du multi-modèle. Comme nous laissons la possibilité d'utiliser directement les FMU, la vignette d'information indique quels sont les paramètres de simulation réservés par l'artéfact de modèle FMI présent dans MECSYCO.

Notons que les mécanismes de validation que nous mettons en place ne vérifient pas que :

— les paramètres du multi-modèle font bien référence à des paramètres existants et de même type dans les sous-modèle.
— les paramètres des opérations correspondent effectivement à des paramètres disponibles pour la classe chargée.
— les ports des sous-modèles connectés entre eux par couplage interne sont bien de même type.

Ces modifications sont des pistes d'améliorations envisagées, pour l'instant elles sont implémentés en Java via des méthodes de test à appliquer sur les descriptions XML des multi-modèles.

6.2.4 Le langage *cs2xml* : description des co-simulations

Le langage *cs2xml* s'occupe du dernier niveau de description de notre approche, la description des expériences sur un multi-modèle. L'objectif de ce langage est de fournir une interface structurée et facile d'utilisation pour lancer les co-simulations MECSYCO.

Les aspects principaux de ce langage sont le choix d'un multi-modèle, le choix de ces paramètres (de modèle et de simulation), et enfin la sélection d'un ou plusieurs artéfacts d'observation.

Éléments de syntaxe

La syntaxe de *cs2xml* est présentée sur la Figure 6.9 dans un format BNF. De son côté la Figure 6.10 présente un script d'exemple pour illustrer l'utilisation du langage. Le script débute par le choix d'un nom pour l'expérience.

Un modèle (atomique ou couplé) est sélectionné ensuite via un chemin vers sa description, puis ses paramètres sont choisis. À noter qu'il n'est pas nécessaire de donner une valeur à tous les paramètres, la valeur par défaut peut être utilisée.

L'observation est ensuite détaillée. Le langage propose deux artéfacts d'observation (*GraphiqueTX* et *GraphiqueXYZ*) identifiés respectivement dans XText par les mots clés "*TX*" et "*3D*". Le premier permet de tracer l'évolution de variables numériques suivant un axe temporel, tandis que le second permet d'afficher une courbe en trois dimensions. Deux paramètres d'affichage peuvent être choisis :

— "*display*" qui prend comme valeur "*live*" pour un affichage en cours de simulation ou "*postmortem*" (la valeur par défaut) pour un affichage après la simulation.
— "*renderer*" donnant le choix entre un affichage avec des points ("*Dot*"), des escaliers ("*Step*") ou des lignes ("*Line*" qui est le choix par défaut). Disponible seulement pour *GraphiqueTX*.

À ces deux types de systèmes d'observation, nous ajoutons la possibilité de charger à chaud un artéfact d'observation développé pour d'autres besoins. Il suffit alors de spécifier le chemin vers la classe implémentée, de définir les ports d'entrée de l'artéfact, les connexions avec les ports du multi-modèle (en utilisant la même structure que les couplages en *mm2xml*), et enfin ses paramètres sous forme de clé-valeur.

Mécanismes de validation

Nous mettons en place les mêmes procédés de validation pour le langage *cs2xml* que pour le langage *mm2xml*, i.e. nous indiquons lorsque des chemins (celui du modèle, d'un artéfact d'observation, ou d'une opération) ne conduisent pas vers des fichiers existants.

6.2.5 Synthèse

Nous avons choisi de mettre en place une démarche d'IDM au travers d'un environnement basé sur des DSL pour deux raisons :

— Limiter les erreurs humaines en automatisant le passage d'une description vers un exécutable.
— Faciliter l'utilisation de la démarche proposée et de la plateforme MECSYCO.

```
CoSimulation := Nom, Modèle, Observateurs?, Information?, Simulateur;

Modèle := Chemin, Paramètres;
Paramètres := Paramètre+;
Paramètre := Nom, Valeur;

Observateurs := Observateur+;
Observateur := GraphiqueTX | GraphiqueXYZ | GraphiqueÀCharger;
GraphiqueTX := PortsÀObserver, OptionAffichage, OptionTrait;
GraphiqueXYZ := PortX, PortY, PortZ, OptionAffichage;
GraphiqueÀCharger := Chemin, ObsPorts, Couplages, Options?;
ObsPorts:=ObsPort+;
Couplages := Couplage+;
Couplage := Envoyeur, Receveur, OpérationTemporelle?, OpérationÉvènementielle?;
Envoyeur := NomPort;
Receveur := [ObsPort]; //Référence à un ObsPort du graphique
OpérationTemporelle := TypeOpérationTemporelle, Argument;
OpérationÉvènementielle := Chemin, ParamètresOpération?;
ParamètresOpération := ParamètreOpération+;
ParamètreOpération := Nom, Valeur;

Information := MotClé*, Description?;

Simulateur := ParamètreDeSimulation+;
ParamètreDeSimulation := Nom, Valeur;

// Terminaux
Nom := Identifiant;
Chemin := ChaineDeCaractères;
Valeur := Entier | Booléen | Flottant | ChaineDeCaractères;
PortsÀObserver := Identifiant+;
ObsPort := Identifiant;
NomPort := Identifiant;
PortX := Identifiant;
PortY := Identifiant;
PortZ := Identifiant;
OptionAffichage := ("live" | "postmortem");
OptionTrait := ("Dot" | "Step" | "Line");
TypeOpérationTemporelle := ("add" | "multiply" | "divide");
Argument := Flottant;
MotClé := ChaineDeCaractères;
Description := ChainesDeCaractères;
```

FIGURE 6.9 – Syntaxe de *cs2xml* dans un format BNF

Chaque étape de la démarche de multi-modélisation hiérarchique proposée est supportée par un langage dédié. Nous obtenons donc un ensemble de langages complémentaires (voir Figure 6.11) qui permettent d'exprimer les différents niveaux de descriptions. Ils évitent la lourdeur de l'écriture XML et se focalisent sur les aspects essentiels plutôt que sur le code). **Chaque langage permet de générer des fichiers XML respectant les structures de description que nous avons définies**. D'un point de vue technique, nous avons utilisé l'outil XText pour ses fonctionnalités avancées, dont le déploiement sous forme de plugins Eclipse.

Il faut noter qu'au niveau des descriptions atomiques, le langage *m2xml* peut être enrichi pour ajouter des informations nécessaires à l'intégration de modèles venant de simulateurs spéci-

```
co-simulation MyCoSimulation
model
  path "Path/to/modelDescription.xml"
  parameters
    param=1
observing
  artifact TX ports out1 out2 out3 display live renderer Step
  artifact 3D ports X:out1 Y:out2 Z:out3
  artifact "myObservingArtifact" ports out1
    links {"out1"->out1}
    settings param1=true param2=5
information
  keywords "Example" "Co-simulation" "Thesis"
  description "Example of a cs2xml file."
simulation parameters
  stopTime=10.
```

FIGURE 6.10 – Exemple de description *cs2xml*.

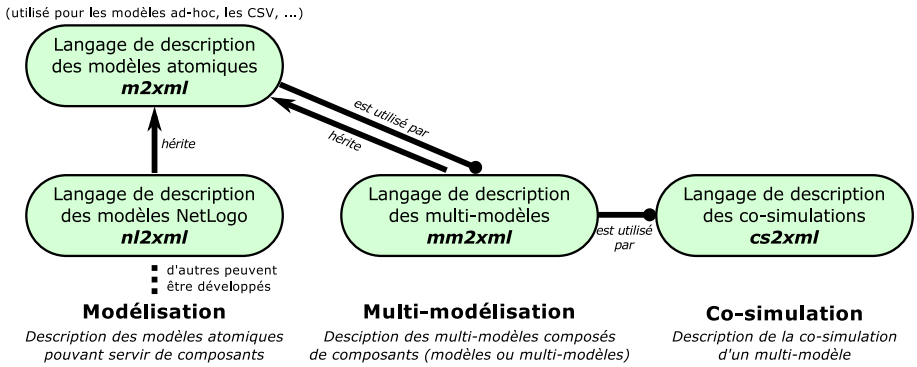


FIGURE 6.11 – À chaque étape son langage dédié

fiques. Cet aspect est détaillé au chapitre 7 pour l'intégration du simulateur multi-agent NetLogo [Wilensky, 1999]).

Pour finir, le Figure 6.12 résume la chaine de traitements utilisée dans l'environnement. Le modélisateur utilise les DSL comme interface d'utilisation pour décrire les modèles, les multi-modèles et les expériences qu'il souhaite réaliser. Ces DSL génèrent les documents de description et ceux-ci sont lus par un programme Java qui les traduits dans l'intergiciel MECSYCO pour lancer la co-simulation et produire les résultats.

6.3 Conclusion

Ce chapitre présente la contribution principale de cette thèse : **l'élaboration d'une démarche associant la multi-modélisation hiérarchique et la co-simulation de modèles hétérogènes**. Cette démarche s'est appuyée sur :

1. L'amélioration de l'intergiciel MECSYCO qui proposait déjà, au travers de DEVS et AA4MM, une approche rigoureuse d'intégration de modèle et de co-simulation. La possibilité de multi-modélisation hiérarchique est alors apportée par une démarche descriptive.
2. Une documentation en trois étapes :

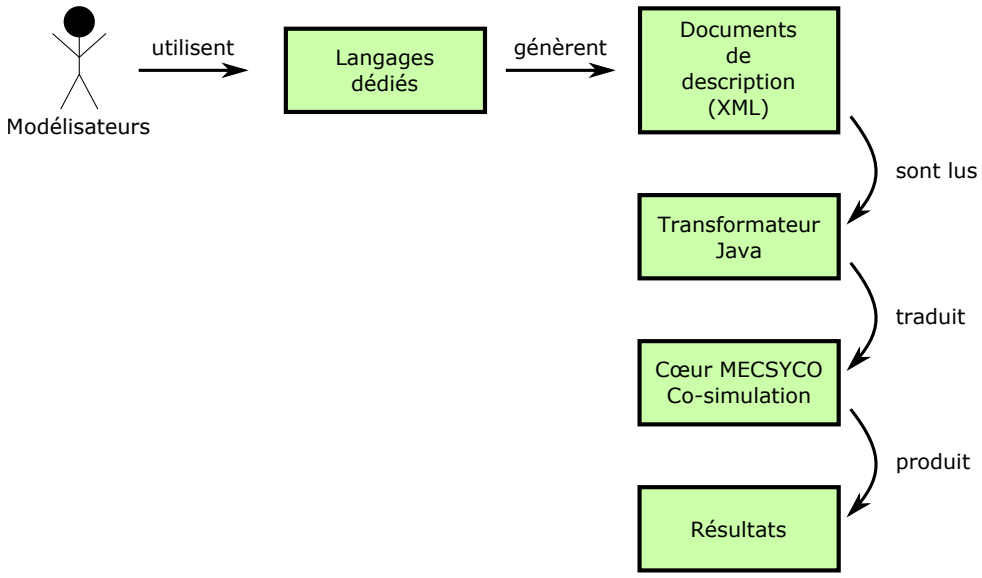


FIGURE 6.12 – Chaine de transformation des DSL jusqu'aux résultats de la co-simulation.

(a) La définition des modèles atomiques, notamment de leur interface ce qui complète leur encapsulation formelle pour permettre leur réutilisation. C'est la première étape en vue de constituer une bibliothèque des modèles intégrés dans MECSYCO (l'une des perspectives de notre travail).

(b) La définition des multi-modèles de sorte à pouvoir les instancier automatiquement mais aussi de pouvoir les réutiliser comme des modèles atomiques grâce à la propriété de fermeture par couplage. La documentation des multi-modèles est structurée selon la définition des modèles couplés DEVS, ce qui nous permet de bénéficier de la propriété de fermeture par couplage et donc d'une multi-modélisation hiérarchique.

(c) La définition des co-simulations qui permet de stocker les informations concernant les expériences effectuées et de les relancer, facilitant leur répétabilité.

3. Le développement d'un environnement à base de langages dédiés pour guider les utilisateurs au cours des trois étapes de documentation et permettre l'utilisation de MECSYCO à des non-informaticiens.

Les structures de description proposées n'ont pas vocation à rester figées, elles seront amenées à évoluer en même temps que les fonctionnalités de MECSYCO. Cependant, comme l'intégration de nouveaux simulateurs et l'implémentation de nouvelles opérations sont des étapes majeures du développement dans MECSYCO, l'ajout de nouveaux m-artéfacts et de nouvelles opérations a été facilité par une spécification de l'instanciation de ces objets et par une stratégie de chargement à chaud. Cette stratégie est propre à Java mais d'autres stratégies équivalentes peuvent être trouvées dans d'autres langages. Grâce à cela, la démarche n'est pas limitée aux simulateurs actuellement intégrés à MECSYCO mais pourra prendre en compte ceux qui seront développés ultérieurement. Nous augmentons ainsi l'extensibilité de l'approche.

Pour augmenter l'utilisabilité de la démarche, nous avons défini des langages dédiés pour soutenir chacune de nos trois étapes. Les documents de description sont au format XML pour faciliter leur traitement par un programme, mais cela les rend très verbeux à écrire pour un utilisateur humain et une écriture dans un éditeur de texte basique serait source de nombreuses erreurs. L'utilisation de langages dédiés permet de palier à ce problème tout en fournissant une

interface utilisateur donnant aux non-informaticiens la possibilité d'aller de la documentation de leurs modèles atomiques (en supposant qu'un m-artéfact existe déjà) jusqu'à la définition des co-simulations.

L'approche DSL que nous proposons est une approche d'IDM au même titre que l'interface graphique proposée dans [Camus, 2015]. Les deux sont pensées pour passer automatiquement d'une description (l'une textuelle, l'autre graphique) à l'implémentation de sorte à limiter les erreurs humaines lors de la programmation et à systématiser les processus de vérification. Les DSL profitent néanmoins de la démarche de multi-modélisation hiérarchique, et d'une meilleure extensibilité grâce à la possibilité d'ajouter des artéfacts de modèles, des artéfacts d'observation et des opérations sans modification du code de l'environnement.

Pour résumer, nous voulions développer une démarche alliant multi-modélisation hiérarchique et co-simulation de système hétérogènes issus de différents logiciels de M&S. Au niveau de la co-simulation, l'intergiciel MECSYCO nous apporte une solution pour l'intégration de simulateurs hétérogènes en s'appuyant sur une stratégie d'encapsulation formelle basée sur DEVS et sur le méta-modèle AA4MM pour gérer les hétérogénéités. Nous y ajoutons une démarche de multi-modélisation hiérarchique basée sur une documentation en trois étapes qui complète l'encapsulation formelle des modèles atomiques, permet une description exploitable et réutilisable des multi-modèles, et pérennise les informations des co-simulations pour pouvoir les rejouer. Cette démarche est supportée par des langages dédiés limitant les erreurs d'implémentation humaines et permettant à des non-informaticiens d'accéder à l'outil.

Ce chapitre s'est focalisé sur la démarche de multi-modélisation et co-simulation, mais pour chacune des activités identifiées des pistes de recherche existent pour offrir de nouvelles possibilités. Le prochain chapitre est dédié à d'autres contributions qui s'inscrivent aux différents niveaux du cycle de M&S. Ces contributions ont nourri notre réflexion et sont sources de plusieurs fonctionnalités de MECSYCO qui sont exploitées dans les exemples du chapitre 8.

7

Apports à l'activité de multi-modélisation et co-simulation

Le chapitre précédent a montré comment nous avons mis en œuvre une démarche hiérarchique de multi-modélisation et co-simulation pour MECSYCO. Dans ce chapitre nous revenons sur les étapes de cette démarche pour y présenter des contributions plus spécifiques.

Nous avons détaillé quatre étapes : le développement des modèles atomiques (qui se fait dans un logiciel de M&S dédié), l'intégration (composée de l'encapsulation logicielle et formelle puis de la description de l'interface des modèles), la multi-modélisation (où sont effectués les couplages et la constructions hiérarchique) et enfin l'expérimentation (la définition des co-simulations). La Figure 7.1 résume notre démarche descriptive appliquée à MECSYCO et positionne les 4 contributions spécifiques que nous faisons dans ce chapitre.

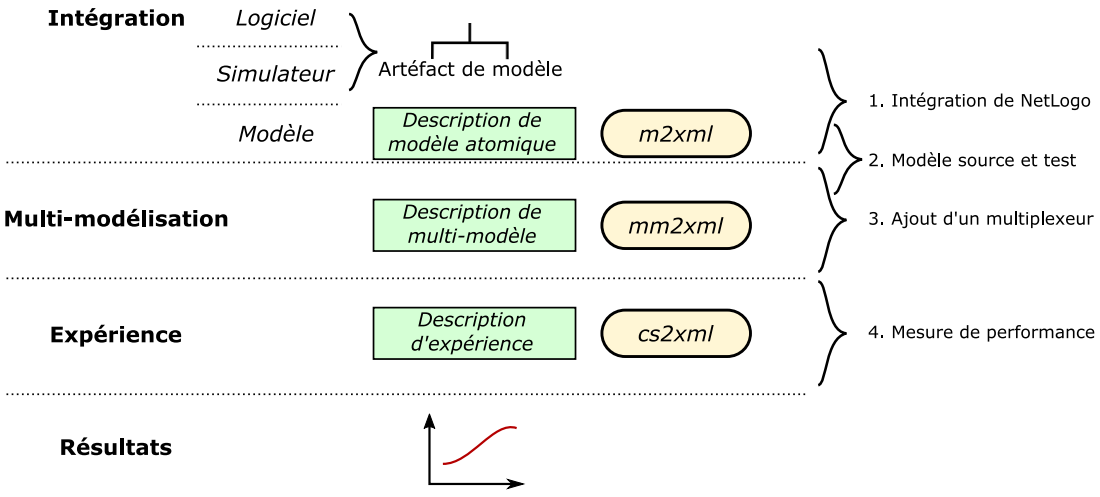


FIGURE 7.1 – Positionnement des contributions dans notre démarche descriptive.

Notre premier apport se situe au niveau de l'intégration de nouveaux simulateurs et concerne celle de NetLogo[Wilensky, 1999]. Elle a abouti au développement d'un nouveau m-artéfact plus générique pour MECSYCO.

Englobant l'intégration et la multi-modélisation, nous nous intéressons ensuite à la notion de modèles sources permettant de tester la dynamique des modèles et multi-modèles avant de les coupler. Nous aboutissons là aussi au développement de m-artéfacts spécifiques pour MECSYCO.

Nous présentons par la suite l'ajout d'un artéfact multiplexeur dans AA4MM et MECSYCO. Cet artéfact concerne la gestion de couplages complexes entre modèles lors de la phase de multi-modélisation. Il offre aussi un moyen de traiter des évènements simultanés.

Enfin, nous revenons sur la notion de performance des modèles en expliquant la création d'une boite à outils permettant d'effectuer des mesures (temps de calcul et log d'évènements) sur les co-simulations MECSYCO. Ce dernier aspect concerne la phase d'expérimentation.

7.1 Intégration d'un simulateur multi-agent

Pour rappel, lors de la multi-modélisation d'un système complexe, chaque modèle représente une perspective, un aspect du système global. Le principe est alors d'utiliser le formalisme le plus adéquat pour chaque sous-modèle. Dans l'approche par intégration et co-simulation proposée par MECSYCO, le principe est d'utiliser le simulateur (le logiciel de M&S) le plus adéquat mais un travail d'intégration est nécessaire pour cela. Plus il y a de simulateurs différents dans MECSYCO (et moins il y a de contraintes à leur utilisation en co-simulation), plus la plateforme offre de perspectives pour la multi-modélisation et la co-simulation.

Dans cette section nous nous intéresserons à l'intégration de simulateurs multi-agents dans une co-simulation. L'intégration se concrétise par le développement d'un nouvel artéfact de modèle pour NetLogo. Cette contribution a été présentée dans [Paris et al., 2017a] et [Paris et al., 2018b]. L'objectif était alors de travailler sur l'intégration rigoureuse de systèmes multi-agents au sein d'une co-simulation DEVS en prenant l'exemple de l'intégration de NetLogo dans MECSYCO.

Dans le cadre de cette thèse, ce travail d'intégration et le développement de l'artéfact de modèle qui a suivi a permis de travailler la démarche d'intégration des simulateurs et de mettre en évidence la nécessité d'utilisation de fichiers de description. Un langage dédié a été spécifiquement développé pour guider le travail d'intégration des modèles NetLogo dans les co-simulations MECSYCO.

7.1.1 Contexte

L'intérêt de l'approche multi-agent est qu'elle est particulièrement adaptée à la représentation de systèmes composés d'un grand nombre d'entités hétérogènes en interaction, ce qui correspond à la définition des systèmes complexe (voir Définition 11, chapitre 3). De plus, elle permet de représenter à la fois les niveaux individuels et collectifs [Michel et al., 2009]. C'est donc une approche de choix pour modéliser et simuler des systèmes complexes.

L'approche multi-agent est adaptée pour les modèles sociaux (dynamique de population, lois des marchés...). Dans le cadre de la co-simulation, la modélisation de systèmes socio-techniques peut passer par le couplage entre un simulateur physique et un simulateur multi-agent comme c'est le cas dans [Bollinger et al., 2016]. L'intégration d'un simulateur multi-agent dans notre intergiciel est donc un moyen d'étendre nos possibilités de modélisation.

D'un point de vue purement multi-agent, la question qui nous intéresse est *comment représenter un système complexe à partir de plusieurs modèles multi-agents*, chacun offrant une perspective complémentaire de ce système. Dans notre approche par co-simulation, la question revient à faire interagir différents simulateurs multi-agents pour qu'ils échangent de l'information et synchronisent leur exécution.

7.1.2 Présentation de NetLogo

NetLogo [Wilensky, 1999][Tisue and Wilensky, 2004] est un environnement de modélisation et simulation de systèmes multi-agents. Il est utilisé pour l'éducation et la recherche dans la modélisation de systèmes complexes et de phénomènes sociaux.

NetLogo permet grâce à un langage de programmation dédié de définir des agents (appelés *turtles*) qui évoluent dans un environnement et de spécifier leurs comportements. L'environnement est un plan dans lequel les agents peuvent se déplacer, il contient des cases (appelée *patch*) qui peuvent avoir des attributs. Un modèle NetLogo est composé de :

1. Une interface graphique contenant des graphes, un affichage de l'environnement et des agents, des boutons pour spécifier les paramètres du modèle et pour lancer la simulation et un interpréteur de commandes qui permet d'interagir avec le modèle avant, pendant ou après la simulation. L'interface graphique sert donc autant à la visualisation qu'à l'interaction avec le modèle pendant l'exécution.
2. Un script écrit dans le langage NetLogo qui correspond au modèle implémenté. C'est principalement dans ce script que sont définies les variables du modèle et les méthodes régissant le comportement des agents au cours de la simulation.
3. Une documentation informelle du modèle qui explique ce qu'il représente, comment il fonctionne et comment l'utiliser.

NetLogo utilise un simulateur discret à pas de temps (appelé *tick*). Par convention, la simulation correspond à l'appel d'une méthode d'initialisation (*setup*) puis à des appels successifs à la méthode *go* qui spécifie ce que font les agents à chaque pas de temps et fait avancer le temps de la simulation (Figure 7.3).

7.1.3 Développement de l'artéfact de modèle NetLogo

Le développement d'un artéfact de modèle se fait en 2 étapes :

1. Intégration logicielle : Cela consiste à effectuer le pont logiciel permettant de faire interagir l'intergiciel MECSYCO avec NetLogo. Cette étape est simple puisque NetLogo met à disposition une API permettant d'interagir avec ses modèles directement depuis Java.
2. Intégration formelle : Cela consiste à rendre compatible les dynamiques des simulateurs. Dans notre cas, cela consiste à effectuer l'encapsulation en DEVS de NetLogo. Il faut pour cela créer une interface entre NetLogo et les 5 méthodes du protocole de simulation DEVS, celles-ci permettront la gestion de la synchronisation et de l'échange de données entre NetLogo et le reste de la co-simulation. Il est important de noter que pour avoir un modèle artéfact générique, il faut que ces cinq méthodes soient indépendantes d'un modèle particulier.

Particularité de NetLogo

Les modèles développés dans NetLogo le sont dans un langage de modélisation spécifique. Ils ne sont généralement pas prévus pour fonctionner en co-simulation. Notamment, vis-à-vis de MECSYCO, les modèles doivent avoir des ports d'entrée et des ports de sortie par lesquels des évènements externes d'entrée et de sortie peuvent être intégrés. NetLogo n'a pas cette notion de port ni d'évènement. En revanche, il est possible d'interagir avec les modèles NetLogo à chaque pas de temps grâce à un interpréteur qui prend en entrée du code NetLogo. L'API NetLogo permet d'utiliser cet interpréteur avec deux types d'instructions :

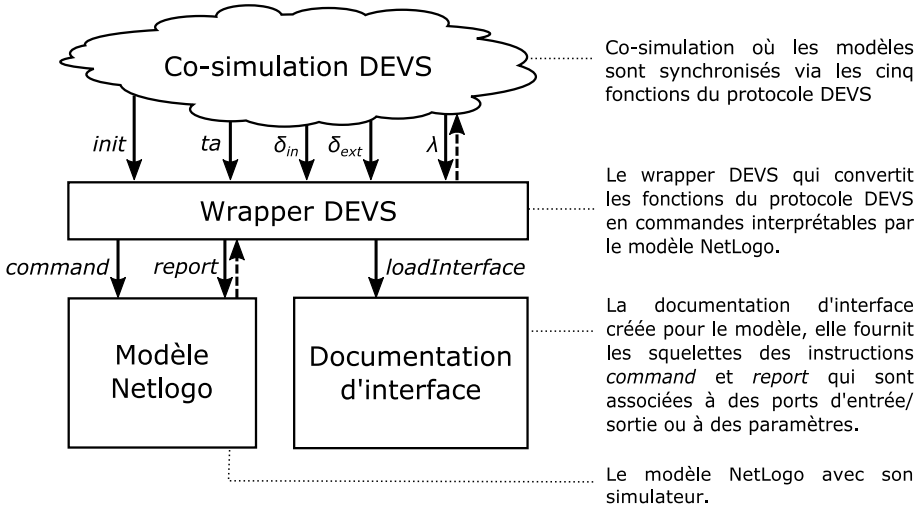


FIGURE 7.2 – Schéma de l'artéfact de modèle NetLogo

1. Les instructions *command* qui permettent d'exécuter du code sans retour.
2. Les instructions *report* qui permettent de récupérer des informations sur le modèle.

Il faut alors faire le lien entre l'interpréteur de NetLogo et les notions de ports d'entrée et de sortie associé à des évènements acceptables. Cela permet permettra ensuite de faire le lien entre l'exécution d'un modèle NetLogo et les 5 méthodes du protocole de simulation DEVS. Concrètement, nous allons associer des commandes à chaque ports d'entrée ou de sortie des modèles NetLogo.

Fonctionnement de l'artéfact NetLogo

Nous avons décomposé l'intégration des modèles NetLogo en deux étapes : un artéfact de modèle qui fait le lien entre l'API NetLogo et les 5 méthodes du protocole de simulation DEVS, et un document d'interface (en XML) qui permet de définir les ports d'entrée et de sortie (ainsi que les paramètres) des modèles NetLogo en associant chaque nom de port à une instruction à exécuter. Les éléments spécifiques d'un modèle particulier sont donc précisés dans le document d'interface, ce qui permet d'avoir un modèle artéfact NetLogo générique. La Figure 7.2 montre l'architecture de la solution proposée.

Nous allons maintenant expliquer comment nous avons implémenté les 5 méthodes du protocole de simulation DEVS.

Initialisation : *init()*

Cette méthode doit permettre d'initialiser le modèle avant la simulation. Dans NetLogo, l'initialisation se fait d'abord au niveau de l'interface graphique où des paramètres peuvent être choisis (cela peut être fait au travers de l'interpréteur)), c'est ensuite la méthode *setup* qui crée l'environnement et les agents en fonction des paramètres.

Les paramètres étant particulier à chaque modèle, nous avons spécifié une section contenant les noms des paramètres associés à des commandes dans le document d'interface. Cela permet de modifier les paramètres issus de l'interface graphique avant l'appel de la fonction *setup*.

Évènement interne : *processInternalEvent()*

La fonction *processInternalEvent()* correspond à l'exécution d'un évènement interne au modèle. Dans le cas de NetLogo, cela correspond directement à l'appel de la méthode *go* qui effectue les

modification à effectuer à chaque *tick*.

*Temps du prochain évènement : getNextInternalEventTime()
Cette fonction est liée à la synchronisation entre les modèles, elle renvoie le temps du prochain évènement interne du modèle. Le simulateur Netlogo est un simulateur discret, chaque pas de calcul correspond à un *tick*. Ces *ticks* ne correspondent pas toujours à un temps. Pour l'interaction avec des modèles temporels au sein d'une co-simulation, il est nécessaire de donner une signification temporelle au *tick* de NetLogo. Comme dans [Quesnel et al., 2005], nous choisissons de laisser le modélisateur choisir une durée fixe t correspondant à un *tick* de NetLogo. Ainsi, la fonction *getNextInternalEventTime()* renverra toujours t peu importe l'état du modèle.

Évènement entrant : processExternalInputEvent()
Cette fonction doit permettre la gestion des évènements entrants dans le modèle lors de la co-simulation. Elle prend en paramètre un nom de port associé à un évènement composé d'une donnée et d'un temps.

Comme il n'y a pas de notion de port d'entrée dans NetLogo, et que cela est spécifique à chaque modèle, nous utilisons le document d'interface pour spécifier des ports. Nous avons défini dans le document d'interface une section où le modélisateur peut définir des noms de ports et y associer des commandes. Ces commandes sont en fait incomplètes car il faudra y intégrer les données des évènements. Les espaces où les données doivent être intégrées sont représentés par des "%s".

À chaque réception d'un évènement d'entrée, l'artéfact de modèle récupère la commande associé au port de l'évènement. Ensuite, il remplace les "%s" trouvé dans la commande par les données reçues.

Nous distinguons deux types d'évènements d'entrée dans NetLogo :

1. Les évènements dont le nombre de données est fixe et qui sont associés à une ou plusieurs commandes. C'est notamment le cas lorsque l'évènement modifie une ou plusieurs variables globales, exécute une action sur un ensemble d'agent prédéfini...
2. Les évènements dont le nombre de données est variable. Par exemple, lorsqu'un groupe d'agents vérifiant une propriété doit être intégré au modèle, nous associons une commande pour l'intégration d'un agent et nous répétons cette commande pour chaque agent. Notons qu'il est possible d'effectuer cette même opération pour impacter un ensemble de *patch*.

Évènement sortant : getExternalOutputEvent()
La gestion des évènements sortants est similaire à la gestion des évènements entrants. Nous spécifions dans le document d'interface des noms de ports associés à des instructions de type *report* pour récupérer des données du modèle.

Nous ajoutons cependant la possibilité d'utiliser aussi des commandes de sorte à modéliser les évènements impactant aussi le modèle lors de leur occurrence. Par exemple, si l'évènement correspond à un groupe d'agents sortant du modèle, nous avons besoin d'une part de récupérer ces agents pour les envoyer à un autre modèle, et d'autre part nous devons les éliminer du modèle actuel.

7.1.4 Développement d'un DSL pour l'intégration de NetLogo

Le document d'interface nécessaire pour l'intégration d'un modèle NetLogo dans une co-simulation, correspond en fait au document de description des modèles atomiques que nous avons vu chapitre 6. La différence est que NetLogo ne proposant pas d'interface sous forme de port, nous devons y ajouter le lien entre les ports que nous définissons et les commandes à appliquer.

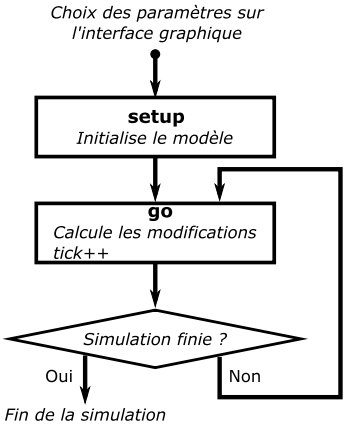


FIGURE 7.3 – Schéma de simulation d'un modèle NetLogo

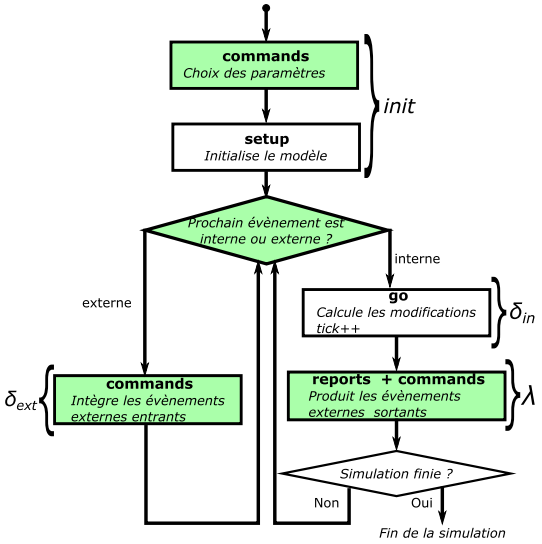


FIGURE 7.4 – Schéma de simulation de l'artéfact de modèle NetLogo

Pour faciliter ce travail, un DSL spécifique appelé *nl2xml*[16] a été développé. Il reprend les concepts déjà présents dans le langage *m2xml* décrivant les modèles atomiques, et y ajoute les éléments spécifiques à NetLogo, i.e. les commandes (sous forme de chaine de caractères) à appliquer pour les paramètres et les ports.

Concrètement, le langage *nl2xml* reprend la syntaxe du langage *m2xml* qui permet de décrire les modèles. Mais pour chaque paramètre, il ajoute la définition d'une commande à appliquer sur le modèle. Pour les ports d'entrée, il ajoute la possibilité d'y associer plusieurs commandes pour récupérer des données (les commandes *report* de NetLogo), et pour impacter le modèle au besoin (commande *command* de NetLogo). De la même manière, une ou plusieurs commandes peuvent être associées aux ports de sortie. Un exemple de script *nl2xml* est disponible en Annexe B.1.3 et a été utilisé pour le premier exemple du chapitre 8.

7.1.5 Discussion

Ce travail est basé sur l'utilisation d'un document d'interface qui permet de faire le lien entre l'interpréteur de commande de NetLogo et la notion d'évènements et de ports d'entrée et sortie de MECSYCO. Cela nous permet d'avoir un artéfact de modèle NetLogo générique pour intégrer des modèles NetLogo dans les co-simulations MECSYCO basées sur DEVS. Nous avons ensuite développé un DSL spécifique pour simplifier et guider l'écriture du document d'interface. Il guide le travail d'intégration des modèles NetLogo. En particulier, un modélisateur habitué à NetLogo n'a pas besoin d'écrire une seule ligne de code Java pour intégrer son modèle dans une co-simulation. Une question ouverte sur ce travail serait de tester cette approche d'intégration sur d'autres simulateurs disposant d'un interprète, par exemple MatLab, et voir si elle peut se généraliser.

Il est important de noter que plusieurs hypothèses ont été faites pour réaliser cette encapsulation. En premier lieu, nous supposons que les pas de temps des modèles NetLogo sont constants, or il est possible d'avoir des pas de temps variables. Pour prendre en compte cela, il faudrait

16. Le but est de ne pas complexifier le langage *m2xml* qui nous sert de base et qui permet de décrire des modèles ayant déjà une interface avec des ports d'entrée/sortie, peu importe leur origine.

que les modèles que nous souhaitons intégrer mettent à disposition une méthode permettant de rendre le temps de leur prochain évènement interne (dit autrement, à chaque appel de la méthode *go*, les modèles devraient être capable de calculer la durée de leur prochain *tick*). L'hypothèse que chaque modèle est capable de calculer le temps de son prochain évènement interne est impérative pour l'encapsulation en DEVS.

L'algorithme de co-simulation de MECSYCO impose aussi qu'à chaque exécution d'un évènement interne, l'horloge du modèle avance (voir Figure 7.4), nous ne pouvons donc intégrer que les modèles ayant une évolution à chaque appel à la méthode *go*. Enfin, notre proposition impose que les modèles soient conçus de sorte à pouvoir être co-simulés (à cause notamment de l'ordre dans lequel les méthodes sont appelées dans l'artéfact de modèle).

En résumé, l'artéfact de modèle NetLogo et le DSL développés nous permettent d'intégrer simplement un grand nombre de modèles NetLogo. Mais pour cela, les modèles doivent respecter les contraintes sus-mentionnées. La question des hypothèses et pré-requis nécessaires pour intégrer un simulateur et ses modèles est générale et se pose en réalité pour chaque travail d'intégration.

7.2 Tester la dynamique d'un modèle

Nous avons vu au chapitre 2 l'importance du cadre expérimental qui correspond aux conditions dans lesquelles nous souhaitons évaluer notre modèle. Ces conditions rassemblent (1) l'initialisation du modèle, i.e. l'état dans lequel il est placé en début de simulation, et (2) l'ensemble des stimulus qu'il reçoit en cours de la simulation, i.e. les évènements qu'il reçoit. Dans cette section nous nous intéressons au second point, l'envoi de stimulus aux modèles.

Cet aspect est lié aux étapes d'intégration des modèles et de multi-modélisation. Pour chacune d'elles il est nécessaire de tester le comportement des modèles (atomiques ou couplés) avant de les utiliser au sein d'autres multi-modèles. Il est aussi souvent nécessaire de les confronter à des données réelles pour vérifier leurs comportements.

7.2.1 Besoins

Un premier besoin important de la spécification des stimulus de nos modèles concerne l'utilisation de données réelles. Par exemple, l'exploitation de données météorologiques pour tester des modèles de thermique des bâtiments.

Un deuxième besoin concerne le test des modèles. Dans une démarche de multi-modélisation, chaque modèle est élaboré avec son propre cadre expérimental. En plus de devoir vérifier que les cadres expérimentaux sont cohérents entre eux, il faut aussi pouvoir vérifier que chaque modèle se comportera comme prévu au sein du multi-modèle. Dit autrement il est important de pouvoir tester indépendamment chaque modèle pour vérifier son comportement avant de le coupler à d'autres modèles. Pour cela il faut pouvoir effectuer deux types d'exploration :

— **Une exploration statique** qui consiste à explorer l'influence de plusieurs jeux de paramètres sur le comportement du modèle (ce qui peut servir à le calibrer). Cet aspect est déjà abordé par le langage *cs2xml*.
— **Une exploration dynamique** qui consiste à explorer l'influence de certains évènements d'entrée sur le modèle, pendant la simulation, pour pouvoir vérifier ses réactions.

C'est en associant ces deux types d'exploration que nous pouvons mettre nos modèles dans les situations expérimentales adaptées pour les tester.

7.2.2 Création de modèles sources

Pour répondre aux besoins précédents, une solution consiste à créer des modèles sources, i.e. des modèles sans ports d'entrée et qui génèrent des évènements préalablement spécifiés par le modélisateur. Pour MECSYCO, nous avons choisi comme première approche de représenter ces modèles sources par des fichiers CSV et d'implémenter un artéfact de modèle spécifique chargé de leur lecture.

Le format retenu pour le fichier CSV présente les évènements ligne par ligne, chaque colonne représentant un port (sauf la première qui correspond au temps). Les évènements qu'il contient n'ont donc que des données de type simple.

L'artéfact de modèle CSV rend possible d'une part l'intégration de données réelles si celles-ci sont stockées au format CSV (des adaptations peuvent être nécessaires). D'autre part, cela permet de tester la dynamique d'un modèle en 3 étapes :

1. Spécification de l'ensemble des évènements qui doivent venir stimuler le modèle dans un fichier CSV dont l'interface est décrit à l'aide de *m2xml*.
2. Définition du multi-modèle contenant le modèle à tester couplé au modèle source.
3. Définition de l'expérience, des jeux de paramètres à tester pour le modèle ainsi que des ports de sortie à observer.

7.2.3 Bilan

La notion de modèles sources pour intégrer des données dans les co-simulations MECSYCO est une solution pratique répondant au besoin de tester et d'alimenter nos modèles. Cela permet de mettre en place une exploration de la dynamique des modèles avant de les utiliser en couplage. Nous avons pour l'instant mis en place la lecture de fichier CSV qui nous servent de générateur de données, mais d'autres types de fichiers auraient pu être utilisés. Nous avons par exemple développé des artéfacts de modèle dédiés à l'intégration de base de données SQL contenant des données météorologiques.

Il est possible, dans MECSYCO, d'enregistrer les évènements sortants d'un modèle ou d'un multi-modèle lors d'une co-simulation (actuellement nous utilisons aussi un format CSV). Un modèle source peut donc aussi permettre de rejouer des co-simulations préalablement enregistrées ce qui évite d'avoir à les relancer lors de démonstration, en particulier si elles sont coûteuses en temps de calcul.

7.3 Agrégation d'évènements et artéfact multiplexeur

Cette section concerne la création d'un artéfact multiplexeur. Dans la spécification d'un multi-modèle MECSYCO, il n'est pas possible de connecter deux artéfacts de couplage sur le même port d'entrée d'un m-agent. Sinon, deux évènements simultanés pourraient arriver sur ce port ce qui engendrerait un conflit non géré. À l'étape de multi-modélisation de notre démarche, dans le cadre de la réutilisation de modèles existants, il est possible que l'évènement entrant dans un modèle/multi-modèle doive être généré à partir des sorties de plusieurs modèles sources (nécessité d'agréger des données) ou que deux modèles transmettent leurs évènements au même port (nécessité de gérer les conflits).

L'intérêt du multiplexeur découle de la manière dont DEVS est utilisé dans MECSYCO, qui utilise une implémentation décentralisée du formalisme DEVS classique. Pour montrer ce que cela implique, nous allons commencer par présenter deux types de conflits et voir comment ces conflits

sont traités dans les formalismes DEVS classique et DEVS parallèle puis dans MECSYCO. Cela permettra de d'éclaircir les raisons de l'hypothèse faite sur les connexions dans MECSYCO. Finalement nous introduirons l'artéfact multiplexeur, un artéfact de couplage particulier qui permet de connecter plusieurs ports de sortie de différents modèles pour les connecter à un même port d'entrée en réalisant au préalable une opération d'agrégation de données.

7.3.1 Différents conflits

Plusieurs évènements internes simultanés

Ce conflit arrive lorsque plusieurs modèles souhaitent exécuter leurs évènements internes simultanément. Dans le formalisme DEVS classique, ce problème est résolu grâce à la fonction de sélection $Select : 2^D - \{\} \rightarrow D$ qui choisi l'ordre dans lequel les modèles seront exécutés.

Dans le cas de MECSYCO, nous utilisons le formalisme DEVS classique avec ports mais implémenté de manière décentralisée, nous n'avons pas de fonction *Select*. Dans le cas décentralisé et parallèle, i.e. pour DEVS parallèle comme pour MECSYCO, les modèles ayant des évènements internes simultanées s'exécutent en même temps. Le problème est décalé sur la gestion entre l'exécution de l'évènement interne du modèle ou l'intégration des évènements externes arrivant au même moment.

Conflit entre évènements internes et externes

Ce type de conflit n'apparait pas dans le formalisme DEVS classique grâce à la fonction *Select*. Au contraire, le formalisme DEVS parallèle n'utilise pas de fonction globale de sélection, chaque modèle doit gérer localement les conflits entre évènements internes et externes grâce à une fonction de confluence δ_{conv}. La spécification d'un modèle atomique en DEVS parallèle est donc :

$$M_i = (X_i, Y_i, S, \delta_{ext}, \delta_{in}, \lambda, ta, \delta_{conv})$$

Par cette méthode δ_{conv}, chaque modèle définit en local s'il exécute en premier son évènement interne, s'il commence par intégrer les évènements externes arrivant au même temps ou tout autre comportement permettant de sortir du conflit.

Il faut noter que dans le formalisme DEVS parallèle, les messages échangés de contiennent pas juste des évènements associés à des ports mais des *Bag* d'évènements. Nativement, si plusieurs évènements externes arrivent simultanément sur un même port, la fonction de transition externe δ_{ext} se chargera du choix de l'ordre d'exécution. Il y a donc deux niveaux de gestion de conflit, (1) choisir entre exécuter l'évènement interne ou les évènements externes imminents (réalisé par la fonction de confluence), et (2) choisir comment intégrer les évènements externes simultanés (réalisé par la fonction de transition externe dans DEVS parallèle).

Pour MECSYCO, le choix de l'algorithme CMB est d'exécuter en premier l'évènement interne puis les évènements externes imminents, ce comportement remplace la fonction de confluence.

Pour les évènements externes simultanés entrant dans un modèle, nous faisons l'hypothèse au travers de nos artéfacts de couplage qu'un port d'entrée n'est connecté qu'à un port de sortie et que **l'ordre d'exécution des évènements sur différents ports n'a pas d'importance**. Nous n'utilisons donc que des évènements atomiques et pas des *Bags* d'évènements. Cette hypothèse entraine qu'à l'heure actuelle, il n'y a jamais de conflits d'évènements externes arrivant simultanément sur un même port dans MECSYCO. Nous verrons qu'un artéfact multiplexeur nous permettra de relaxer cette hypothèse parfois non valable.

7.3.2 Artéfact multiplexeur

Le but de la création d'un artéfact multiplexeur est de permettre la connexion de plusieurs ports de sortie sur un même port d'entrée, autorisant l'agrégation de données issus de plusieurs ports et la gestion des évènements simultanées sur un même port. Le problème de gestion des évènements simultanés est l'une des faiblesses de l'algorithme de Chandy-Misra-Bryant [Zeigler et al., 2000]).

Fonctionnement

La Figure 7.5 présente un exemple d'utilisation de l'artéfact multiplexeur pour agréger des données venant de trois modèles et les afficher sur un graphe 3D. Le multiplexeur se place entre les artéfacts de couplage connectés aux agents émetteurs et l'artéfact de couplage connecté à l'agent receveur. Le multiplexeur se charge de capter les évènements et les estampilles temporelles liées à la synchronisation, et de renvoyer l'évènement correspondant après l'application d'une opération d'agrégation (appelée dans l'exemple *StateXYZ*) ou l'estampille temporelle la plus imminente en absence d'évènement. Il faut noter que l'application de l'opération d'agrégation ne coûte pas de temps simulé (pas de délai minimum de propagation).

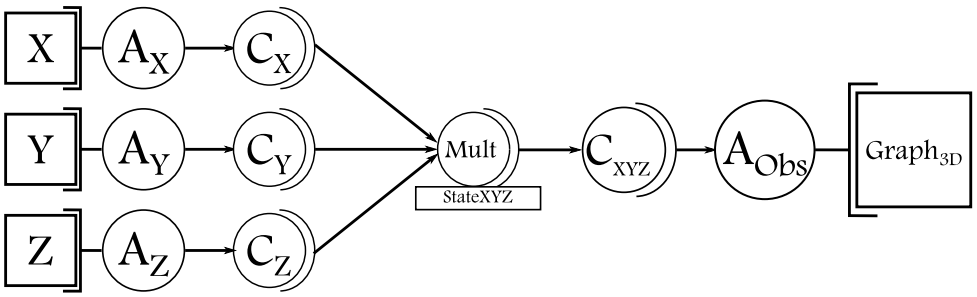


FIGURE 7.5 – Exemple d'utilisation du multiplexeur pour agréger trois données.

Pour résumer, l'artéfact multiplexeur :

1. est connecté à plusieurs artéfacts de couplage et regroupe leurs évènements sous forme de *Bag*,
2. applique une opération d'agrégation sur le *Bag* d'évènements créé pour générer un évènement unique,
3. renvoi cet évènement à un (ou plusieurs) artéfact de couplage sans influencer l'estampille temporelle.

Le multiplexeur peut donc aussi servir pour choisir dans quel ordre des évènements arrivant sur un même port doivent être appliqués.

Représentation de l'opération

Pour se rapprocher de la notation DEVS, une opération d'agrégation entre les ports de sortie *out* de n modèles m_i et les ports d'entrée *in* de k modèles m_j peut être représentée par la fonction :

$$O^{aggregation}_{(m_i^{out})_{1\leq i\leq n}\rightarrow(m_j^{in})_{1\leq j\leq k}} : V_{m_1^{out}} \times \ldots \times V_{m_n^{out}} \rightarrow \bigcap_{1\leq j\leq k} V_{m_j^{in}}$$

Avec :

— $V_{m_i^{out}}$, $0 \leq i \leq n$, l'ensemble des valeurs admissibles des ports de sortie *out* des modèles m_i.
— $V_{m_j^{in}}$, $0 \leq j \leq k$, l'ensemble des valeurs admissibles du port d'entrée *in* du modèle m_j.

L'opération d'agrégation prend en entrée un ensemble des valeurs admissibles de plusieurs ports de sortie issus d'un ou plusieurs modèles, et leur applique une transformation pour renvoyer une valeur admissible pour tous les ports d'entrée (généralement de différents modèles) qui doivent recevoir l'information.

Intégration à *mm2xml*

L'ajout de l'artéfact multiplexeur aux artéfacts de MECSYCO a été effectué dans les langages au niveau de la description des couplages. La Figure 7.6 présente deux exemples d'ajout d'opération d'agrégation en *mm2xml*, l'utilisation est la même dans *cs2xml*.

```
internal couplings
       {aggregation StateAggregationXY, ( model1."outX" , model1."outY" ) -> (model2."in")}
       {aggregation "operation.MyOp" arg1=2., (model1."a" , model2."b") -> (model3."in")}
```

FIGURE 7.6 – Description d'opérations d'agrégation

La première opération nommée "*StateAggregationXY*" permet simplement de transformer deux évènements dont les données sont des réels, en un évènement contenant un *tuple* (un paquet) de deux réels. Cette opération, déjà présente dans le cœur MECSYCO, est identifiée par un mot-clé.

La deuxième opération est identifiée par le nom de la classe qui effectuera la transformation. Cet exemple montre le cas où l'opération n'est pas disponible dans le cœur MECSYCO et devra être chargée à chaud lors de la co-simulation. Notons que les opérations d'agrégation disposent de paramètres. Dans le langage, ces paramètres sont définis juste après le nom de l'opération en donnant l'identifiant du paramètre et sa valeur.

7.3.3 Bilan

Dans le cas général, MECSYCO utilise des évènements atomiques et donc des connexions simples entre les modèles. Le multiplexeur nous permet de créer des liaisons complexes entre modèles en agrégeant des évènements venant de plusieurs sources, ainsi que de traiter des évènements simultanés arrivant sur un même port d'entrée.

Le fonctionnement du multiplexeur correspond à l'utilisation de *Bags* de données à l'image de ce qui est fait dans DEVS parallèle. Mais contrairement à ce dernier, le traitement des *Bags* est extérieur au modèle. Il est donc possible d'implémenter différentes politiques de gestion des évènements simultanés pour un modèle sans toucher à son implémentation propre ni à l'artéfact de modèle qui a permis son intégration.

L'utilisation de l'artéfact multiplexeur est permise dans le langage *mm2xml* décrivant les multi-modèles, mais aussi dans le langage *cs2xml* lors de la description des connexions aux outils d'observation (graphes...). Les opérations d'agrégation, au même titre que les autres opérations, peuvent être chargées à chaud dans MECSYCO Java. Les nouvelles opérations peuvent donc être facilement utilisées dans les langages. Pour l'instant, l'utilisation des opérations d'agrégation dans les langages et dans MECSYCO souffre des mêmes limites que les opérations standards sur les évènements. Les informations concernant les opérations (paramètres, données d'entrée et données

de sortie) ne sont disponibles que dans le code. Il faudrait y associer des descriptions pour faciliter leur réutilisation. C'est l'un des points d'amélioration de notre travail.

7.4 Outils de mesure

Nous l'avons vu, parmi les critères d'évaluation d'un modèle se trouvent les indicateurs concernant ses performances du point de vue de la précision des résultats, du temps d'exécution et de configuration requise pour la simulation [Brooks and Tobias, 1996]. Ces considérations sont liées à la phase d'expérimentation de notre démarche.

Dans cette section, nous nous intéressons à la mesure de tels indices de performance [17] au sein d'une co-simulation MECSYCO. Nous présentons comment des outils de mesure de performance ont été développés pour notre intergiciel. Ces outils ont notamment servi dans [Paris et al., 2016] pour évaluer l'impact, sur les performances, de différentes stratégies de décomposition d'un modèle continu (une FMU) en plusieurs sous-modèles (plusieurs FMU) co-simulées via MECSYCO.

7.4.1 Présentation et objectifs

Pour une co-simulation, nous avons deux niveaux d'analyse lorsque nous nous intéressons aux performances des algorithmes utilisés :

- Au niveau macroscopique, nous avons les propriétés de l'algorithme de co-simulation qui gère les synchronisations et les échanges de données.
- Au niveau microscopique, nous avons les performances des simulateurs de chaque modèle.

Dans notre cas, les modèles sont vus comme des boites noires et l'algorithme de simulation qu'ils utilisent n'est pas explicité. N'ayant pas les informations de chaque simulateur au niveau microscopique, il n'est pas possible de conduire à priori une étude analytique précise sur les performances attendues. Nous nous contentons donc de mesures empiriques effectuées au cours des co-simulations.

Nos mesures doivent nous permettre d'évaluer le déroulement d'une co-simulation MECSYCO, elles s'inscrivent sur les deux niveaux précédents :

- Microscopique :
  - Au niveau des temps, nous souhaitons avoir des indicateurs sur les temps de simulation de chaque modèles. Nous récupérons aussi les temps de calcul pris pour chaque évènement interne et externe tout en comptant le nombre de ces évènements. Ces mesures permettent d'identifier les points bloquants dans le calcul (par exemple lorsqu'un modèle se retrouve dans une configuration où il doit effectuer des calculs supplémentaires pour détecter ou intégrer un évènement).
  - Au niveau des résultats, les études analytiques n'étant pas directement possibles pour prédire la précision, nous nous contentons de récupérer les valeurs des données échangées en enregistrant les évènements d'entrée et de sortie. Cette récolte de données permet d'effectuer des comparaisons avec d'autres ensembles de données pour nous comparer à des modèles validés ou à des systèmes réels.
- Macroscopique, nous voulons la durée globale de la co-simulation (comprenant les temps de calcul, mais aussi les temps de communication). En compilant cette mesure avec les temps de chaque modèle, nous pouvons estimer les temps de communication et donc l'impact de l'algorithme de co-simulation sur le temps global. Cela permet aussi d'évaluer

17. Dans cette section, nous désignons par performance la vitesse de calcul et la précision des résultats.

les caractéristiques de différents choix de déploiement des calculs notamment lorsque l'expérience est amenée à être répétée sur une autre architecture, ou avec un niveau de résolution plus poussé.

7.4.2 Implémentation et utilisation

Les mesures que nous effectuons doivent pouvoir être faites sur tout modèle intégré dans MECSYCO (sans hypothèse sur le simulateur interne). Pour ce faire, nous basons nos outils sur l'interface de manipulation des modèles, i.e. les m-artéfacts qui permettent de manipuler les modèles à l'aide des 5 méthodes du simulateur abstrait DEVS.

Intuitivement, ajouter des fonctionnalités de mesure aux artéfacts de modèle revient à leur donner des capacités supplémentaires. Pour obtenir des outils génériques (appliquables à tous les m-artéfacts), indépendants de chaque implémentation spécifique et utilisables sans modifier les codes existants des agents et des artéfacts, nous choisissons d'utiliser le patron de conception décorateur [Gamma et al., 1994]. La Figure 7.7 présente l'architecture utilisée pour ajouter nos outils au cœur MECSYCO.

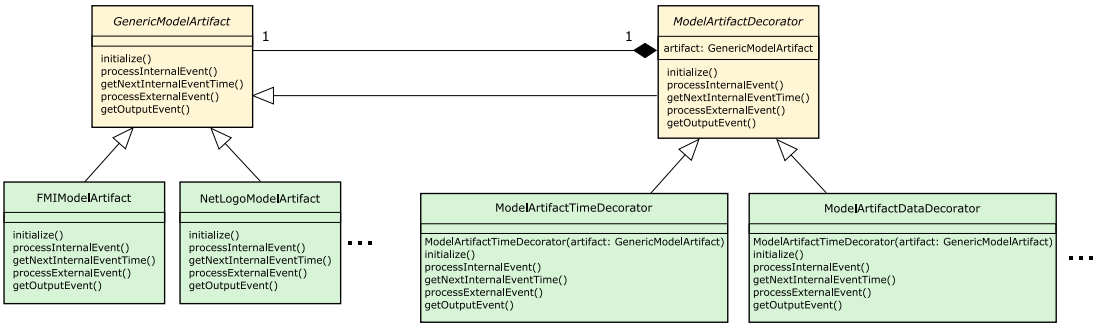


FIGURE 7.7 – Schéma UML présentant l'ajout d'outils de mesure dans les artéfacts de modèle

Chaque outil de mesure devient une surcouche que nous plaçons au-dessus des artéfacts de modèle avant de les lier aux m-agents, i.e. ce sont des modules que nous plaçons entre l'agent et son artéfact de modèle (voir Figure 7.8). Ils interceptent ainsi les commandes de l'agent et y ajoute des traitements avant de les transmettre et après leur terminaison. Notons qu'il est possible de cumuler plusieurs outils de mesure (Figure (voir Figure 7.9), la seule contrainte est qu'ils ne doivent pas se perturber les uns les autres. Par exemple, notre outil de mesure de temps ne doit pas être placé au dessus d'autres outils effectuant des calculs. Si c'est le cas, la mesure comprendra autant le temps de traitement des outils placés à sa suite que le temps de calcul du simulateur.

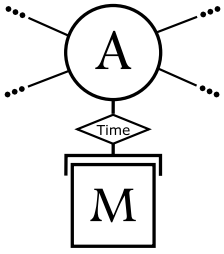


FIGURE 7.8 – Intégration dans les concepts AA4MM

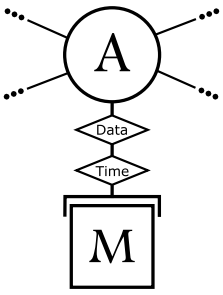


FIGURE 7.9 – Cumul d'outils

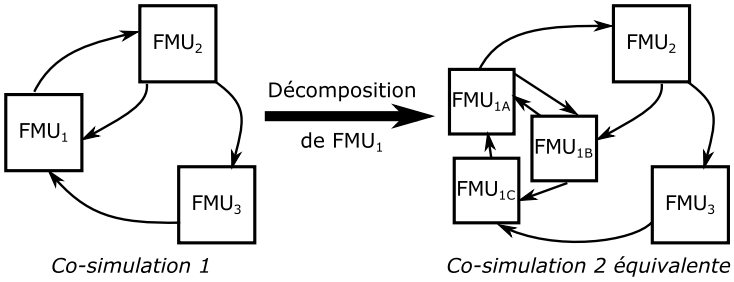


FIGURE 7.10 – Exemple de décomposition d'une FMU en plusieurs

Actuellement, deux types d'outils sont disponibles dans la version Java. Le premier s'occupe des mesures de temps sur les modèles et sur le comptage d'appels des méthodes de calcul des évènements internes et externes. Le second enregistre les évènements et les stocke dans un fichier (pour une analyse ultérieure).

Concernant ce deuxième outil, la récupération des évènements d'entrée et de sortie des modèles peut se faire via un artéfact d'observation dans MECSYCO. La différence avec l'approche présentée dans cette section est que les données sont récoltées localement, sans envoi de message à un autre agent. C'est une alternative locale aux agents d'observation.

7.4.3 Bilan

Nous nous sommes intéressés dans cette section aux problèmes de performance en M&S, en nous focalisant sur la notion d'outils utilisés pour effectuer des mesures. La stratégie que nous mettons en place se base 1) sur les interfaces des modèles intégrés dans MECSYCO, ce qui nous permet d'impacter n'importe quel modèle peu importe son origine, et 2) sur le patron de conception décorateur. Cette méthode nous permet de développer un ensemble d'outils modulaires, composables et indépendants de l'implémentation de chaque artéfact de modèle particulier.

Ces outils out été utilisés dans [Paris et al., 2016] pour étudier un problème de décomposition de modèles continus, plus précisément de décomposition d'une FMU en plusieurs FMU. La question était la suivante : dans le cadre d'une co-simulation de plusieurs FMU volumineuses (quant à la taille du système d'équations à résoudre), quelle est l'influence de la décomposition d'une FMU en plusieurs (exemple Figure 7.10) sur les temps de calcul et sur la précision des résultats ? Cette décomposition peut, par exemple, être nécessaire pour diminuer la configuration matérielle requise pour la simulation (en utilisant des FMU plus légères), ou pour modéliser une sous-partie du système via un autre logiciel plus spécialisé. L'étude a consisté en l'analyse de plusieurs co-simulations d'un même système cible suivant différentes stratégies de décomposition des FMU (ce qui correspond à différentes découpes du système d'équation global).

Le résultat général est que, dans MECSYCO, il vaut mieux utiliser une unique FMU plutôt que la décomposer en plusieurs. Ce phénomène est normal étant donné que l'algorithme de synchronisation n'est pas optimisé pour la résolution de systèmes équationnels. Dans le cas où cette décomposition est nécessaire (par exemple lorsque le système est décrit sur différents outils), l'obtention d'indicateurs sur le temps et la précision offre la possibilité de chercher un compromis entre les deux (par un choix des paramètres des simulateurs par exemple). Ces différentes questions autour des performances des co-simulations sont importantes pour la M&S de systèmes complexes car ceux-ci peuvent nous amener à mettre en relation de nombreux modèles différents.

Ce travail sur les indicateurs de performance dans nos co-simulations est une première étape

pour étudier les performances de l'intergiciel et potentiellement l'améliorer [18].

7.5 Conclusion

Nous avons présenté dans cette section plusieurs travaux ayant trait aux différentes étapes de la démarche de multi-modélisation et co-simulation que nous avons présentée au chapitre précédent (voir Figure 7.1). Ils nous ont permis d'insister sur plusieurs aspects de l'activité de M&S et d'illustrer des pistes de recherche.

Le premier élément concerne **le problème de l'intégration de nouveaux simulateurs**. Le développement d'un artéfact de modèle générique pour NetLogo a permis de faire avancer la réflexion sur l'intégration des modèles. La documentation y joue un rôle essentiel puisque cela nous permet de faire le lien entre les commandes de l'interpréteur et les ports d'entrée et de sortie des modèles. L'ajout du document d'interface complète l'étape d'intégration. Nous pourrions en perspective de ce travail vérifier si cette approche fonctionne pour d'autres simulateurs dotés d'interpréteurs de commande. Plus généralement, connaitre l'ensemble des hypothèses à faire sur les modèles et les simulateur pour pouvoir les intégrer est aussi un aspect à explorer.

Le deuxième élément concerne **le test des modèles ou multi-modèles** pour vérifier leurs dynamiques. L'idée générale est qu'il est nécessaire d'offrir des outils pour stimuler nos modèles et tester leur dynamique avant de les coupler. Nous passons par la création de modèles sources, des sources de données contenant des évènements que nous lisons au fur et à mesure de l'avancée d'une simulation pour les transmettre à nos modèles. En pratique, nous avons mis en place un artéfact de modèle lisant des fichiers CSV (n'importe quel autre format aurait pu être choisi) pour décrire simplement des évènements d'entrée pour nos modèles.

Le développement de **l'artéfact multiplexeur** ouvre les portes à des connexions plus complexes entre les modèles, mais aussi à une gestion particulière des évènements pouvant arriver simultanément sur un modèle. Ce développement nous a amené à nous interroger sur le traitement des évènements internes, externes et sur le traitement des conflits dans MECSYCO. Une piste de travail pour relaxer les hypothèses faites dans l'intergiciel serait de proposer une encapsulation formelle basée sur DEVS parallèle et un algorithme de co-simulation adapté. La question suivante étant la compatibilité entre différents algorithmes de co-simulation.

Enfin, le **développement d'outils pour obtenir des indices de performance** en matière de temps de calcul et de précision nous a fait explorer un dernier aspect lié à l'expérimentation des modèles. Ces types d'outils sont fournis dans presque tous les environnements de simulation et permettent des études empiriques sur nos simulations.

Travailler les mécanismes d'intégration, d'expérimentation, offrir de nouvelles possibilités de connexions entre modèles et s'intéresser aux performances des algorithmes utilisés sont autant d'aspects gravitant autour de l'activité de M&S. Nous pourrions y ajouter les mécanismes d'observation, le déploiement du calcul sur plusieurs machines et l'analyse des résultats... Tous ces éléments sont nécessaires pour former un environnement de M&S complet. Les contributions présentées dans ce chapitre rentrent dans ce cadre et ouvrent d'autres pistes de réflexion.

Le chapitre suivant est dédié à deux exemples mettant en pratique l'ensemble de nos contributions pour les évaluer.

18. Une perspective possible serait de proposer des algorithmes spécialisés pour gérer des sous-ensembles de modèles atomiques, e.g. gérer les FMU connectées entre elles avec un algorithme dédié qui s'intègre bien à l'algorithme de co-simulation principal.

8

Expérimentations et évaluation de la proposition

Ce chapitre est dédié à l'illustration et à l'évaluation de nos contributions. Nous nous appuyons sur deux exemples : le premier reprend un multi-modèle d'autoroute hybride tiré de [Camus, 2015] tandis que le second est inspiré du système de chauffage intelligent que nous avons utilisé dans [Camus et al., 2018].

Nous commençons par rappeler les éléments de nos travaux avant de présenter notre méthode d'évaluation ainsi que nos critères. Nous détaillons ensuite nos deux exemples illustratifs. Pour chacun d'eux, nous présentons les systèmes cibles, les modèles atomiques utilisés puis nous suivons les étapes de notre démarche pour aboutir à nos expérimentations. Nous conclurons nos expériences par des bilans sur les critères d'évaluation, les limites et les perspectives éventuelles.

8.1 Rappels et objectifs

Le but de ce chapitre est d'illustrer et d'évaluer nos travaux de thèse. Pour rappel, la première partie de nos contributions concerne :

1. l'identification de cinq étapes pour la multi-modélisation et co-simulation, puis
2. la proposition d'une démarche descriptive appliquée à MECSYCO, et enfin
3. le développement d'un environnement DSL pour soutenir l'approche.

Nous avons ensuite plusieurs contributions plus spécifiques.

4. des apports sur différents niveaux de notre activité :
   (a) intégration du simulateur multi-agent NetLogo,
   (b) définition de modèles sources,
   (c) ajout d'un artéfact multiplexeur à AA4MM et MECSYCO, et
   (d) développement d'outils de mesures pour nos co-simulations.

8.1.1 Critères d'évaluation

Nous n'évaluons pas notre structuration de l'activité de multi-modélisation et co-simulation en cinq étapes. C'est un cadre de travail, hérité de notre expérience de MECSYCO, que nous retrouvons dans les travaux connexes que nous connaissons (voir la Figure 4.7 au chapitre 4).

Nous nous contentons d'illustrer cette structure en l'appliquant à nos exemples et en utilisant notre démarche descriptive et l'environnement associé.

Concernant ces deux éléments, le premier objectif était d'avoir une approche modulaire autorisant la construction hiérarchique de multi-modèles associée à un environnement pour faciliter son utilisation. Les deux expériences nous permettent d'évaluer l'environnement (et par extension la démarche descriptive), les possibilités de hiérarchie, sa facilité d'utilisation, et aussi son extensibilité.

Un second objectif inhérent à la contribution était de conserver les propriétés et capacités existantes de MECSYCO : la gestion des hétérogénéités à tous les niveaux, la modularité et la représentation décentralisée de la co-simulation autorisant une distribution aisée des calculs.

L'évaluation de la démarche et de l'environnement portera donc sur quatre critères :

— **La conservation des capacités de gestion des hétérogénéités de MECSYCO** : Cette capacité repose sur les artéfacts de modèle (support de l'intégration logicielle et formelle) et sur les opérations temporelles et évènementielles (support de correction des différences de représentation). La conservation de cette propriété découle du fait que nous pouvons, dans notre approche, réutiliser facilement les anciens artéfacts de modèle et les opérations évènementielles comme temporelles utilisées dans des exemples précédents (voir 8.2).

— **La construction hiérarchique** des multi-modèles : C'est le but premier de la démarche, nous montrons que nous pouvons réutiliser un multi-modèle comme un modèle atomique pour construire hiérarchiquement nos multi-modèles (voir 8.3).

— **La facilité d'utilisation** : Cette propriété subjective est difficile à évaluer, nous illustrons tout de même (voir 8.2 et 8.3) la possibilité (1) de construire hiérarchiquement un multi-modèle à partir de modèles hétérogènes et (2) de le co-simuler en n'utilisant que nos langages dédiés (évitant ainsi l'utilisation de code Java pour les non-informaticiens).

— **L'extensibilité**, i.e. la possibilité d'ajouter de nouveaux composants dans l'environnement : Nous illustrons cette capacité en définissant et en ajoutant simplement de nouveaux artéfacts de modèle et de nouvelles opérations évènementielles (voir 8.2 et 8.3).

Nous détaillons et discutons les deux exemples puis, en conclusion, nous reprenons et argumentons chacun de ces critères.

Nos autres contributions plus spécifiques ne sont pas évaluées dans cette thèse. L'intégration de NetLogo et le développement d'outils de mesure de performance ont déjà été discutés respectivement dans [Paris et al., 2017b] et [Paris et al., 2016]. L'utilisation du m-artéfact NetLogo est tout de même illustré au cours de l'exemple d'autoroute hybride (section 8.2). De même, l'exemple de thermique (section 8.3) exemplifie d'une part l'utilisation de modèle source pour l'ajout de données réelles dans une co-simulation, et d'autre part l'utilisation de l'artéfact multiplexeur pour agréger des données issues de plusieurs évènements.

8.1.2 Structuration

Nous détaillons ci-dessous la structuration de nos exemples, qui reprend les étapes de notre démarche.

Contexte

Cette étape regroupe la présentation du système cible à modéliser et celle des modèles atomiques qui sont utilisés. Nous y précisons notamment l'origine de chaque modèle.

À la fin de cette étape, nous avons un ensemble de modèles hétérogènes qui doivent servir de briques de base pour la multi-modélisation.

Intégration

Pour coupler nos modèles de base hétérogènes et les faire interagir, il faut tout d'abord résoudre les problèmes d'hétérogénéités logicielles et formelles. L'intégration est la première étape à effectuer pour la co-simulation de modèles issus de différents logiciels.

Nous distinguons deux sous-parties dans l'étape d'intégration : la partie technique et la partie description. La partie technique concerne la création des artéfacts de modèle dans MECSYCO, i.e. l'encapsulation logicielle et formelle des simulateurs. La partie description s'occupe de la spécification des interfaces des modèles via l'un de nos langages dédiés.

Une fois l'étape d'intégration effectuée, les modèles atomiques hétérogènes que nous avions deviennent des composants modulaires utilisables pour construire des multi-modèles.

Multi-modélisation

Cette étape consiste en la construction progressive et hiérarchique (chaque multi-modèle étant un composant réutilisable) du multi-modèle représentant le système cible. Les multi-modèles sont décrits via notre langage dédié : *mm2xml*.

Les hétérogénéités de représentation sont résolus lors de cette étape, dans MECSYCO, par l'utilisation d'opérations associées aux couplages entre modèles.

Expérimentation

Dès que le multi-modèle représentant le système cible est construit, nous pouvons spécifier les expériences à réaliser via *cs2xml* C'est lors de cette étape que sont choisis les paramètres des expériences et les systèmes d'observation à utiliser.

8.2 Reprise d'un exemple d'autoroute hybride

Nous avons choisi pour notre premier exemple de reprendre un modèle d'autoroute hybride tiré de la thèse de Benjamin Camus [Camus, 2015][Camus et al., 2015]. Son objectif était alors de montrer les capacités de MECSYCO en terme de gestion des hétérogénéités, notamment des hétérogénéités de représentation puisqu'il met en relation des modèles multi-agents, équationnels et évènementiels.

Dans notre cas, il s'agit de démontrer que nous conservons les propriétés et possibilités de MECSYCO dans notre démarche et dans l'environnement DSL. Il illustrera aussi l'utilisation du nouvel artéfact NetLogo et l'import de nouveaux composants.

Tous les détails étant expliqués dans les papiers précédents, nous nous contenterons de présenter les grandes lignes des différents modèles et de leurs interactions sans s'attarder sur leurs comportements internes. Le modèle était inspiré de la modélisation hybride d'un flux de trafic présenté dans [Hmam et al., 2006].

8.2.1 Contexte

Le modèle d'autoroute hybride est un modèle d'autoroute découpée en trois tronçons modélisés séparément selon trois visions différentes :

1. Le premier modèle m_1 représente une route à trois voies, c'est un modèle équationnel où le trafic est vu comme un flux (à l'image de l'écoulement d'un fluide).
2. Le deuxième modèle m_2 représente une autoroute à deux voies, c'est un modèle évènementiel où les voitures sont vues de manière atomique.
3. Enfin, le troisième modèle m_3 représente une route avec une seule voie, c'est un modèle multi-agent où les voitures évoluent continuellement dans un environnement. C'est le modèle le plus coûteux en calcul mais qui présente la granularité la plus basse.

Ces trois modèles sont connectés selon la Figure 8.2.1 pour former un espace torique.

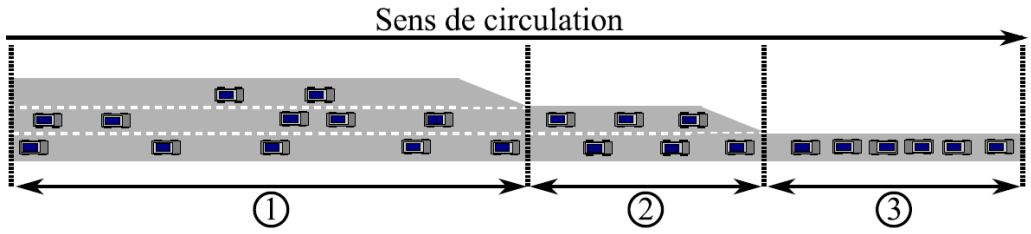


FIGURE 8.1 – Représentation des trois tronçons d'autoroute. Tiré de [Camus, 2015]

Les trois modèles présentent des niveaux de représentation différents, macroscopique pour le modèle équationnel qui considère un flux de voitures, et microscopique pour les deux autres modèles qui voient les voitures comme des entités atomiques. C'est un multi-modèle multi-niveaux où chaque modèle aborde un niveau de détail et de résolution différent. La difficulté est donc de coupler ces niveaux de représentation. Dans notre cas, ces modèles utilisent des simulateurs différents et sont écrits dans des logiciels différents. Le multi-modèle illustre donc bien la capacité de MECSYCO à résoudre les problèmes d'hétérogénéités logicielles et formelles.

Le modèle cellulaire équationnel m_1

Le premier tronçon à trois voies est représenté de manière macroscopique via des équations différentielles, nous considérons alors un flux de voitures défini par sa densité, son débit et une vitesse moyenne. Ce modèle correspond au modèle équationnel présenté dans [Hmam et al., 2006].

Le modèle se compose d'un ensemble de cellules identiques qui représentent des portions successives du tronçon. Chaque cellule peut se représenter comme suit (Figure 8.2).

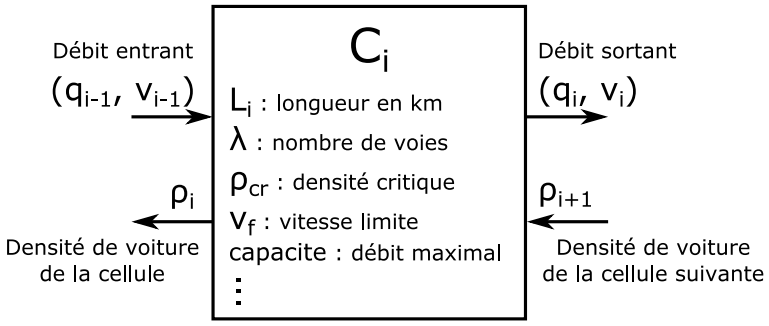


FIGURE 8.2 – Représentation d'une cellule du modèle m_1.

Chaque cellule reçoit en entrée un flux de voiture (en véhicule/heure) venant de la cellule précédente et la densité de véhicule dans la cellule suivante (en véhicule/km/voie). À partir de

ces informations, elle met à jour son propre débit de voitures sortantes et sa propre densité de véhicules qu'elle envoie respectivement à la cellule suivante et à la cellule précédente. L'ensemble des cellules permettent alors d'observer l'évolution des flux de véhicules dans notre tronçon d'autoroute à trois voies. Le tout peut se voir comme un unique modèle *Cell*, contenant deux ports d'entrée et deux ports de sortie (Figure 8.3).

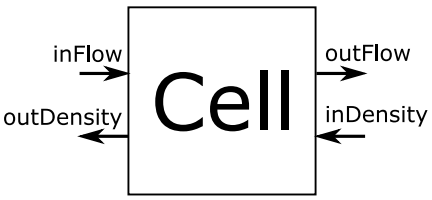


FIGURE 8.3 – Représentation schématique du modèle m_1 à trois voies.

Le modèle évènementiel m_2

Le modèle m_2 qui représente le tronçon à deux voies est un modèle évènementiel stochastique. Pour chaque voiture en entrée du modèle, celui-ci détermine sa date de sortie en fonction de la longueur du tronçon et de la vitesse moyenne de la voiture (choisie aléatoirement entre une vitesse minimale et une vitesse maximale). Les évènements internes du modèles correspondent donc aux dates de sortie des véhicules.

Le modèle peut se représenter comme suit (Figure 8.4). Dans ce modèle, nous considérons que les voitures vont toujours à leur vitesse souhaitée, il n'y a donc n'y ralentissement ni bouchon. De ce fait, nous ne considérons qu'un seul port d'entrée *in* et un seul port de sortie *out* car il est inutile de savoir les voies sur lesquelles elles rentrent ou sortent.

Le modèle multi-agent m_3

Le troisième modèle est basé sur le modèle *Traffic Basic* disponible dans la bibliothèque de modèles NetLogo [Wilensky, 1999].

Comme nous pouvons le voir sur la Figure 8.5, le modèle utilisé représente une route unique sur laquelle des voitures se déplacent de la gauche vers la droite. Contrairement au modèle original qui est torique, celui-ci est ouvert, i.e. les voitures arrivant à l'extrémité droite du monde disparaissent.

Les voitures sont représentées par des *turtles*, i.e. des agents NetLogo. Chaque voiture dispose d'une vitesse maximale, d'une puissance d'accélération et d'une capacité de freinage. Les voitures accélèrent jusqu'à une vitesse maximale et freinent lorsqu'elles veulent éviter de rentrer en collision avec une autre, dans ce cas elles synchronisent leur vitesse sur la voiture placée juste devant elles et peuvent s'arrêter complètement si nécessaire.

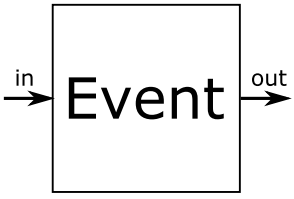


FIGURE 8.4 – Représentation schématique du modèle m_2 à deux voies.

FIGURE 8.5 – Modèle *Traffic Basic* de NetLogo adapté.

Comme vu en 7.1, NetLogo est un simulateur à pas de temps cyclique. L'interface du modèle comprend un port d'entrée *in* permettant de faire entrer des voitures dans le modèle et d'un port *out* permettant de récupérer les voitures sortant du modèle (Figure 8.6).

FIGURE 8.6 – Représentation schématique du modèle m_3 à une voie

Bilan et multi-modèle

Le tableau 8.1 fait un résumé des trois modèles en mettant en lumière les différents niveaux de représentation et les entrées et sorties utilisées dans le multi-modèle. La Figure 8.7 présente le multi-modèle et les connexions réalisées.

Chaque modèle utilise une représentation des données différentes, et les ports d'entrée et de sortie connectés au sein du multi-modèle ne sont pas directement compatibles. Pour résoudre ce problème MECSYCO propose d'utiliser des opérations de transformation de données :

1. Le débit de sortie de m_1 à chaque pas de temps est transformé en voitures atomiques avant d'être injecté dans m_2.
2. Les voitures sortant de m_2 sont transformées de sorte à pouvoir être injectées dans m_3 pour créer des agents.
3. L'état du tronçon m_3 est utilisé pour calculer un débit de sortie qui est ensuite injecté dans m_1.

Au niveau de la représentation du temps, le modèle NetLogo m_3 n'est pas directement compatible avec les deux autres modèles. Dans [Camus, 2015], le problème est résolu via des opérations de transformation du temps en entrée et en sortie de m_3, nous choisissons de conserver cette approche ici pour montrer l'utilisation des opérations temporelles. Avec le développement du nouvel artéfact de modèle NetLogo (voir section 7.1, une autre solution aurait été de choisir directement, dans l'artéfact de modèle, la signification d'un pas de temps NetLogo pour le reste de la co-simulation.

Il faut cependant garder à l'esprit que, même si les opérations de transformation permettent d'assurer la communication entre les modèles pour lancer la co-simulation, **il n'est pas possible de rendre complètement compatibles deux modèles utilisant des hypothèses et des simplifications contradictoires** [Tolk et al., 2012][Hofmann, 2005][Pennock and Rouse, 2014]. Dans notre cas, les ports *inDensity* et *outDensity* du modèle m_1 ne sont pas utilisés pour cette raison, il n'est pas possible d'obtenir la densité de voiture à l'entrée des tronçons des modèles m_2 et m_3. Nous faisons donc une hypothèse supplémentaire, nous considérons que le trafic est fluide à

Modèle	Représentation des voitures	Entrées	Sorties	Échelle de temps
Modèle Cellulaire Équationnel Java ad-hoc	Macroscopique, flux de voitures	Reçoit un flux de voitures en véhicule/km en entrée associé à une vitesse	Retourne un débit de sortie en véhicule/km associé à une vitesse	*heure*
Modèle évènementiel Java ad-hoc	Microscopique Voitures vues comme des évènements	Reçoit un nombre entier de voitures en entrée	Renvoie le nombre de voitures sortant du modèle et leur vitesse moyenne	*heure*
Modèle multi-agent NetLogo	Microscopique Voitures vues comme des agents autonomes	Reçoit une liste de voitures à intégrer	Retourne la liste des voitures sortantes	$\emptyset$

TABLE 8.1 – Résumé des trois modèles

l'entrée des tronçons à trois et deux voies (modèles m_1 et m_2). Les concentrations de voitures sont alors négligeables ce qui justifie la non utilisation des ports *inDensity* et *outDensity*. Notons que cette nouvelle simplification entraine une légère perte de précision.

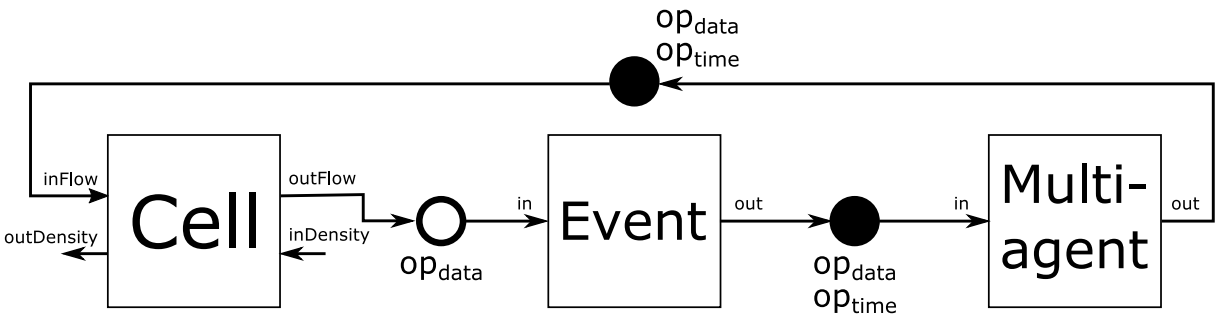


FIGURE 8.7 – Représentation schématique du multi-modèle d'autoroute hybride avec les opérations.

Jusqu'à présent, les trois modèles utilisés pour cet exemple étaient manipulés sous forme de code Java. C'étaient tous des artéfacts de modèles implémentés de manière ad-hoc, et la co-simulation était directement décrite en Java. Les modèles atomiques n'étaient pas décrits ce qui rendait leur réutilisation difficile, e.g. il était nécessaire de lire directement le code Java pour connaitre leurs interfaces.

Nous détaillons dans la suite comment nous avons repris l'exemple de l'autoroute hybride en suivant notre démarche et en employant notre environnement DSL.

8.2.2 Intégration des modèles atomiques

Nous reprenons ici des modèles ayant déjà été intégrés dans MECSYCO. Le travail d'encapsulation technique et formelle a donc déjà été réalisé. Cependant, dû a une spécification insuffisante des m-artéfacts (voir explications en 5.2.4), des modifications ont dues être opérées sur les classes existantes pour les intégrer dans notre démarche et dans l'environnement DSL.

Au niveau technique

La première modification concerne les types de données échangées. Pour assurer la compatibilité entre tous les modèles intégrés dans MECSYCO via des artéfacts de modèle, nous avons restreints les types accessibles. Dans nos langages dédiés, nous avons choisi de permettre la description des types simples (réel, entier, chaine de caractères et booléen) et des types complexes (des tuples de 2 à 5 entrées simples ou des tableaux de tuples). L'exemple existant utilisait des types spécifiquement définis qu'il a donc fallu convertir pour assurer la compatibilité avec de futurs modèles.

La deuxième modification concernait les modèles eux-même. Nous avons utilisé le nouvel artéfact de modèle NetLogo pour le modèle m_3 plutôt qu'un artéfact ad-hoc. Ensuite, nous avons adapté les constructeurs disponibles pour les artéfacts de modèles ad-hoc qui représentaient les modèles m_1 et m_2. Cette contrainte sur les constructeurs nous permet par la suite de charger à chaud les artéfacts de modèle écrits en Java, nous faisons de même avec les opérations. **Nous montrons ici que nous pouvons réutiliser les artéfacts de modèle et les opérations existantes en les adaptant pour correspondre à nos spécifications**.

Au niveau description

Il est nécessaire de décrire l'interface des modèles atomiques pour compléter l'intégration. Nous avons donc utilisé le langage *m2xml* pour décrire les interfaces des modèles m_1 et m_2 et le langage *nl2xml* spécifique à NetLogo pour décrire le modèle m_3 (voir annexes B.1.1, B.1.2 B.1.3).

8.2.3 Multi-modélisation

La deuxième étape consiste à décrire le multi-modèle. Celui-ci est constitué des trois modèles précédents qui sont référencés grâce à leurs documents de description XML.

C'est à cette étape que nous décrivons les liens entre les sous-modèles et que nous spécifions les opérations à appliquer. Le fichier *mm2xml* correspondant est disponible en annexe B.1.4.

8.2.4 Expérimentation

La dernière étape consiste alors à décrire l'expérience que nous souhaitons réaliser avec le multi-modèle. Nous décrivons donc l'ensemble des paramètres que nous choisissons et le système d'observation que nous souhaitons utiliser (voir annexe B.1.5). Nous pouvons choisir un système d'observation standard ou en utiliser un spécifiquement développé pour l'exemple (dans ce cas, l'artéfact d'observation utilisé sera là encore chargé à chaud).

La Figure 8.8 présente l'évolution du nombre de voitures dans les trois modèles lors de la co-simulation. Les modifications que nous avons apportées n'impactent pas les comportements des modèles mais seulement la manière de les manipuler, nous obtenons donc les mêmes évolutions que dans [Camus, 2015].

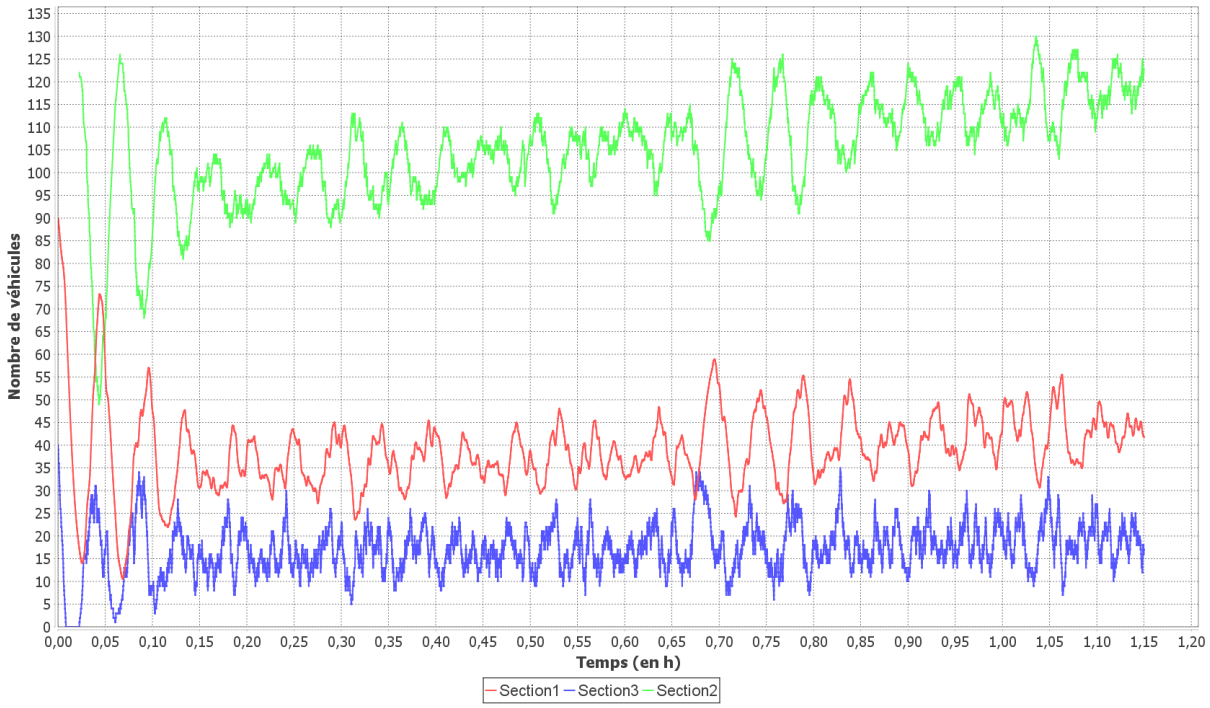


FIGURE 8.8 – Évolution du nombre de véhicules dans les trois tronçons au cours du temps (en heure).

8.2.5 Bilan

Le multi-modèle d'autoroute hybride est intéressant car il rassemble toutes les d'hétérogénéités gérées par MECSYCO (logicielle, formelle, de représentation des données et du temps). La reprise de cet exemple montre que nous conservons bien les propriétés initiales de MECSYCO concernant la gestion des hétérogénéités : nous pouvons reprendre les m-artéfacts et les opérations utilisés dans les anciens exemples de MECSYCO pour les utiliser au sein de notre démarche et de notre environnement DSL. Nous illustrons aussi dans cet exemple l'utilisation du nouvel m-artéfact NetLogo.

Plusieurs autres points sont à noter :

— Augmentation des contraintes et extensibilité : L'utilisation de l'environnement impose des contraintes supplémentaires au niveau du code des artéfacts de modèle, des artéfacts d'observation et des opérations. Cependant, en respectant ces contraintes il est possible d'utiliser dans notre environnement n'importe quel nouvel artéfact de modèle ou opération évènementielle ce qui assure l'extensibilité.

— Étapes supplémentaires, réutilisation et modularité : Notre démarche est décomposée en trois parties allant des modifications au niveau du programme jusqu'à la description d'expérience. Cet effort supplémentaire (avant le multi-modèle et la co-simulation étaient écrits en même temps en Java) est cependant payant puisqu'il simplifie la création du multi-modèle et facilite la réutilisation des modèles et leur modularité. La description des modèles atomiques, en documentant leur interface, permet de les réutiliser pour réaliser d'autres expériences. Pour échanger un sous-modèle par un autre, à interface identique, il suffit de pointer vers une autre description XML au niveau du multi-modèle.

— Simplicité et accessibilité : L'utilisation des DSL permet de se focaliser sur les aspects

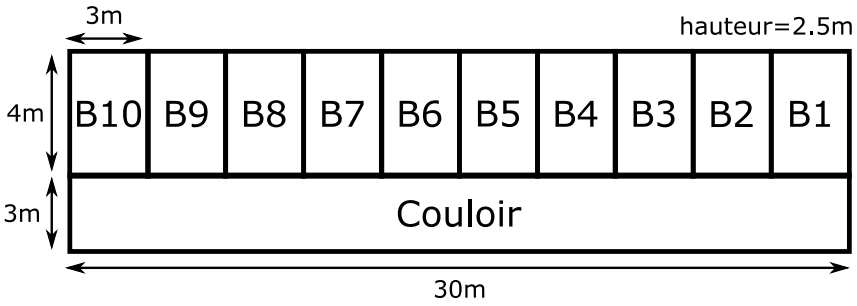


FIGURE 8.9 – Schéma du bâtiment que nous modélisons

essentiels de la modélisation et de la co-simulation sans s'attarder sur le code Java. Cela permet l'accès à des non-informaticiens. À titre indicatif, la réalisation du multi-modèle via le DSL occupe environ 60 lignes tandis que la description du multi-modèle (et de la co-simulation) en Java demande environ 150 lignes.

8.3 Un système de chauffage intelligent

La seconde expérience est un modèle d'évolution de la température et de la consommation énergétique d'un bâtiment. Nous nous intéressons ici à la climatisation d'un bâtiment en été. Cet exemple s'inspire de plusieurs articles utilisant MECSYCO [Camus et al., 2017][Gilpin et al., 2014]. Bien que naïf, il permet d'illustrer encore une fois la gestion des hétérogénéités en utilisant des modèles issus de différents logiciels. Son intérêt principal est de montrer la construction hiérarchique du multi-modèle du bâtiment.

8.3.1 Contexte

Nous souhaitons modéliser un bâtiment composé de 10 bureaux reliés entre eux par un couloir suivant le schéma suivant (Figure 8.9). Chaque pièce est équipée d'un système de régulation de la température, il y a des transferts thermiques entre les pièces et avec la température extérieure via les murs.

L'exemple est construit autour de 4 composants atomiques :

— Un modèle de pièce qui prend en entrée deux flux de chaleur (le premier issu des murs, le deuxième issu d'un système de régulation thermique) et qui retourne la température intérieure. Le modèle est implémenté sous forme de FMU.
— Un modèle de mur, prenant en entrée les températures sur chacune de ses faces et retournant le flux thermique résultant. C'est aussi une FMU.
— Un relevé de la température extérieure à Vandœuvre sur la période Juillet-Août 2018 sous forme de base de données SQL. Ce relevé nous est fourni par la résidence APHEEN [19] (Association Pour l'Hébergement des Étudiants et Enseignants de Nancy) dans le cadre d'un projet commun concernant le confort thermique dans leurs appartements.
— Un modèle de climatiseur, prenant en entrée une température à l'intérieur d'une pièce et retournant un flux de chaleur. C'est un modèle ad-hoc écrit en Java.

Nous ne nous focalisons pas ici sur la complexité atomique des composants (prise en compte du rayonnement, des forces de convection...), mais sur la complexité structurelle de construction

19. https://www.apheen.fr/

du multi-modèle. Nous avons donc des modèles aux comportements simples. Notre objectif est d'illustrer l'utilisation de modèles/composants hétérogènes et la construction hiérarchique.

1. Création d'un premier multi-modèle représentant une salle avec chauffage.
2. Création d'un multi-modèle représentant un bâtiment composé d'un couloir alimentant une dizaine de bureaux, chacun équipé d'une climatisation.

8.3.2 Intégration des modèles atomiques

La première étape de création des modèles est effectuée en dehors de MECSYCO. Du point de vue de l'intergiciel, deux étapes sont nécessaires à leur intégration : la création d'un m-artéfact particulier au logiciel intégré et à son simulateur, et la description des modèles eux-mêmes.

Modèle de pièce et modèle de mur

Ces deux modèles FMU CS sont écrits dans Modelica puis exportés via JModelica [Andersson, 2017]. Nous disposons déjà d'un artéfact de modèle pour les FMU de type co-simulation et grâce au standard FMI, les modèles possèdent déjà un document de description. Ces descriptions se rapprochent de celles que nous avons définies (documentation des paramètres, des ports d'entrée et des ports de sortie), nous nous donnons donc les moyens de les exploiter directement.

Modèle météo

Le modèle de météo est une base de données SQL contenant des relevés de température toutes les 20 secondes environ et allant du 4 Juillet au 20 Août. Nous sommes donc dans le même cas qu'en section 7.2, c'est un modèle source. Des bibliothèques Java existent pour intégrer des bases de données SQL dans des programmes. Le développement d'un artéfact de modèle adapté à notre besoin est donc grandement facilité.

La seule contrainte consistait à faire le lien entre les dates des relevés (représentées en secondes, minutes, heures, jours, mois, années) et le temps simulé (représenté par une valeur réelle). Nous choisissons d'exprimer le temps simulé en seconde, et nous mettons en paramètre le choix de date de début de simulation. Notre m-artéfact envoie des évènements dont l'étiquette temporelle correspond au nombre de secondes écoulées depuis la date initiale choisie. Cette valeur réelle est calculée en faisant la différence entre la date initiale en paramètre et la date de l'évènement contenue dans la base de données.

À l'échelle où nous effectuons la co-simulation, les systèmes thermiques ont beaucoup d'inertie et ne nécessitent pas l'utilisation d'un pas de calcul de l'ordre de la seconde. Pour limiter le nombre d'évènements à envoyer, nous ajoutons un pas de temps en paramètre de simulation, celui-ci correspond au délai minimum entre deux évènements. Ce pas de temps nous permet de contrôler, et donc d'augmenter, le temps entre deux envois d'évènements de notre modèle météo.

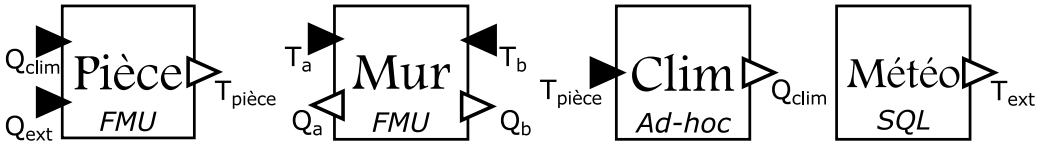
Nous documentons ensuite l'interface de cette source de données pour pouvoir la réutiliser dans notre multi-modèle. Nous utilisons pour cela le langage *m2xml* (voir annexe B.2.1). Notons que cet artéfact de modèle ne fait pas partie des artéfacts pris en compte dans l'environnement, il sera chargé à chaud lors de la co-simulation.

Modèle de climatiseur

Le dernier modèle atomique est le modèle du climatiseur. C'est un modèle ad-hoc écrit en Java que nous adaptons pour correspondre à l'interface des artéfacts de modèle MECSYCO.

Nous utilisons là encore le langage *m2xml* pour le décrire et ce modèle sera lui aussi chargé à chaud (voir annexe B.2.2).

La Figure 8.10 présente les différents modèles atomiques utilisés sous forme de schéma.



(a) Modèle de pièce (b) Modèle de mur (c) Modèle de climatiseur (d) Modèle source météo

FIGURE 8.10 – Ensemble des modèles atomiques utilisés pour l'exemple.

8.3.3 Multi-modélisation

Après avoir intégré et documenté les modèles atomiques, nous construisons hiérarchiquement notre modèle d'étage.

Multi-modèle de pièce climatisée

Le premier multi-modèle que nous créons représente une pièce climatisée, il est composé d'un modèle de pièce et d'un modèle de climatiseur (voir Figure 8.11). Il nous servira de brique de base pour la construction d'un multi-modèle d'étage contenant plusieurs salles climatisées adjacentes.

L'étape de création du multi-modèle se réalise entièrement via le langage *mm2xml*, le fichier se trouve en annexe B.2.3.

Nous testons ce premier multi-modèle. Pour cela, nous le connectons au modèle de température extérieure au sein d'un nouveau multi-modèle et nous décrivons une co-simulation via le langage *cs2xml*. Nous utilisons les relevés de températures entre le 2 et le 4 Août 2018, les résultats sont présentés Figure 8.12.

Remarques : Nous avons aussi la possibilité de tester un multi-modèle sans avoir à décrire une expérience. Nous pouvons lancer une co-simulation directement à partir d'une description de modèle. Nous utilisons alors les paramètres par défaut, et les mécanismes d'observation sont limités à l'affichage de courbes.

Multi-modèle d'étage climatisé

À partir du modèle précédent et du modèle du mur, nous réalisons un multi-modèle représentant l'étage de la Figure 8.9, le fichier *mm2xml* correspondant est disponible en annexe B.2.4. La Figure 8.13 présente une vision du multi-modèle, avec l'ensemble des sous-modèles et des

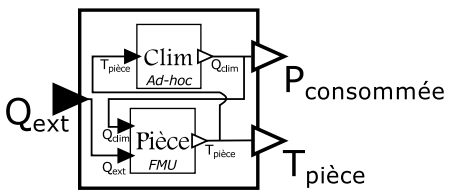


FIGURE 8.11 – Schéma présentant le multi-modèle de pièce climatisée.

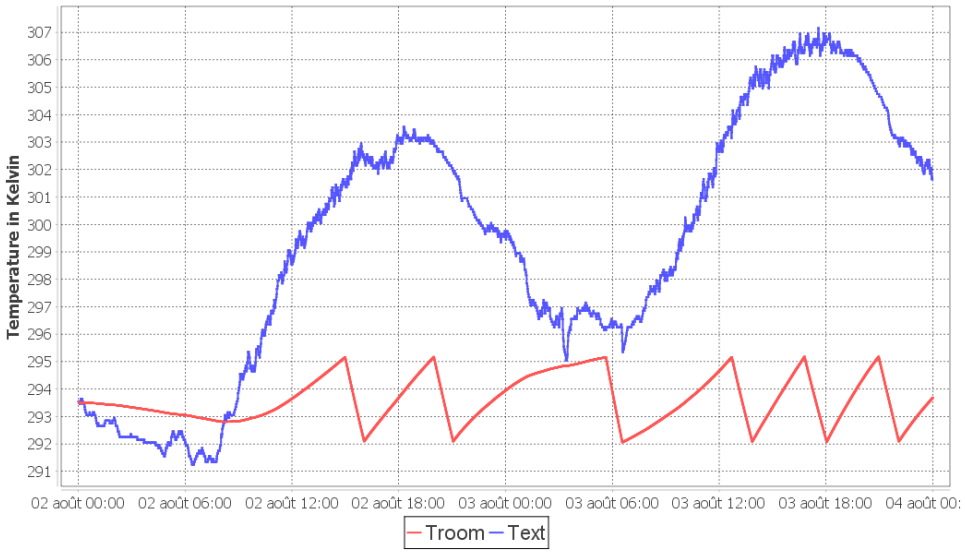


FIGURE 8.12 – Température du multi-modèle de pièce avec climatiseur couplé avec les relevés de température entre le 2 et le 4 Août 2018.

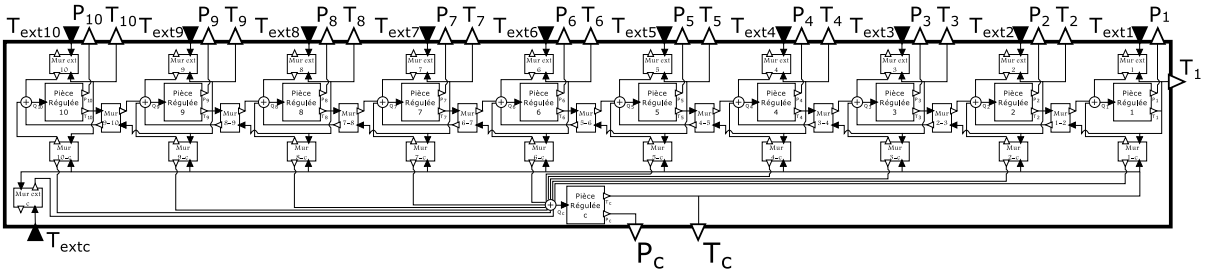


FIGURE 8.13 – Schéma présentant le multi-modèle d'étage.

connections. Il est composé de 11 salles régulées (10 bureaux et le couloir) et de 30 murs qui font les liens thermiques entre les salles.

Remarques

Dans ce multi-modèle, les modèles de pièces sont connectés entre eux via des modèles de murs. Pour chaque pièce, l'influence des pièces adjacentes est donnée par la somme des flux de chaleur envoyés par les murs. Il faut donc agréger ces flux de chaleur, les sommer, avant des les envoyer dans les modèles de pièces. Nous utilisons des artéfacts multiplexeurs pour effectuer cette somme.

Le nombre d'entités et de relations rend ce multi-modèle difficile à lire et à écrire, que ce soit via une modélisation graphique (voir Figure 8.13) ou via une description à l'aide du langage *mm2xml* (voir annexe B.2.4).

Nous avons vu que la gestion de la complexité passait par l'organisation des composants. Dans notre cas, une autre piste de simplification consisterait à définir la notion de connexion entre pièces, i.e. définir qu'une connexion entre deux pièces corresponde à l'utilisation d'un mur et que tous les flux de chaleur en provenance des murs soient sommés. Si nous poursuivons cette idée, cela reviendrait à créer des langages dédiés à des domaines d'application particuliers. Nos DSL permettent de guider la démarche de multi-modélisation et co-simulation, simplifient le travail de multi-modélisation en permettant la hiérarchisation, mais l'utilisation de langage dédiés à un cadre applicatif peut permettre d'aller plus loin en matière de simplification et de vérification.

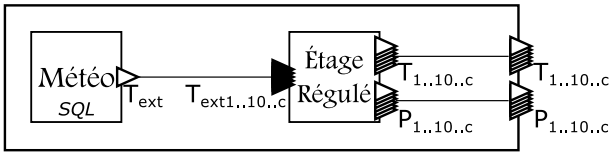


FIGURE 8.14 – Schéma présentant le multi-modèle d'étage climatisé couplé au modèle météo.

Avant de passer à l'expérimentation, il convient de connecter ce multi-modèle d'étage climatisé avec les données météo. Nous réalisons cela dans un dernier multi-modèle (Figure 8.14), le fichier *mm2xml* est disponible en annexe B.2.5.

8.3.4 Expérimentation

La dernière étape consiste à décrire la co-simulation que nous souhaitons effectuer sur le multi-modèle d'étage climatisé couplé à nos données météo. Le fichier *cs2xml* correspondant est disponible en annexe B.2.6.

Notons que, dans la description de l'expérience, nous utilisons un nouvel artéfact d'observation permettant d'afficher les courbes avec en abscisses des dates plutôt que des valeurs réelles. Nous chargeons aussi à chaud les opérations de transformations pour avoir les températures de la base de données (en Celsius) et les températures du multi-modèle d'étage (en Kelvin) dans la même unité.

Les Figures 8.15 et 8.16 présentent les résultats obtenus où nous pouvons observer la réponse de nos salles climatisées aux fortes chaleurs de l'été 2018. Nous n'affichons que les courbes des températures des bureaux 1 à 5, la symétrie du multi-modèle entrainant des résultats identiques dans les bureaux suivants.

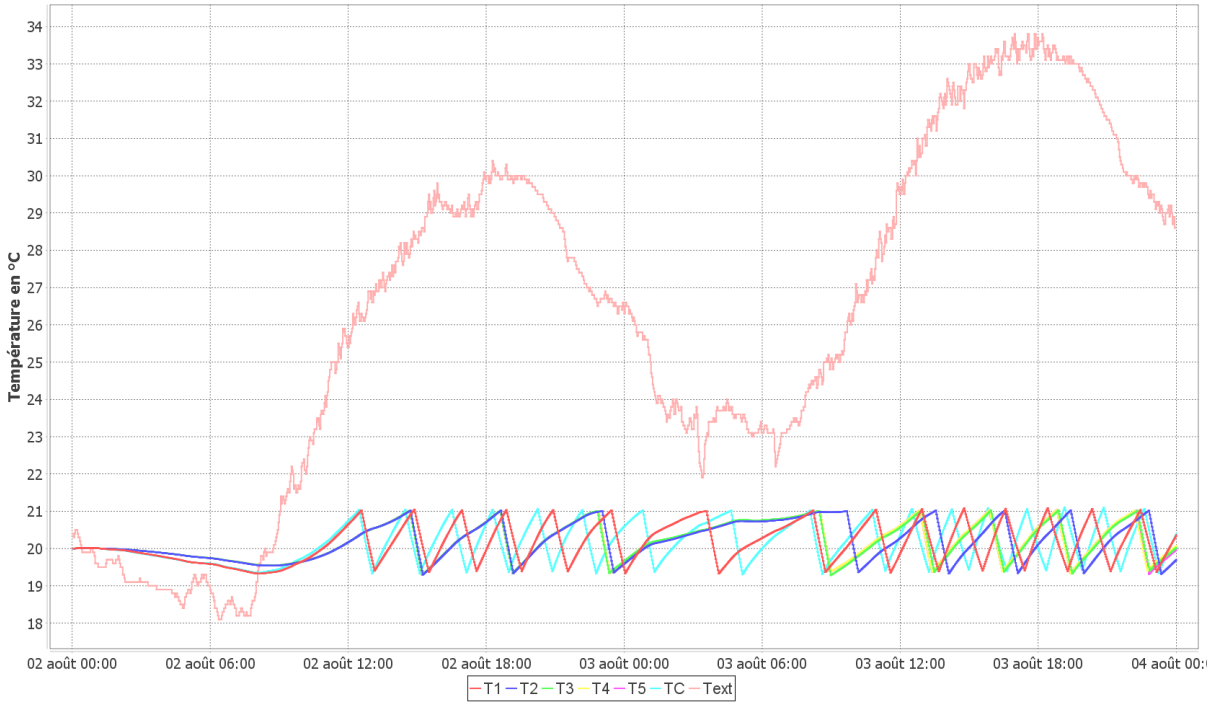


FIGURE 8.15 – Températures du multi-modèle d'étage entre le 2 et le 4 Août 2018.

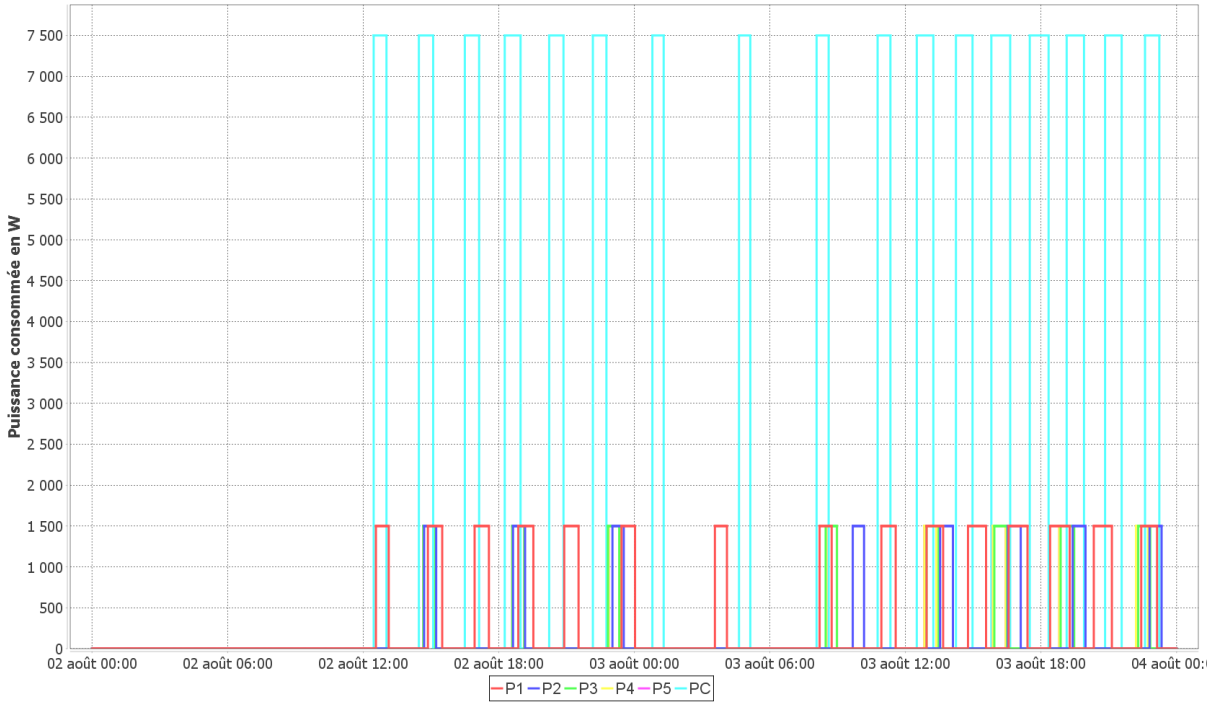


FIGURE 8.16 – Puissances instantanées consommées dans chaque pièce par nos climatiseurs.

8.3.5 Bilan

La Figure 8.17 résume les 4 étapes d'intégration et de construction de multi-modèles qui ont été nécessaires à la réalisation de l'exemple.

La première étape concerne la partie intégration de notre démarche tandis que les étapes suivantes présentent la construction hiérarchique du multi-modèle sur lequel est effectuée notre expérience.

L'étape 1 présente de nouveau la conservation de la capacité d'intégration et de gestion des hétérogénéités dans MECSYCO (au niveau des logiciels et des simulateurs).

L'étape 2 nous montre la construction d'un multi-modèle composé de sous-modèles hétérogènes. Celui-ci est directement réutilisé comme composant à l'étape 3 pour construire le multi-modèle d'étage. Cette étape 3 met en lumière une piste d'amélioration concernant l'utilisation de DSL dédiés à des domaines applicatifs pour limiter la complexité d'écriture lorsqu'il y a de nombreuses entités sur un même niveau et que les relations entre elles respectent des schémas identiques.

Enfin, pour la dernière étape nous réutilisons directement notre modèle d'étage régulé dont la complexité interne n'apparait plus à ce niveau. Les différentes étapes de construction du multi-modèle final montrent notre capacité à hiérarchiser le développement du modèle en plusieurs composants réutilisables. Il faut cependant garder en tête que cette hiérarchie de description ne reflète pas ce qui ce passe pendant la co-simulation. Nous n'avons pas modifié l'algorithme de co-simulation de MECSYCO, l'exécution s'effectue "à plat" et de manière décentralisée peu importe comment les multi-modèles sont organisés (et donc le choix de hiérarchie n'influence pas les résultats).

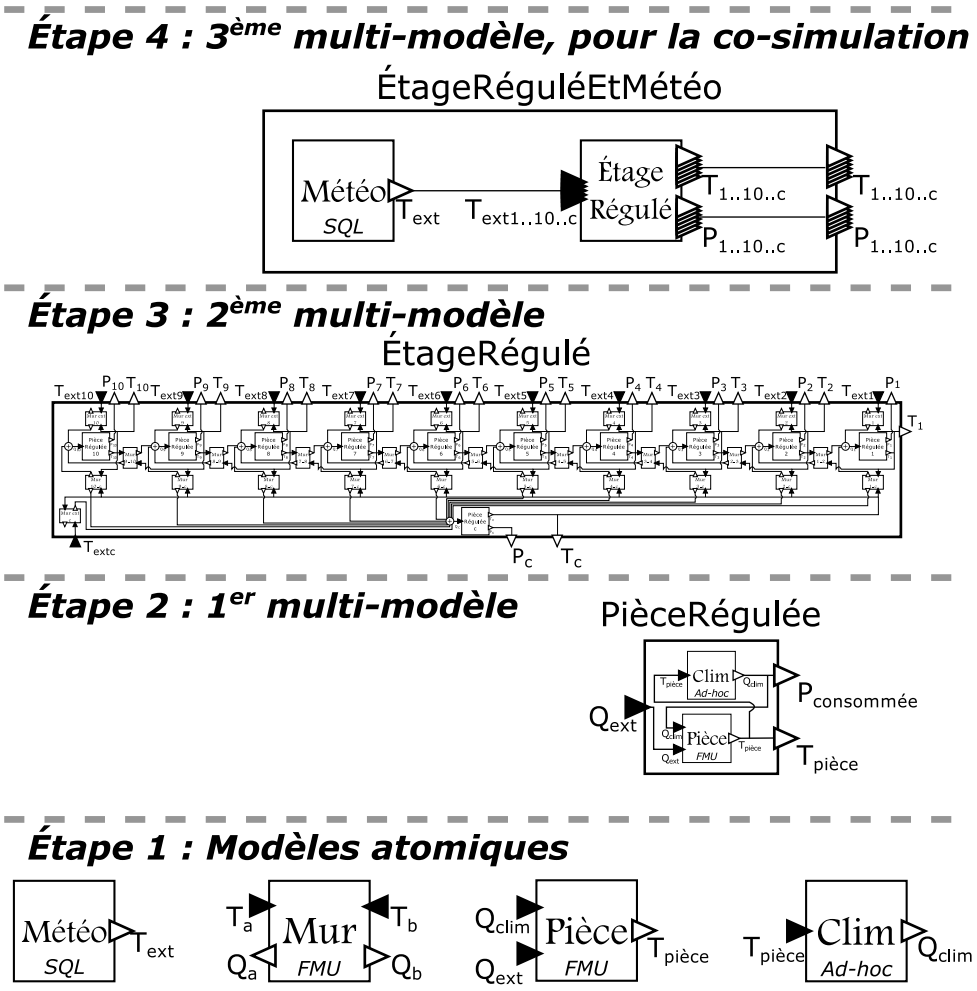


FIGURE 8.17 – Synthèse des étapes de construction jusqu'à l'expérimentation.

8.4 Conclusion

Nous avons présenté dans ce chapitre deux preuves de concept pour tester et évaluer nos travaux des chapitres 6 et 7. Nous développons ces exemples en suivant la démarche que nous proposons et en utilisant l'environnement DSL qui l'accompagne. Nous les évaluons ensuite suivant quatre aspects : la conservation des propriétés de MECSYCO (dont la gestion des hétérogénéités la modularité et l'algorithme décentralisé), la capacité à hiérarchiser, l'extensibilité et la facilité d'utilisation.

Le premier aspect qui nous intéresse, la conservation des propriétés, doit permettre de vérifier que notre apport n'ajoute pas de limitations dans MECSYCO. C'est la première expérience sur l'autoroute hybride qui nous permet de répondre à cette question. Elle montre la possibilité de reprendre, dans notre environnement DSL et en suivant notre démarche, un exemple complet illustrant l'ensemble des possibilités de MECSYCO en terme de gestion des hétérogénéités. Un deuxième aspect important de MECSYCO est l'utilisation d'un algorithme de co-simulation décentralisée et la possibilité de distribuer les calculs. Ce deuxième aspect est toujours présent de part la conservation de l'algorithme de co-simulation. Il reste cependant à implémenter un mécanisme de déploiement automatique en partant des descriptions pour exploiter pleinement

cette capacité. Enfin, la modularité est conservée, et il est très facile de remplacer un modèle par un autre puisque chaque modèle est vu dans MECSYCO comme un modèle DEVS. À partir du moment où ils fournissent la même interface (ports d'entrée/sortie) il suffit de changer le chemin d'accès au modèle dans le multi-modèle pour tester différentes configurations. Une voie d'amélioration de MECSYCO serait de traiter les sous-modèles comme des paramètres pour faciliter ce changement.

Le deuxième aspect, la capacité à hiérarchiser la construction de multi-modèle qui est fondamentale pour l'étude de système complexe, est démontré via la seconde expérience. Pour construire le multi-modèle d'étage climatisé soumis à une température extérieure, nous développons successivement un multi-modèle de pièces climatisées et un multi-modèle d'étage climatisé. Ces deux modèles sont réutilisés aux niveaux suivants comme si c'étaient des modèles atomiques. De plus, de part l'instanciation "à plat" de l'algorithme de co-simulation, la manière de hiérarchiser le multi-modèle n'a pas d'importance sur le résultat (grâce aux propriétés de DEVS, cela serait aussi le cas en utilisant les algorithmes DEVS classiques hiérarchisés et centralisés).

La propriété d'extensibilité est illustrée dans les deux exemples. Dans le premier, par le chargement à chaud des deux modèles implémentés en Java, et des opérations sur les évènements résolvant les problèmes d'hétérogénéités. Dans la deuxième expérience, par le chargement à chaud :

1. du modèle ad-hoc Java du climatiseur,
2. d'une opération permettant de transformer des degrés Celsius en degrés Kelvin et inversement, et
3. d'un nouvel artéfact d'observation.

Nous montrons ainsi que sans modification ni du cœur MECSYCO, ni de l'environnement DSL, il est possible de travailler avec de nouveaux types de simulateur, de gérer les hétérogénéités par de nouveaux types d'opération, et d'ajouter de nouveaux rendus graphiques pour nos besoins.

Enfin, l'amélioration de l'utilisabilité de MECSYCO est l'aspect le plus subjectif à évaluer. Nous partons du principe que la séparation de l'utilisation de MECSYCO en trois étapes bien définies (description des modèles atomiques, des multi-modèles puis des expériences) supportées par des DSL, facilite l'utilisation de MECSYCO en évitant aux non-informaticiens d'utiliser Java. De plus, nous avons plusieurs retours tendant à valider que notre approche simplifie l'utilisation de MECSYCO. En effet, notre environnement a été utilisé dans le cadre d'un TP d'introduction à la co-simulation à l'ENSEM [20] (École Nationale Supérieure d'Électricité et de Mécanique) et dans le cadre de projets de fin d'études. Dans chacun de ces cas, les retours utilisateurs étaient positifs et nous ont aidé à améliorer la plateforme.

En perspective, plusieurs pistes d'amélioration sont envisagées pour simplifier l'utilisation de MECSYCO ou augmenter l'expressivité des descriptions. Un premier élément serait d'augmenter le nombre de vérifications dans l'interface Eclipse, i.e. signaler automatiquement des erreurs comme le non respect de l'interface d'un sous-modèle. Un point d'extension serait aussi de permettre l'expression de relation entre les paramètres des modèles. Par exemple, dans le modèle évènementiel de notre expérience d'autoroute hybride, le délai minimum de propagation dépend de la longueur de la route considérée et de la vitesse maximale des voitures. Il serait intéressant de pouvoir exprimer directement cette relation.

20. École d'ingénieur rattachée à l'Université de Lorraine et située à Vandœuvre-lès-Nancy, informations sur : https://ensem.univ-lorraine.fr/

9

Conclusion

9.1 Contexte initial

Le contexte global de ce travail de thèse est la M&S de systèmes complexes, i.e. de systèmes composés d'un grand nombre d'entités en interaction. La multi-modélisation et la co-simulation sont deux approches prometteuses, mais elles requièrent de développer de nouvelles méthodes et outils pour pouvoir être mises en pratique.

C'est dans ce contexte que nous avons poursuivi le travail autour de l'intergiciel de co-simulation MECSYCO. Celui-ci fournit une approche et un environnement de co-simulation permettant d'intégrer différents outils de M&S. L'intergiciel repose sur trois niveaux de réflexion, une encapsulation logicielle, une encapsulation formelle basée sur DEVS et le méta-modèle AA4MM qui permettent la gestion des hétérogénéités à tous les niveaux. Cependant, MECSYCO n'est pas associé à une démarche de multi-modélisation hiérarchique et de co-simulation. La plateforme ne permet pas une construction incrémentale de ses multi-modèles et ne fournit pas d'environnement de multi-modélisation.

Nous nous sommes donc intéressés dans cette thèse à la démarche de multi-modélisation et co-simulation appliquée à MECSYCO en identifiant les étapes nécessaires à la réalisation d'une co-simulation. Nous proposons aussi d'autres apports plus spécifiques en lien avec les étapes identifiées et qui nous ont permis d'enrichir la plateforme.

9.2 Contributions

La contribution principale de cette thèse a consisté à associer une démarche de multi-modélisation hiérarchique et de co-simulation à MECSYCO. Cette démarche permet la réutilisation des multi-modèles MECSYCO et leur composition pour en former d'autres, autorisant ainsi une construction hiérarchique des multi-modèles. Cet apport est concrétisé au sein d'un environnement de multi-modélisation et de co-simulation pour MECSYCO basé sur l'utilisation de DSL. La contribution est organisée autour de 3 axes :

1. La décomposition de l'activité de multi-modélisation et de co-simulation en cinq étapes distinctes : la conception du multi-modèle, le développement des modèles atomiques (se faisant dans des logiciels dédiés), leur intégration, la description des multi-modèles et la définition des co-simulations.
2. La spécification des informations nécessaires pour décrire et manipuler les entités produites (modèles atomiques intégrés dans MECSYCO, multi-modèles et co-simulations).

Ces spécifications se retrouvent dans la structure de fichiers de description qui permettent de réutiliser les éléments décrits et de réaliser des analyses et des vérifications.

3. Le développement de plusieurs DSL pour chaque étape de description. Nous sortons ainsi du cadre de la programmation pour se concentrer sur les problèmes de multi-modélisation et co-simulation.

Notre démarche demande à l'utilisateur de préciser l'interface des modèles qu'il compte co-simuler (i.e. des modèles issus de logiciels de M&S déjà intégrés dans MECSYCO), puis de décrire la structure du multi-modèle et son interface, et enfin de spécifier la ou les co-simulations qu'il souhaite effectuer. Chaque étape de la démarche est supportée par un langage dédié différent et génère une description au format XML qui peut être réutilisée. L'utilisation de l'outil XText et de l'environnement Eclipse permet de bénéficier d'un environnement d'édition complet qui effectue des vérifications sur la syntaxe du langage, restreignant les erreurs possibles de l'utilisateur directement lorsque celui-ci écrit.

Ce travail autour de la démarche de multi-modélisation et co-simulation s'accompagne de plusieurs travaux ayant trait au cycle de M&S et qui ont servi à enrichir l'intergiciel MECSYCO :

— Nous nous sommes intéressés à la question de l'intégration de nouveaux simulateurs dans MECSYCO en prenant l'exemple de NetLogo. Ce travail a conduit au développement d'un artéfact de modèle NetLogo plus générique (i.e. permettant d'intégrer plusieurs modèles NetLogo), et à la définition d'un DSL facilitant l'intégration de nouveaux modèles NetLogo dans MECSYCO.
— Au niveau de la multi-modélisation et des connexions entre modèles, nous avons travaillé sur la possibilité de liaisons complexes et le besoin d'agréger des données issus de plusieurs modèles. Nous avons ainsi développé un nouvel artéfact à ajouter au méta-modèle AA4MM, l'artéfact multiplexeur. Il apporte un mécanisme de gestion des évènements simultanés arrivant sur un même port d'entrée et permet d'agréger des évènements venant de plusieurs modèles.
— Nous avons exploré l'utilisation de modèles sources pour rejouer les co-simulations à partir de log, et/ou pour soumettre les modèles à une série d'évènements pré-définis (possiblement des données réelles) pour tester leur fonctionnement et leur comportement.
— Des outils de mesure de performance ont été définis pour récolter des informations lors des co-simulations. Ils forment une source d'information complémentaire pour analyser les co-simulations et la dynamique des modèles.

Toutes ces contributions ont été utilisées au sein de deux exemples. Le premier, repris de [Camus, 2015], montre que nous conservons les propriétés initiales de la plateforme MECSYCO. Le second inspiré d'un problème de régulation thermique de bâtiment intelligent illustre les possibilités de construction hiérarchique. Tous deux nous permettent de mettre en avant la démarche, l'environnement DSL associé et nos autres apports.

9.3 Perspectives

Outre des perspectives d'ordre général sur MECSYCO (structure dynamique, *Hardware in the Loop*...) ; nous tenons ici à mentionner quelques perspectives en relation directe avec nos travaux.

Nous avons maintenant des supports (les documents de description) pour noter toutes les informations pertinentes sur un modèle ou multi-modèle et effectuer des vérifications statiques. Une première prolongation directe de nos travaux serait le développement d'une bibliothèque de modèles et multi-modèles réutilisables dans MECSYCO. Une seconde prolongation consisterait

à ajouter de nouvelles informations dans nos descriptions et à y associer des vérifications, e.g. ajouter des informations sur les unités des données échangées (Ampères, kilomètres, Joules...) et assurer la cohérence des connexions. En partant des descriptions d'expériences, nous pourrions aussi proposer un mécanisme de déploiement automatique des co-simulations sur un réseau d'ordinateurs en prenant en compte des contraintes comme la présence des logiciels, des licences ou des modèles. Ces développements basés directement sur l'environnement que nous proposons rentrent dans le cadre d'une amélioration des fonctionnalités de MECSYCO pour lui faire gagner en attractivité.

Une perspective à plus long terme consisterait à développer des langages dédiés à des domaines d'études particuliers. En effet, nos DSL sont prévus pour l'expression de multi-modèles ou de co-simulations MECSYCO dans un cadre général, mais il serait possible de gagner encore en expressivité en développant des langages dédiés à des études précises comme la thermique des bâtiments ou la simulation de micro-grids. L'objectif serait de fournir des langages adaptés aux experts pour exprimer des multi-modèles liés à leurs domaines, tout en générant des spécifications compatibles pour effectuer des co-simulations MECSYCO ou pour construire incrémentalement des multi-modèles pluridisciplinaires. Il faudrait alors s'intéresser à la démarche de conception de multi-modèles interdisciplinaires, un aspect non développé dans nos travaux. Nous pourrions pour cela nous inspirer de démarche de conception de modèles de systèmes complexes telle qu'ASPECS [Cossentino et al., 2010] et d'approches basées sur les ontologies telle MIMOSA [Müller, 2010].

Enfin, nous avons dorénavant dans MECSYCO une interface générique basée sur DEVS et des documents de description précisant l'interface de nos modèles. Il est possible d'envisager à long terme de proposer des composants MECSYCO facilement intégrables dans d'autres logiciels de M&S, permettant à ceux-ci d'utiliser des modèles intégrés dans MECSYCO sans refaire le travail d'encapsulation. Plus généralement, de nombreux travaux ont souligné l'intérêt d'un processus de standardisation basé sur DEVS [Wainer et al., 2010b][Wainer et al., 2010a][Wainer et al., 2010c] pour rendre les outils DEVS interopérables et permettre l'échange de modèles. Le fort engouement pour la norme FMI (adaptées pour les systèmes continus ou hybrides) nous interroge sur l'intérêt de développer le même type d'approche composant basée sur le formalisme DEVS pour bénéficier de ses propriétés (comme nous l'avons suggéré dans [Paris et al., 2018a]).

A

Exemple de Lorenz sur MECSYCO

Nous donnons ci-dessous un exemple de code MECSYCO pour co-simuler trois FMU représentant le système dynamique de Lorenz.

```
public class LorenzExampleFMU {

  public static void main(String[] args)
  {
    //parameters of the simulation
    double stopTime=10.0, timeStep=0.01, startTime=0;

    //parameters and initial input values of the X model
    Map<String, Double> initVarsX = new HashMap<>();
    initVarsX.put("y", 1.);
    initVarsX.put("x", 1.);
    initVarsX.put("a", 10.);
    // create the model artifact, the agent and link them, for X
    FMIModelArtifact X = new FMIModelArtifact("X", "My Models/fmu/LorenzX_OM_Cs2_x64.fmu",
        stopTime, timeStep, startTime, initVarsX);
    EventMAgent XAgent = new EventMAgent("X", stopTime);
    XAgent.setModelArtifact(X);

    //parameters and initial input values of the Y model
    Map<String, Double> initVarsY = new HashMap<>();
    initVarsY.put("x",1.);
    initVarsY.put("z", 4.);
    initVarsY.put("b", 28.);
    // create the model artifact, the agent and link them, for Y
    FMIModelArtifact Y = new FMIModelArtifact("Y", "My Models/fmu/LorenzY_OM_Cs2_x64.fmu",
        stopTime, timeStep, startTime, initVarsY);
    EventMAgent YAgent = new EventMAgent("Y", stopTime);
    YAgent.setModelArtifact(Y);

    //parameters and initial input values of the Z model
    Map<String, Double> initVarsZ = new HashMap<>();
    initVarsZ.put("x", 1.);
    initVarsZ.put("y", 1.);
    initVarsZ.put("c", 2.67);
    // create the model artifact, the agent and link them, for Z
    FMIModelArtifact Z = new FMIModelArtifact("Z", "My Models/fmu/LorenzZ_OM_Cs2_x64.fmu",
        stopTime, timeStep, startTime, initVarsZ);
    EventMAgent ZAgent = new EventMAgent("Z", stopTime);
    ZAgent.setModelArtifact(Z);
```

```
    //Creation of the links between models
    CentralizedEventCouplingArtifact XOutputToY_x_x = new CentralizedEventCouplingArtifact();
    XAgent.addOutputCouplingArtifact(XOutputToY_x_x, "x");
    YAgent.addInputCouplingArtifact(XOutputToY_x_x, "x");

    CentralizedEventCouplingArtifact XOutputToZ_x_x = new CentralizedEventCouplingArtifact();
    XAgent.addOutputCouplingArtifact(XOutputToZ_x_x, "x");
    ZAgent.addInputCouplingArtifact(XOutputToZ_x_x, "x");

    CentralizedEventCouplingArtifact YOutputToX_y_y = new CentralizedEventCouplingArtifact();
    YAgent.addOutputCouplingArtifact(YOutputToX_y_y, "y");
    XAgent.addInputCouplingArtifact(YOutputToX_y_y, "y");

    CentralizedEventCouplingArtifact YOutputToZ_y_y = new CentralizedEventCouplingArtifact();
    YAgent.addOutputCouplingArtifact(YOutputToZ_y_y, "y");
    ZAgent.addInputCouplingArtifact(YOutputToZ_y_y, "y");

    CentralizedEventCouplingArtifact ZOutputToY_z_z = new CentralizedEventCouplingArtifact();
    ZAgent.addOutputCouplingArtifact(ZOutputToY_z_z, "z");
    YAgent.addInputCouplingArtifact(ZOutputToY_z_z, "z");

    // create the observing agent, the observing dispatcher, and an observing graph, link them
    ObservingMAgent LorenzObsAgent = new ObservingMAgent("Lorenz", stopTime);
    SwingDispatcherArtifact LorenzObsArtifact = new SwingDispatcherArtifact();
    LorenzObsAgent.setDispatcherArtifact(LorenzObsArtifact);
    LorenzObsArtifact.addObservingArtifact(new PostMortemTXGraphic("Lorenz", "x,y,z",
        Renderer.Line, new String[]{"x","y","z"}, new String[]{"x","y","z"}) );

    //Creation of the links from models to Obs
    CentralizedEventCouplingArtifact XOutputToObs_x = new CentralizedEventCouplingArtifact();
    XAgent.addOutputCouplingArtifact(XOutputToObs_x, "x");
    LorenzObsAgent.addInputCouplingArtifact(XOutputToObs_x, "x");

    CentralizedEventCouplingArtifact YOutputToObs_y = new CentralizedEventCouplingArtifact();
    YAgent.addOutputCouplingArtifact(YOutputToObs_y, "y");
    LorenzObsAgent.addInputCouplingArtifact(YOutputToObs_y, "y");

    CentralizedEventCouplingArtifact ZOutputToObs_z = new CentralizedEventCouplingArtifact();
    ZAgent.addOutputCouplingArtifact(ZOutputToObs_z, "z");
    LorenzObsAgent.addInputCouplingArtifact(ZOutputToObs_z, "z");

    // start software, initialize values, and start co-simuation
    XAgent.startModelSoftware();
    YAgent.startModelSoftware();
    ZAgent.startModelSoftware();
    LorenzObsAgent.startModelSoftware();

    XAgent.setModelParameters();
    YAgent.setModelParameters();
    ZAgent.setModelParameters();

    XAgent.start();
    YAgent.start();
    ZAgent.start();
    LorenzObsAgent.start();
  }
}
```

B

Détail des descriptions

Nous détaillons dans cette annexe les différents documents de description écrits dans nos langages dédiés et utilisés pour nos deux exemples du chapitre 8.

B.1 Multi-modèle d'autoroute hybride

B.1.1 Modèle Cell en *m2xml*

```
model Cell type "model.highway.equation.CellModelArtifact"

interface
  parameters
    length:0.2 //Lenght of the road is 0.2 km
    density:30. speed:70.
    free_speed:100. capacity:1800.
    nb_voie:3 tau:1.
    kappa:1. nu:1.
    nbCells:5 delay:1
    critical_density:45.
  inputs
    upstream:Tuple(Double, Double, Double) //inFlow (flow, speed, ...)
    downstream:Tuple(Double, Double, Double) //inDensity (..., ..., density)
  outputs
    observation:Double
    upstream:Tuple(Double, Double, Double) //outDensity (..., ..., density)
    downstream:Tuple(Double, Double, Double) //outFlow (flow, speed, ...)

information
  description "Cell model used in the hybrid highway example for the MECSYCO middleware.
The explanation can be found in the PhD Thesis of Benjamin Camus."

simulator
  simulation variables
    stopTime:1.
    startTime:0.
    time_step:0.0001
```

Note : Le code du *CellModelArtifact* n'a pas été modifié dans son fonctionnement interne. À l'origine, chaque port envoyait toujours 3 valeurs (flux de véhicule, vitesse moyenne, densité) sous forme d'un type de donnée particulier. Nous avons changé ce type en un tuple de 3 *Double*. Les noms d'origine des ports ont été conservés, les correspondances avec ceux du schéma Figure

8.3 sont précisées en commentaires.

B.1.2 Modèle Event en *m2xml*

```
model Event type "model.highway.event.EventModelArtifact"

interface
  parameters
    road_length:2.
    max_speed:90.
    min_speed:50.
    initial_car_number:20
    headless:false
  inputs
    in:Tuple(Double, Integer, String) //(speed, count, color)
  outputs
    observation:Double //Number of vehicles
    out:Tuple(Double, Integer, String) //(speed, count, color)

information
  description "Traffic model used in the hybrid highway example for the MECSYCO middleware.
The explanation can be found in the PhD Thesis of Benjamin Camus."

simulator
  simulation variables
    stopTime:1.
    startTime:0.
    minPropagationDelay:0.022222222222222223 //road_length/max_speed
```

B.1.3 Modèle Micro en *nl2xml*

```
model Micro path "My Models/netlogo/Traffic-2Roads-100.nlogo"

interface
  //No parameters
  inputs
    in:Vector<String, String> loop command "create-cars 1 [set shape \"car\" setxy min-pxcor 0
                set heading %s set color blue set speed %s set speed-limit 1 set speed-min 0]"
  outputs
    out:Vector<String, String, String, String> report TurtleSet["xcor", "ycor", "heading",
        "speed"] "turtles with [not hidden?]" //Vector of turtles identified by their attributes
information
  description "
    Multi-agent model in the hybrid highway example. Represent a highway with a single road.
    The maximum speed is 100 km/h.
    The time must be adapted, each step of the model must represent 0.00004 hour.
    The road is 0.4 km long, each patch represents 0.004 km.
  "
simulator
  simulation variables
    stopTime:1000.
    discretization:1.
    windowHeight:230
    windowWidth:1250
```

Note : Ici, nous n'avons pas réutilisé l'artéfact de modèle spécifique qui était utilisé. Nous avons utilisé le nouvel artéfact NetLogo. Chaque port est ainsi associé à des commandes à destination de l'interpréteur NetLogo.

B.1.4 Description *mm2xml* du multi-modèle d'autoroute hybride

```
multimodel HybridHighway
submodels
  name Micro path "Library/NetLogo/Micro.xml" //Multi-agent
  name Event path "Library/Basic/Event.xml"
  name Cell path "Library/Basic/Cell.xml"
interface
  parameters
    //Cell parameters
    length:0.2 refers to (Cell:"length")
    density:30. refers to (Cell:"density")
    speed:70. refers to (Cell:"speed")
    free_speed:100. refers to (Cell:"free_speed")
    capacity:1800. refers to (Cell:"capacity")
    critical_density:45. refers to (Cell:"critical_density")
    nb_voie:3 refers to (Cell:"nb_voie")
    tau:1. refers to (Cell:"tau")
    kappa:1. refers to (Cell:"kappa")
    nu:1. refers to (Cell:"nu")
    nbCells:5 refers to (Cell:"nbCells")
    delay:1 refers to (Cell:"delay")
    //Event
    road_length:2. refers to (Event:"road_length")
    max_speed:90. refers to (Event:"max_speed")
    min_speed:50. refers to (Event:"min_speed")
    initial_car_number:20 refers to (Event:"initial_car_number")
    headless:false refers to (Event:"headless")
  outputs //Output ports for observation.
    obsCell:Double refers to (Cell:"observation")
    obsMicro:Double refers to (Micro:"out")
    obsEvent:Double refers to (Event:"observation")
internal couplings
  {Micro."out"->Cell."upstream" time operation multiply 0.00004 //Multiply by the time step of
      the micro model
    event operation "model.highway.operation.Turtles2Flow" //Type of operation
      micro_max_speed=100. micro_length=0.4 //Parameters of the operation
      macro_length=0.2 patch_length=0.004
  }
  {Cell."downstream"->Event."in"
    event operation "model.highway.operation.FlowRate2Group"
      macro_time_step=0.0001
  }
  {Event."out"->Micro."in" time operation divide 0.00004 //Time operation division by the time
      step of the micro model
    event operation "model.highway.operation.Group2Turtle"
      microMaxSpeed=100.
  }
simulator simulation variables
    stopTime:4. refers to (Event:"stopTime") (Cell:"stopTime")
    stopTimeMicro:100000. refers to (Micro:"stopTime") //stopTime : 0.00004
    startTime:0. refers to (Event:"startTime") (Cell:"startTime")
    startTimeMicro:0. refers to (Micro:"startTime")
    timeStep:0.0001 refers to (Cell:"time_step")
    minPropagationDelayEventModel:0.022222222222222223
      refers to (Event:"minPropagationDelay") //Computed -> equal to road length / max speed
```

Note : Lieu du couplage entre les modèles et de l'appel des opérations temporelles et évènementielles.

B.1.5 Description *cs2xml* de l'expérience d'autoroute hybride

```
co-simulation ExperimentHybridHighway
model path "Library/Multimodel/HybridHighway.xml"
  parameters
    //Cell parameters
    length=0.2 density=30. speed=70. free_speed=100.
    nb_voie=3 tau=1. kappa=1. nu=1.
    nbCells=5 delay=1 capacity=1800.
    critical_density=45.
    //Event
    road_length=2. max_speed=90.
    min_speed=50. initial_car_number=20
    headless=false
  observing
    artifact "mecsyco.observing.jfreechart.xy.LiveTXGraphic"
      ports Section1 Section2 Section3
      links
        {"obsCell" -> Section1
          event operation "model.highway.operation.Density2Cars" macroLength=0.2 trackCount=3.
              cellCount=5.
        }
        {"obsMicro" -> Section3
          time operation multiply 0.00004
          event operation "model.highway.operation.Turtle2SimulDataDouble"
        }
        {"obsEvent" -> Section2}
      settings
        axisYName="Number of vehicles" lineThickness=2.
        render="Line" legendSize=22

  information
    description "Hybrid highway experiment."

  simulation parameters
    stopTime=1.15 stopTimeMicro=37500. //stopTime : 0.00004
    startTime=0. startTimeMicro=0.
    timeStep=0.0001
    minPropagationDelayEventModel=0.022222222222222223 //road_length : max_speed
```

Note : Choix des paramètres, mais aussi sélection du graphique et couplages des ports de sortie du modèle sur les ports graphiques (avec ajout d'opérations).

B.2 Multi-modèle thermique

B.2.1 Description *m2xml* du modèle météo

```
model WeatherDB type "model.thermic.SQLWeatherModelArtifact"
interface
  parameters
    startDate:"2018-08-01"
    endDate:"2018-08-02"
  outputs
    TWeather:Double
    DatedTWeather:Tuple(String, Double)
information
  description
"This model is a database that contains data about the weather from July 4th to August 20th,
    stored every 20 seconds.
At each time step the model advances its simulator clock by one timeStep.
Like this, when the clock of the simulator advances of 1, it represents 1 second."
simulator
  simulation variables
    stopTime:86400. //End time of the simulation
    startTime:0. //Start time of the simulation
    timeStep:60. //Chosen time step, in s
```

B.2.2 Description *m2xml* du modèle du climatiseur

```
model SmartConditioner type "model.thermic.SmartConditionerModelArtifact"
interface
  parameters
    Twanted:293.15
    tolerance:1.
    maxHeatflow:200. //Used to heat or to cool down
    Troom_init:293.15
  inputs
    Troom:Double initOption parameter
  outputs
    Qroom:Double initOption no default value 0.
    PowerConsumed:Double initOption no default value 0.
information
  keywords "Thermic" "Smart" "Heater" "Conditioner"
  description "Basic regulator used to show the integration of a new model artifact in a MECSYCO
      co-simulation."
simulator
  simulation variables
    stopTime:86400.
    startTime:0.
    timeStep:1.
```

B.2.3 Description *mm2xml* du modèle de pièce régulée

```
multimodel ConditionedRoom
  submodels
    name Room path "My Models/fmu/Thesis_example_Room_2inputs_JM21_Cs2.fmu"
    name SmartConditioner path "Library/Basic/SmartConditioner.xml"
  interface
    parameters
      //Room
      V:30. refers to (Room: "V")
      VolumicHeatCapacity:6000. refers to (Room: "cv")
      Tinit: 293.15 refers to (Room: "heatCapacitor1.T") (SmartConditioner: "Troom_init")
      //Air conditioner
      Twanted:293.15 refers to (SmartConditioner: "Twanted")
      tolerance:1. refers to (SmartConditioner: "tolerance")
      maxHeatflow:200. refers to (SmartConditioner: "maxHeatflow")
    inputs
      Qext:Double refers to (Room:"Qext")
    outputs
      Troom:Double refers to (Room:"Tin")
      PowerConsumed:Double refers to (SmartConditioner:"PowerConsumed")
  internal couplings
    {Room."Tin" -> SmartConditioner."Troom"}
    {SmartConditioner."Qroom" -> Room."Qheater"}
  information
    keywords "Thermic" "Example" "Conditioner"
    description "Multimodel of a room heated by a smart heater (designed in Java)."
  simulator
    simulation variables
      stopTime:86400. refers to (Room:"stopTime") (SmartConditioner:"stopTime")
      timeStep:60. refers to (Room:"timeStep") (SmartConditioner:"timeStep")
      startTime:0. refers to (Room:"startTime") (SmartConditioner:"startTime")
```

B.2.4 Description *mm2xml* du multi-modèle de l'étage climatisé

```
multimodel CorridorWith10Offices
  submodels
    name Office1 path "Library/Multimodel/ConditionedRoom.xml"
    name Office2 path "Library/Multimodel/ConditionedRoom.xml"
    ...
    name Corridor path "Library/Multimodel/ConditionedRoom.xml"
    name OuterWallCorridor path "My Models/fmu/Thesis_example_Wall_JM21_Cs2.fmu"
    name Wall1Corridor path "My Models/fmu/Thesis_example_Wall_JM21_Cs2.fmu"
    ...
    name OuterWall1 path "My Models/fmu/Thesis_example_Wall_JM21_Cs2.fmu"
    ...
    name Wall1_2 path "My Models/fmu/Thesis_example_Wall_JM21_Cs2.fmu"
    ...
    interface
    parameters
      VolumeOffices:30. refers to (Office1: "V") ...
      VolumeCorridor:225 refers to (Corridor:"V")
      VolumicHeatCapacityOffices:6000. refers to (Office1: "VolumicHeatCapacity")...
      ...
    inputs
      Text1:Double refers to (OuterWall1:"T_b")
      Text2:Double refers to (OuterWall2:"T_b")
      ...
    outputs
      TOffice1:Double refers to (Office1:"Troom")
      TOffice2:Double refers to (Office2:"Troom")
      ...
      PowerConsumed1:Double refers to (Office1:"PowerConsumed")
      PowerConsumed2:Double refers to (Office2:"PowerConsumed")
      ...
  internal couplings
    {Office1."Troom" -> Wall1_2."T_a"} {Office2."Troom" -> Wall1_2."T_b"}
    ...
    {Office1."Troom" -> Wall1Corridor."T_a"}
       ...
    {Office1."Troom" -> OuterWall1."T_a"}
    ...
    {aggregation SumDouble,
           (Wall1_2."Q_a", OuterWall1."Q_a", Wall1Corridor."Q_a") -> (Office1."Qext")}
    ...
  information
    keywords "Thermic" "Example"
    description "Multimodel of ten successive rooms accessible by a common corridor.
The heat flow is computed through a Wall model (just a thermal resistance).
This model shows how hard it can be to connect several model at the same level."
  simulator
    simulation variables
      stopTime:86400. refers to (Office1:"stopTime") (OuterWall1:"stopTime") ...
      timeStep:60. refers to (Office1:"timeStep") (OuterWall1:"timeStep") ...
      startTime:0. refers to (Office1:"startTime") (OuterWall1:"startTime") ...
/* 40 models, 15 parameters with numerous "refers to", 11 inputs, 22 outputs, 60 internal
    couplings et 3 simulation parameter with 40 references each. This description lies in more
    than 200 lines. */
```

B.2.5 Description *mm2xml* du multi-modèle d'étage climatisé couplé au modèle météo

```
multimodel CorridorWith10OfficesSubmittedToRealOutdoorTemperature
  submodels
    name Floor path "Library/Multimodel/CorridorWith10Offices.xml"
    name Weather path "Library/Basic/WeatherDB.xml"
  interface
    parameters
      VolumeOffices:30. refers to (Floor: "VolumeOffices")
      VolumeCorridor:225 refers to (Floor:"VolumeCorridor")
      VolumicHeatCapacityOffices:8000. refers to (Floor: "VolumicHeatCapacityOffices")
      VolumicHeatCapacityCorridor:4000. refers to (Floor:"VolumicHeatCapacityCorridor")
      TextInit: 288.15 refers to (Floor: "TextInit") (Weather: "offset")
      Tinit: 293.15 refers to (Floor: "Tinit")
      Twanted:293.15 refers to (Floor: "Twanted")
      tolerance:1. refers to (Floor:"tolerance")
      maxHeatflow:150. refers to (Floor: "maxHeatflow")
      maxHeatflowCorridor:450. refers to (Floor:"maxHeatflowCorridor")
      SOuterWalls2To9:7.5 refers to (Floor:"SOuterWalls2To9")
      SOuterWalls1And10:17.5 refers to (Floor:"SOuterWalls1And10")
      SOuterWallCorridor:95. refers to (Floor:"SOuterWallCorridor")
      OuterWallLambda:0.7 refers to (Floor: "OuterWallLambda")
      InnerWallLambda:0.3 refers to (Floor:"InnerWallLambda")
      startDate:"2018-08-01" refers to (Weather:"startDate")
      endDate:"2018-08-02" refers to (Weather:"endDate")
    outputs
      TOffice1:Double refers to (Floor:"TOffice1")
      TOffice2:Double refers to (Floor:"TOffice2")
      TOffice3:Double refers to (Floor:"TOffice3")
      ...
      TCorridor:Double refers to (Floor:"TCorridor")
      OutTemp:Double refers to (Weather:"TWeather")
      PowerConsumed1:Double refers to (Floor:"PowerConsumed1")
      PowerConsumed2:Double refers to (Floor:"PowerConsumed2")
      PowerConsumed3:Double refers to (Floor:"PowerConsumed3")
      ...
      PowerConsumedCorridor:Double refers to (Floor:"PowerConsumedCorridor")
  internal couplings
    {Weather."TWeather" -> Floor."Text1"
           event operation "model.thermic.operation.event.AddDoubleOperation" arg=273.15}
    {Weather."TWeather" -> Floor."Text2" event operation
           "model.thermic.operation.event.AddDoubleOperation" arg=273.15}
    {Weather."TWeather" -> Floor."Text3" event operation
           "model.thermic.operation.event.AddDoubleOperation" arg=273.15}
    ...
    {Weather."TWeather" -> Floor."TextCorridor" event operation
           "model.thermic.operation.event.AddDoubleOperation" arg=273.15}
  simulator
    simulation variables
      stopTime:86400. refers to (Floor:"stopTime") (Weather:"stopTime")
      timeStep:60. refers to (Floor:"timeStep") (Weather:"timeStep")
      startTime:0. refers to (Floor:"startTime") (Weather:"startTime")
```

Note : Nos tests, avant l'accès à une base de données réelles, utilisaient une FMU qui générait une sinusoïde pour modéliser la température. Pour échanger ces deux modèles, il a suffit (1) de changer le chemin de *Weather* et (2) d'adapter les paramètres et ports de sortie utilisés puisque l'interface de la FMU était différente. À interface identique, seul le chemin doit être changé.

B.2.6 Description *cs2xml* de l'expérience sur le multi-modèle d'étage régulé

```
co-simulation ExperimentOnFloorSubmittedToRealOutdoorTemperature
model path "Library/Multimodel/CorridorWith10OfficesSubmittedToRealOutdoorTemperature.xml"
parameters
  //ConditionedRooms
  VolumeOffices=30. //Volume of the room
  VolumeCorridor=225 //Volume of the corridor
  VolumicHeatCapacityOffices=50000. //Volumic heat capacity of the room
  VolumicHeatCapacityCorridor=30000. //Volumic heat capacity of the corridor
  Tinit= 293.15 //Initial temperature in rooms
  Twanted=293.15 //Setpoint temperature
  tolerance=1. //Tolerance on the indoor temperature
  maxHeatflow=1500. //Max heatflow of the air conditioners in rooms
  maxHeatflowCorridor=7500. //Max heatflow of the air conditioners in the corridor
  //Walls
  TextInit= 288.15 //Initial temperature of the outside
  SOuterWalls2To9=7.5 //Outer wall surface for room 2 to 9
  SOuterWalls1And10=17.5 //Outer wall surface for room 1 and 10
  SOuterWallCorridor=95. ////Outer wall surface for the corridor
  OuterWallLambda=3. //Thermic resistance of the outer walls
  InnerWallLambda=7. //Thermic resistance of the inner walls
  //Outdoor temperature (real data)
  startDate="2018-08-02"
  endDate="2018-08-04"
observing
  artifact "mecsyco.observing.jfreechart.xy.LiveTXTimeGraphic"
      ports TOffice1 TOffice2 TOffice3 TOffice4 TOffice5 TCorridor Text
      links
        {"TOffice1" -> TOffice1} {"TOffice2" -> TOffice2}
        {"TOffice3" -> TOffice3} {"TOffice4" -> TOffice4}
        {"TOffice5" -> TOffice5} {"TCorridor" -> TCorridor}
        {"OutTemp" -> Text event operation "model.thermic.operation.event.AddDoubleOperation" arg
            =273.15}
      settings
        dateFormat="yyyy-MM-dd" startDate="2018-08-02"
        displayDateFormat="dd MMM HH:mm" scale="SECOND"
        Yaxis="Temperature in Celsius" render="Step" lineThickness=2. legendSize=22
    artifact "mecsyco.observing.jfreechart.xy.LiveTXTimeGraphic"
      ports PowerConsumed1 PowerConsumed2 PowerConsumed3 PowerConsumed4 PowerConsumed5
          PowerConsumedCorridor
      links
        {"PowerConsumed1" -> PowerConsumed1} {"PowerConsumed2" -> PowerConsumed2}
        {"PowerConsumed3" -> PowerConsumed3} {"PowerConsumed4" -> PowerConsumed4}
        {"PowerConsumed5" -> PowerConsumed5} {"PowerConsumedCorridor" -> PowerConsumedCorridor}
      settings
        dateFormat="yyyy-MM-dd" startDate="2018-08-02"
        displayDateFormat="dd MMM HH:mm"
        scale="SECOND" Yaxis="Power in W" render="Step" thickness=3. legendSize=22
simulation parameters
    stopTime=172800.
    timeStep=120.
    startTime=0.
```

Bibliographie

[Amblard et al., 2006] Amblard, F., Bommel, P., and Rouchier, J. (2006). Évaluation et validation de modèles multi-agents. In *Modélisation et Simulation Multi-Agent : application pour les Sciences de l'Homme et de la Société*, pages 87–103.

[Andersson, 2017] Andersson, C. (2017). JModelica.org User Guide - Version 2.1. page 169.

[Awais et al., 2013] Awais, M. U., Palensky, P., Elsheikh, A., Widl, E., and Matthias, S. (2013). The High Level Architecture RTI as a master to the Functional Mock-up Interface components. pages 315–320. IEEE.

[Bastian et al., 2011] Bastian, J., Clauß, C., Wolf, S., and Schneider, P. (2011). Master for co-simulation using FMI. In *8th International Modelica Conference, Dresden*. Citeseer.

[Bettini and Efftinge, 2016] Bettini, L. and Efftinge, S. (2016). *Implementing domain-specific languages with Xtext and Xtend : learn how to implement a DSL with Xtext and Xtend using easy-to-understand examples and best practices*. Packt open source community experience distilled. Packt Publishing, Birmingham Mumbai, second edition edition. OCLC : 965315593.

[Blochwitz et al., 2014] Blochwitz, T., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Franke, R., Friedrich, M., Greenberg, L., Junghanns, A., Mauss, J., Nakhimovski, I., Neumerkel, D., Olsson, H., Otter, M., and Viel, A. (2014). FMI for Model-Exchange and Co-Simulation v2.0.

[Bollinger et al., 2016] Bollinger, L. A., van Blijswijk, M. J., Dijkema, G. P., and Nikolic, I. (2016). An Energy Systems Modelling Tool for the Social Simulation Community. *Journal of Artificial Societies and Social Simulation*, 19(1).

[Bonneaud, 2008] Bonneaud, S. (2008). *Des agents-modèles pour la modélisation et la simulation de systèmes complexes : application à l'écosystémique des pêches*. PhD thesis, Université de Bretagne occidentale-Brest.

[Borshchev and Filippov, 2004] Borshchev, A. and Filippov, A. (2004). From System Dynamics and Discrete Event to Practical Agent Based Modeling : Reasons, Techniques, Tools. page 23.

[Brück et al., 2002] Brück, D., Elmqvist, H., Mattsson, S. E., and Olsson, H. (2002). Dymola for Multi-Engineering Modeling and Simulation. page 9.

[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.

[Brooks and Tobias, 1996] Brooks, R. J. and Tobias, A. M. (1996). Choosing the best model : Level of detail, complexity, and model performance. *Mathematical and computer modelling*, 24(4) :1–14.

[Bryant, 1979] Bryant, R. E. (1979). Simulation on a Distributed System. In *Proceedings of the 16th Design Automation Conference*, pages 544–552.

[Bézivin, 2004] Bézivin, J. (2004). Sur les principes de base de l'ingénierie des modèles. page 13.

[Camus, 2015] Camus, B. (2015). *Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes*. PhD thesis, Université de Lorraine.

[Camus et al., 2015] 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*, pages 85–90. Society for Computer Simulation International.

[Camus et al., 2018] Camus, B., Paris, T., Vaubourg, J., Presse, Y., Bourjot, C., Ciarletta, L., and Chevrier, V. (2018). Co-simulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware. *SIMULATION*.

[Camus et al., 2017] Camus, B., Vaubourg, J., Paris, T., Presse, Y., Bourjot, C., Ciarletta, L., and Chevrier, V. (2017). Wrapping DEVS de modèles IP dans MECSYCO pour la co-simulation de systèmes cyber-physiques. *Technique et Science Informatiques*, 36(3-6) :185–215.

[Chandy and Misra, 1979] Chandy, K. M. and Misra, J. (1979). Distributed Simulation : A Case Study in Design and Verification of Distributed Programs. In *IEEE Transactions on software engineering*, pages 440–452.

[Collberg et al., 2015] Collberg, C., Proebsting, T., and Warren, A. M. (2015). Repeatability and benefaction in computer systems research. *University of Arizona TR 14*, 4.

[Cossentino et al., 2010] Cossentino, M., Gaud, N., Hilaire, V., Galland, S., and Koukam, A. (2010). ASPECS : an agent-oriented software process for engineering complex systems : How to design agent societies under a holonic perspective. *Autonomous Agents and Multi-Agent Systems*, 20(2) :260–304.

[Dahmann, 1999] Dahmann, J. S. (1999). The High Level Architecture and beyond : technology challenges. In *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 64–70. IEEE Computer Society.

[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 Anderson, 2004] Davis, P. K. and Anderson, R. H. (2004). Improving the composability of DoD models and simulations. *The Journal of Defense Modeling and Simulation : Applications, Methodology, Technology*, 1(1) :5–17.

[De Lara and Vangheluwe, 2002] De Lara, J. and Vangheluwe, H. (2002). AToM3 : A Tool for Multi-formalism and Meta-modelling. In *International Conference on Fundamental Approaches to Software Engineering*, pages 174–188. Springer.

[Delinchant, 2011] Delinchant, B. (2011). *La CAO et l'optimisation de systèmes, une approche par couplages dynamiques de composants*. PhD thesis, Université de Grenoble.

[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*, pages 84–91. Society for Computer Simulation International.

[Diallo et al., 2014] Diallo, S. Y., Padilla, J. J., Gore, R., Herencia-zapana, H., and Tolk, A. (2014). Toward a formalism of modeling and simulation using model theory. *Complexity*, 19(3) :56–63.

[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.

[Eker et al., 2003] Eker, J., Janneck, J., Lee, E., Jie Liu, Xiaojun Liu, Ludvig, J., Neuendorffer, S., Sachs, S., and Yuhong Xiong (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, 91(1) :127–144.

[Fatès and Chevrier, 2010] Fatès, N. and Chevrier, V. (2010). How important are updating schemes in multi-agent systems ? An illustration on a multi-turmite model. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 533–540, Toronto, Canada. International Foundation for Autonomous Agents and Multiagent Systems.

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

[Fomel and Claerbout, 2009] Fomel, S. and Claerbout, J. F. (2009). Reproducible research. *Computing in Science & Engineering*, 11(1) :5–7.

[Fritzson et al., 2005] Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., and Broman, D. (2005). The OpenModelica Modeling, Simulation, and Development Environment. page 8.

[Fritzson and Bunus, 2002] Fritzson, P. and Bunus, P. (2002). Modelica - a general object-oriented language for continuous and discrete-event system modeling and simulation. pages 365–380. IEEE Comput. Soc.

[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.

[Galtier et al., 2015] Galtier, V., Vialle, S., Dad, C., Tavella, J.-P., Lam-Yee-Mui, J.-P., and Plessis, G. (2015). FMI-based distributed multi-simulation with DACCOSIM. In *Proceedings of the Symposium on Theory of Modeling & Simulation : DEVS Integrative M&S Symposium*, pages 39–46. Society for Computer Simulation International.

[Gamma et al., 1994] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series. Addison Wesley, 01 edition.

[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.

[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 *Principles and Practice of Multi-Agent Systems*, pages 620–635. Springer.

[Gilpin et al., 2014] Gilpin, L., Ciarletta, L., Presse, Y., Chevrier, V., and Galtier, V. (2014). Co-simulation solutions using AA4mm-FMI applied to smart space heating models. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 153–159. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Goderis et al., 2007] Goderis, A., Brooks, C., Altintas, I., Lee, E. A., and Goble, C. (2007). Composing different models of computation in Kepler and Ptolemy II. In *Computational Science–ICCS 2007*, pages 182–190. Springer.

[Gomes et al., 2017] Gomes, C., Casper, T., Gorm Larsen, P., and Vangheluwe, H. (2017). Co-simulation state of the art. Technical report.

[Gomes et al., 2018] Gomes, C., Thule, C., Broman, D., Larsen, P. G., and Vangheluwe, H. (2018). Co-Simulation : A Survey. *ACM Computing Surveys*, 51(3) :1–33.

[Grignard et al., 2013] Grignard, A., Taillandier, P., Gaudou, B., Vo, D. A., Huynh, N. Q., and Drogoul, A. (2013). GAMA 1.6 : Advancing the Art of Complex Agent-Based Modeling and Simulation. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Boella, G., Elkind, E., Savarimuthu, B. T. R., Dignum, F., and Purvis, M. K., editors, *PRIMA 2013 : Principles and Practice of Multi-Agent Systems*, volume 8291, pages 117–131. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Henderson et al., 2006] Henderson, T. R., Roy, S., Floyd, S., and Riley, G. F. (2006). ns-3 Project Goals. In *Proceedings of WNS2'06*, page 13. ACM.

[Herbert A., 1962] Herbert A., S. (1962). The architecture of complexity. *Proceedings of the American Philosophical Society*, 106(6) :467–482.

[Hmam et al., 2006] Hmam, M. S. E., Abouaissa, H., and Jolly, D. (2006). Modélisation Hybride du Flux de Trafic. *Revue électronique Sciences et Technologies de l'Automatique*, page 8.

[Hofmann, 2005] Hofmann, M. A. (2005). Modeling Assumptions : How they affect Validation and Interoperability.

[Jacques and Wainer, 2002] Jacques, C. J. D. and Wainer, G. A. (2002). Using the CD++ DEVS Tookit to Develop Petri Nets. In *Summer Computer Simulation Conference*, pages 51–56. Society for Computer Simulation International ; 1998.

[Koenig and Howard, 2004] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154, Sendai, Japan. IEEE.

[Kuhn, 1972] Kuhn, T. S. (1972). La structure des révolutions scientifiques.

[Lane, 2006] Lane, D. (2006). Hierarchy, complexity, society. In *Hierarchy in Natural and Social Science*, pages 81–119. Springer.

[Larsen et al., 2016] Larsen, P. G., Fitzgerald, J., Woodcock, J., Fritzson, P., Brauer, J., Kleijn, C., Lecomte, T., Pfeil, M., Green, O., Basagiannis, S., and Sadovykh, A. (2016). Integrated tool chain for model-based design of Cyber-Physical Systems : The INTO-CPS project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6.

[Lausdahl et al., 2017] Lausdahl, K., Bjerge, K., Bokhove, T., Groen, F., and Gorm Larsen, P. (2017). Transitioning from Crescendo to INTO-CPS. pages 16–30. OCLC : 961996377.

[Le Moigne, 2006] Le Moigne, J.-L. (2006). Théorie du Système Général : Théorie de la Modélisation. *Université Paul Cézanne-Aix Marseille*.

[Lee, 2010] Lee, E. A. (2010). Disciplined Heterogeneous Modeling. In *Model Driven Engineering Languages and Systems*, pages 273–287, Berlin, Heidelberg. Springer.

[Mesirov, 2010] Mesirov, J. P. (2010). Accessible reproducible research. *Science*, 327(5964) :415–416.

[Michel et al., 2009] Michel, F., Ferber, J., and Drogoul, A. (2009). Multi-agent systems and simulation : a survey from the agents community's perspective. In *Multi-Agent Systems : Simulation and Applications*, Computational analysis, synthesis, and design of dynamic models series, pages 3–52. Adelinde M. Uhrmacher and Dany Weyns, crc press/taylor & francis edition. OCLC : ocn262719409.

[Minsky, 1965] Minsky, M. (1965). Matter, mind and models. *MIT Press*.

[Müller, 2010] Müller, J.-P. (2010). A framework for integrated modeling using a knowledge-driven approach. page 9.

[Müller and Widl, 2013] Müller, W. and Widl, E. (2013). Linking FMI-based components with discrete event systems. In *2013 IEEE International Systems Conference (SysCon)*, pages 676–680.

[Morin, 2015] Morin, E. (2015). *Introduction à la pensée complexe*. Le seuil edition.

[Morse, 2004] Morse, K. L. (2004). Data and metadata requirements for composable mission space environments. In *Proceedings of the 36th conference on Winter simulation*, pages 271–278. Winter Simulation Conference.

[North et al., 2013] North, M. J., Collier, N. T., Ozik, J., Tatara, E. R., Macal, C. M., Bragen, M., and Sydelko, P. (2013). Complex adaptive systems modeling with Repast Simphony. *Complex Adaptive Systems Modeling*, 1(1).

[Paris et al., 2018a] Paris, T., Chevrier, V., and Ciarletta, L. (2018a). Une approche composant pour DEVS. In *Journées DEVS Francophones 2018 (JDF'18)*, Cargèse.

[Paris et al., 2017a] Paris, T., Ciarletta, L., and Chevrier, V. (2017a). Designing co-simulation with multi-agent tools : a case study with NetLogo. In *Proceedings of the 15th European Workshop on Multi-Agent Systems. EUMAS*.

[Paris et al., 2017b] Paris, T., Ciarletta, L., and Chevrier, V. (2017b). Intégration d'un simulateur multi-agent dans une plateforme de co-simulation DEVS (poster et démonstration). In *25èmes Journées Francophones sur les Systèmes Multi-Agents (JFSMA'17)*, Caen.

[Paris et al., 2018b] Paris, T., Ciarletta, L., and Chevrier, V. (2018b). Co-simulation à base d'outils multi-agents : un cas d'étude avec NetLogo. In *JFSMA'18*, pages 201–210, Métabief. Cépaduès.

[Paris et al., 2016] Paris, T., Tan, A., Chevrier, V., and Ciarletta, L. (2016). Study about decomposition and integration of continuous systems in discrete environment. In *Proceedings of the Annual Simulation Symposium*, Pasadena. SCS/ACM.

[Pennock and Rouse, 2014] Pennock, M. J. and Rouse, W. B. (2014). Why connecting theories together may not work : How to address complex paradigm-spanning questions. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 373–378. IEEE.

[Petty and Weisel, 2003a] Petty, M. D. and Weisel, E. W. (2003a). A Composability Lexicon.

[Petty and Weisel, 2003b] Petty, M. D. and Weisel, E. W. (2003b). A formal basis for a theory of semantic composability. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, pages 416–423.

[Plessis et al., 2014] Plessis, G., Kaemmerlen, A., and Lindsay, A. (2014). BuildSysPro a Modelica library for modelling buildings and energy systems. pages 1161–1169.

[Quesnel, 2006] Quesnel, G. (2006). *Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes*. PhD thesis, Université du Littoral-Côte d'Opale.

[Quesnel et al., 2005] Quesnel, G., Duboz, R., and Ramat, E. (2005). Wrapping into DEVS Simulator : A Study Case. *International Mediterranean Modeling Multiconference*, pages 374–382.

[Quesnel et al., 2009] Quesnel, G., Duboz, R., and Ramat, r. (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., 2007] Quesnel, G., Duboz, R., Ramat, r., and Traoré, M. K. (2007). VLE - A Multimodeling and Simulation environment. In *Proceedings of the 2007 summer computer simulation conference*, pages 367–374. Society for Computer Simulation International.

[Rajkumar et al., 2010] Rajkumar, R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical systems : The next computing revolution. In *Design Automation Conference*, pages 731–736.

[Ramat, 2006] Ramat, E. (2006). Introduction à la simulation : principaux concepts. In *Modélisation et Simulation Multi-Agent : application pour les Sciences de l'Homme et de la Société*, pages 37–60.

[Ören, 2011] Ören, T. (2011). The many facets of simulation through a collection of about 100 definitions. *SCS M&S Magazine*, 2(2) :82–92.

[Ricci et al., 2007] Ricci, A., Viroli, M., and Omicini, A. (2007). 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*, page 150. ACM.

[Robinson, 1997] Robinson, S. (1997). Simulation model verification and validation : increasing the users' confidence. In *Proceedings of the 29th conference on Winter simulation*, pages 53–59. IEEE Computer Society.

[Robinson, 2004] Robinson, S. (2004). *Simulation : the practice of model development and use.* John Wiley & Sons, Ltd, Chichester, West Sussex, England ; Hoboken, NJ.

[Sargent, 2013] Sargent, R. G. (2013). Verification and validation of simulation models. *Journal of simulation*, 7(1) :12–24.

[Sarjoughian, 2006] Sarjoughian, H. S. (2006). Model composability. In *Proceedings of the 38th conference on Winter simulation*, pages 149–158. Winter Simulation Conference.

[Sarjoughian and Singh, 2004] Sarjoughian, H. S. and Singh, R. (2004). Building simulation modeling environments using systems theory and software architecture principles. In *Proceedings of the Advanced Simulation Technology Conference*, pages 99–104.

[Schütte et al., 2011] Schütte, S., Scherfke, S., and Tröschel, M. (2011). Mosaik : A framework for modular simulation of active components in Smart Grids. In *Smart Grid Modeling and Simulation (SGMS), 2011 IEEE First International Workshop on*, pages 55–60. IEEE.

[Seck and Honig, 2012] Seck, M. D. and Honig, H. J. (2012). Multi-perspective modelling of complex phenomena. *Computational & Mathematical Organization Theory*, pages 1–17.

[Sendall and Kozaczynski, 2003] Sendall, S. and Kozaczynski, W. (2003). Model transformation : the heart and soul of model-driven software development. *IEEE Software*, 20(5) :42–45.

[Siebert, 2011] Siebert, J. (2011). *Approche multi-agent pour la multi-modélisation et le couplage de simulations. Application á l'étude des influences entre le fonctionnement des réseaux ambiants et le comportements de leurs utilisateurs.* PhD thesis, Université Henri Poincaré-Nancy.

[Society, 1990] Society, I. C., editor (1990). *IEEE standard computer dictionary : a compilation of IEEE standard computer glossaries, 610.* Institute of Electrical and Electronics Engineers, New York, NY, USA.

[Tavella et al., 2016] Tavella, J.-P., Caujolle, M., Tan, C., Plessis, G., Schumann, M., Vialle, S., Dad, C., Cuccuru, A., and Revol, S. (2016). Toward an Hybrid Co-simulation with the FMI-CS Standard.

[Taylor et al., 2013] Taylor, S. J., Khan, A., Morse, K. L., Tolk, A., and Yilmaz, L. (2013). Grand challenges on the theory of modeling and simulation.

[Tisue and Wilensky, 2004] Tisue, S. and Wilensky, U. (2004). NetLogo A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA.

[Tolk et al., 2012] Tolk, A., Diallo, S., Padilla, J., and Turnitsa, C. (2012). How is M&S Interoperability different from other Interoperability Domains ? *GUEST EDITORIAL*, page 5.

[Tolk et al., 2007] Tolk, A., Diallo, S. Y., and Turnitsa, C. (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*.

[Tolk and Muguira, 2003] Tolk, A. and Muguira, J. A. (2003). The levels of conceptual interoperability model. In *Proceedings of the 2003 Fall Simulation Interoperability Workshop*, volume 7, pages 1–11. Citeseer.

[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.

[Van Deursen et al., 2000] Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-Specific Languages : An Annotated Bibliography. *Sigplan Notices*, 35(6) :26–36.

[Van Tendeloo and Vangheluwe, 2018] Van Tendeloo, Y. and Vangheluwe, H. (2018). Extending the DEVS Formalism with Initialization Information. *arXiv :1802.04527 [cs]*. arXiv : 1802.04527.

[Vangheluwe, 2000] Vangheluwe, H. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 129–134. IEEE.

[Vangheluwe et al., 2002] Vangheluwe, H., De Lara, J., and Mosterman, P. J. (2002). An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems), Lisboa, Portugal*, pages 9–20.

[Varga and Hornig, 2008] Varga, A. and Hornig, R. (2008). An Overview of the OMNeT++ Simulation Environment. ICST.

[Vaubourg, 2017] Vaubourg, J. (2017). *Intégration de modèles de réseaux IP à un multi-modèle DEVS, pour la co-simulation de systèmes cyber-physiques*. PhD thesis, Université de Lorraine.

[Vaubourg et al., 2016] Vaubourg, J., Chevrier, V., Ciarletta, L., and Camus, B. (2016). Co-simulation of IP network models in the Cyber-Physical systems context, using a DEVS-based platform. In *Proceedings of the 19th Communications & Networking Symposium*, page 2. Society for Computer Simulation International.

[Vaubourg et al., 2015] 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*, volume 9086, pages 240–251. Springer International Publishing, Cham.

[Versmisse et al., 2006] Versmisse, D., Soulié, J. C., and Quesnel, G. (2006). Une étude de cas dans le domaine des pêcheries pour la simulation de systèmes complexes. In *Actes de la 6ème Conférence Francophone de Modélisation et Simulation.*, Rabat, Maroc.

[Vitek and Kalibera, 2011] Vitek, J. and Kalibera, T. (2011). Repeatability, Reproducibility and Rigor in Systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 33–38. ACM.

[Voelter et al., 2013] Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L., Visser, E., and Wachsmuth, G. (2013). DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. *dslbook.org*.

[Wainer et al., 2010a] Wainer, G., Al-Zoubi, K., Dalle, O., Hill, D. R. C., Mittal, S., Martin, J. L. R., Sarjoughian, H., Touraille, L., Traoré, M. K., and Zeigler, B. P. (2010a). Standardizing DEVS Simulation Middleware. In *Discrete-Event Modeling and Simulation : Theory and Applications*, page 459.

[Wainer et al., 2010b] Wainer, G., Al-Zoubi, K., Hill, D., Mittal, S., Martín, J., Sarjoughian, H., Touraille, L., Traoré, M., and Zeigler, B. (2010b). An Introduction to DEVS Standardization. In Wainer, G. and Mosterman, P., editors, *Discrete-Event Modeling and Simulation*, volume 20101361, pages 393–425. CRC Press.

[Wainer et al., 2010c] Wainer, G., Al-Zoubi, K., Hill, D., Mittal, S., Martín, J., Sarjoughian, H., Touraille, L., Traoré, M., and Zeigler, B. (2010c). Standardizing DEVS Model Representation. In Wainer, G. and Mosterman, P., editors, *Discrete-Event Modeling and Simulation*, volume 20101361, pages 427–458. CRC Press.

[Wilensky, 1999] Wilensky, U. (1999). NetLogo (and NetLogo User Manual).

[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. *Simulation Series*, page 6.

[Yilmaz and Ören, 2013] Yilmaz, L. and Ören, T. (2013). Toward replicability-aware modeling and simulation : Changing the conduct of m&s in the information age. *Ontology, Epistemology, and Teleology for Modeling and Simulation*, pages 207–226.

[Yilmaz et al., 2014] Yilmaz, L., Taylor, S. J., Fujimoto, R., and Darema, F. (2014). Panel : the future of research in modeling & simulation. In *Proceedings of the 2014 Winter Simulation Conference*, pages 2797–2811. IEEE Press.

[Zacharewicz, 2006] Zacharewicz, G. (2006). *UN ENVIRONNEMENT G-DEVS/HLA :APPLICATION A LA MODELISATION ET SIMULATION DISTRIBUEE DE WORKFLOW*. PhD thesis, Université de droit, d'économie et des sciences-Aix-Marseille III.

[Zacharewicz et al., 2008] Zacharewicz, G., Frydman, C., and Giambiasi, N. (2008). G-DEVS/HLA Environment for Distributed Simulations of Workflows. *SIMULATION*, 84(5) :197–213.

[Zeigler et al., 2012] Zeigler, B., Sarjoughian, H. S., Duboz, R., and Soulie, J.-C. (2012). *Guide to Modeling and Simulation of Systems of Systems*. Springer Science & Business Media.

[Zeigler, 2006] Zeigler, B. P. (2006). Embedding DEV&DESS in DEVS. In *DEVS Integrative Modeling & Simulation Symposium*, pages 125–132.

[Zeigler et al., 2000] Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic press, New York, 2nd edition.

[Zeigler and Sarjoughian, 2002] Zeigler, B. P. and Sarjoughian, H. S. (2002). Implications of M&S foundations for the V&V of large scale complex simulation models. In *Proceedings of the Foundations for V&V in the 21st Century Workshop, Laurel, MD*. Citeseer.

Publications de l'auteur

Revue internationale

[1] Benjamin Camus, Thomas Paris, Julien Vaubourg, Yannick Presse, Christine Bourjot, Laurent Ciarletta et Vincent Chevrier. "Co-simulation of cyber-physical systems using a DEVS wrapping strategy in the MECSYCO middleware". In : *SIMULATION* (2018).

Conférences internationales avec actes

[2] Thomas Paris, Laurent Ciarletta et Vincent Chevrier. "A component approach for DEVS". In : *Proceedings of the 50th Computer Simulation Conference*. Bordeaux, France : Society for Computer Simulation International, 2018.

[3] Thomas Paris, Laurent Ciarletta et Vincent Chevrier. "Designing co-simulation with multi-agent tools : a case study with NetLogo". In : *Proceedings of the 15th European Workshop on Multi-Agent Systems. EUMAS*. 2017.

[4] Thomas Paris, Alexandre Tan, Vincent Chevrier et Laurent Ciarletta. "Study about decomposition and integration of continuous systems in discrete environment". In : *Proceedings of the Annual Simulation Symposium*. Pasadena : SCS/ACM, 2016.

Revue nationale

[5] Benjamin Camus, Julien Vaubourg, Thomas Paris, Yannick Presse, Christine Bourjot, Laurent Ciarletta et Vincent Chevrier. "Wrapping DEVS de modèles IP dans MECSYCO pour la co-simulation de systèmes cyber-physiques". In : *Technique et Science Informatiques*. Lavoisier 36.3-6 (2017), p. 185–215.

Conférence nationale avec actes

[6] Thomas Paris, Laurent Ciarletta et Vincent Chevrier. "Co-simulation à base d'outils multi-agents : un cas d'étude avec NetLogo". In : *JFSMA'18*. Métabief : Cépaduès, oct. 2018, p. 201–210.

[7] Thomas Paris, Vincent Chevrier et Laurent Ciarletta. "Une approche composant pour DEVS". In : *Journées DEVS Francophones 2018 (JDF'18)*. Cargèse, mai 2018.

Démonstration

[8] Thomas Paris, Laurent Ciarletta et Vincent Chevrier. "Intégration d'un simulateur multi-agent dans une plateforme de co-simulation DEVS (poster et démonstration)". In : *25èmes Journées Francophones sur les Systèmes Multi-Agents (JFSMA'17)*. Caen, 2017.

Résumé

Le contexte de ce travail est la modélisation et simulation (M&S) de systèmes complexes. Ces systèmes se caractérisent par un grand nombre d'entités hétérogènes en interaction faisant apparaitre plusieurs niveaux d'organisation et plusieurs domaines. Leur étude nécessite de combiner plusieurs points de vue (différentes échelles temporelles et spatiales, différents domaines scientifiques et formalismes, différents niveaux de résolution...).

Le challenge est l'intégration rigoureuse de ces différents points de vue sur un système au sein d'une démarche de M&S. Dit autrement, le défi est de définir une marche à suivre permettant d'intégrer plusieurs perspectives au sein d'un même modèle. La multi-modélisation et la co-simulation sont deux approches prometteuses pour cela. La difficulté sous-jacente est de fournir une démarche de M&S modulaire, hiérarchique, dotée d'une approche d'intégration de composants hétérogènes rigoureuse et associée à un environnement logiciel supportant l'ensemble du cycle de M&S pour la mettre en pratique.

MECSYCO [21] est un intergiciel de co-simulation se focalisant sur la réutilisation de modèles issus d'autres logiciels. Il se base sur une stratégie d'encapsulation logicielle et formelle fondée sur DEVS, fournit des mécanismes de gestion des hétérogénéités, et assure une co-simulation décentralisée et modulaire. MECSYCO répond au besoin d'intégration de composants hétérogènes au sein d'une co-simulation, mais ne propose pas de démarche complète comprenant l'ensemble des propriétés énoncées précédemment. Il manque notamment la possibilité de hiérarchiser.

Pour pallier à ce manque, dans la continuité des travaux sur MECSYCO nous proposons une démarche de multi-modélisation et co-simulation descriptive autorisant la construction incrémentale de multi-modèles à partir de modèles issus d'autres logiciels. Notre démarche est décomposée en trois étapes : l'intégration des modèles atomiques, la composition (création hiérarchique du multi-modèle) et enfin l'expérimentation. Nous adoptons une approche descriptive où chaque élément produit lors de ces étapes est associé à une description permettant de le manipuler. L'utilisation des descriptions complète le processus d'intégration, permet la construction incrémentale et modulaire des multi-modèles, et isole l'expérimentation.

Nous mettons ensuite en place un environnement de développement basé sur des langages dédiés aux descriptions, et nous automatisons le passage d'une description d'expérience à sa co-simulation effective. C'est une démarche d'IDM [22] qui nous permet de mettre en pratique notre approche en facilitant le travail des modélisateurs et en évitant les erreurs d'implémentation.

Nous apportons à MECSYCO la propriété de hiérarchisation et un environnement de développement tout en conservant l'intégration rigoureuse et la modularité. Nous évaluons notre contribution sur deux exemples. Le premier reprend un multi-modèle d'autoroute hybride implémenté dans MECSYCO, il montre la conservation des propriétés d'intégration. Le second est un multi-modèle simple de thermique de bâtiment intelligent, il illustre la construction incrémentale d'un multi-modèle et l'intégration de nouveaux composants tout en mettant en pratique l'ensemble de notre démarche.

Mots-clés: Système complexe, Multi-modélisation, Co-simulation, Hiérarchie

21. *Multi-agent Environment for Complex SYstem CO-simulation*
22. Ingénierie Dirigée par les Modèles

Abstract

This work deals with complex system Modeling and Simulation (M&S). The particularity of such systems is the numerous heterogeneous entities in interaction involved inside them. This particularity leads to several organization layers and scientific domains. As a consequence, their study requests many perspectives (different temporal and spatial scales, different domains and formalisms, different granularities...).

The challenge is the rigorous integration of these various system perspectives inside a M&S process. In other words, the difficulty is to define successive steps to follow in order to integrate several points of view inside the same model. Multi-modeling and co-simulation are promising approaches to do so. The underlying problem is to define a modular and hierarchical process fitted with a rigorous way to integrate heterogeneous components and which is supported by a software environment that covers the whole M&S cycle.

MECSYCO [23] is a co-simulation middleware focusing on the reuse of existing models from other software. It relies on a software and formal DEVS-based wrapping, provides heterogeneity handling mechanisms and ensures a decentralized and modular co-simulation. MECSYCO deals with the heterogeneous component integration need but its M&S process does not have all the properties above-mentioned. Notably, the hierarchical modeling ability is missing.

To overcome this, we propose to fit MECSYCO with a descriptive multi-modeling and co-simulation process that allows the hierarchical design of multi-models using models from other software. Our process is split into three steps : the atomic model integration, the composition (hierarchical multi-model construction) and finally the experimentation. We adopt a descriptive approach where a description file is linked to each product of these steps, these documents enable to manipulate them. The use of description files completes the integration steps, allows a hierarchical and modular multi-model design and isolates the experiments.

Then we set up a development environment based on Domain Specific Languages (DSL) to support the description work, and we automate the transition from an experiment description to its effective co-simulation. This is a MDE [24] approach which allows us to put into practice our contribution by facilitating the modelers' work and by avoiding implementation mistakes.

Our contribution fits MECSYCO with the hierarchical design property and with a DSL-based M&S environment while keeping its rigorous integration process and its modularity. Our work is evaluated on two examples. The first one renews a hybrid highway multi-model already implemented in MECSYCO, it shows the conservation of the middleware former properties. The second one is a simple thermal smart-building multi-model which highlights the incremental design of a multi-model and the integration of new components while putting our entire approach into practice.

Keywords: Complex system, Multi-modeling, Co-simulation, Hierarchy.

23. Multi-agent Environment for Complex SYstem CO-simulation
24. Model-Driven Engineering