# Intégration de modèles de réseaux IP à un multi-modèle DEVS, pour la co-simulation de systèmes cyber-physiques

Julien Vaubourg

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

# Intégration de modèles de réseaux IP à un multi-modèle DEVS, pour la co-simulation de systèmes cyber-physiques

## THÈSE

présentée et soutenue publiquement le 25 avril 2017

pour l'obtention du

**Doctorat de l'Université de Lorraine**

**(mention informatique)**

par

Julien VAUBOURG

**Composition du jury**

| | | |
|---|---|---|
| *Président :* | Dominique MÉRY | Professeur, Université de Lorraine |
| *Rapporteurs :* | Dominique GAÏTI | Professeuse, Université de Troyes |
| | Éric RAMAT | Professeur, Université du Littoral - Côte d'Opale |
| *Examinateurs :* | Olivier DALLE | Maître de Conférences, Université de Nice - Sophia Ant. |
| | Laurent CIARLETTA | Maître de Conférences, Université de Lorraine |
| *Invité :* | Jean-Philippe TAVELLA | Expert Simulation, EDF Lab Paris-Saclay |
| *Directeur de thèse :* | Vincent CHEVRIER | Professeur, Université de Lorraine |

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

Mis en page avec la classe thesul.

# Remerciements

En premier lieu à Vincent et Laurent, qui m'ont fait confiance pour assurer cette thèse, et qui m'ont soutenu jusqu'à la fin.

Ensuite à EDF R&D, et en particulier à Jean-Philippe TAVELLA, pour avoir contribué à permettre que ces travaux aient lieu.

Puis à mes deux joyeux camarades de route, Benjamin et Yannick, qui ont été d'un soutien inestimable ces dernières années : en particulier à Benjamin pour m'avoir aidé à appréhender le monde scientifique, et à Yannick pour m'avoir aidé à le relativiser.

À toutes les merveilleuses personnes qui composent le monde associatif militant, qui a donné du sens à ma passion pour l'informatique. Beaucoup d'entre elles sont aussi mes ami·e·s, pour qui j'ai une pensée particulière.

Évidemment à mes parents, qui m'ont offert tout ce dont j'avais besoin pour arriver à ce résultat, et qui m'ont toujours inconditionnellement soutenus.

Et enfin, pour les rôles différents mais appréciables qu'illes ont joués en amont de ou durant mon doctorat, à Lucas NUSSBAUM, Martin QUINSON, Thomas PARIS, Alexandre TAN, Victorien ELVINGER, Raphaël CHERFAN et Christine BOURJOT. Ainsi qu'à tou·te·s les autres doctorant·e·s, ingénieur·e·s, personnels du laboratoire et stagiaires que j'ai eu l'occasion de cotoyer à cette occasion.

*À Z. de Z.*

# Sommaire

# Liste des Figures et Tableaux

# Liste des Algorithmes et Implémentations

# Chapitre 1

# Introduction

**Sommaire**



## 1.1 Contexte de la thèse

### 1.1.1 Les systèmes cyber-physiques

Ce travail de thèse porte sur l'intégration de modèles et de simulateurs de réseaux IP dans des co-simulations, destinées à étudier le comportement de systèmes cyber-physiques.

Un système est dit cyber-physique dès lors qu'il met en œuvre des systèmes physiques qui sont contrôlés, supervisés, coordonnés et intégrés par des systèmes informatiques [Rajkumar et al., 2010]. Une boucle d'interaction permet en général aux systèmes physiques d'envoyer des retours d'information aux systèmes informatiques, qui les prennent en considération pour adapter leur comportement, et vice-versa [Lee, 2008].

Le terme de système cyber-physique (SCP) est en concurrence avec celui d'Internet des Objets (IoT). Les deux termes sont très similaires et se réfèrent en général aux mêmes concepts. Le terme IoT a toutefois la préférence des industriels, et a été largement popularisé auprès du grand public, tandis que le terme de SCP semble plutôt restreint au monde scientifique. Ces objets connectés correspondent généralement à ce qu'on appelle des systèmes embarqués. Les SCP sont également concernés par tout ce qu'on gratifie du suffixe « intelligent » (ou du préfixe *smart-* en anglais). Ainsi, les réseaux électriques, villes, voitures, appartements et autres choses qui se revendiquent comme intelligents ont pour principale caractéristique de se reposer sur des SCP pour « augmenter » leurs capacités.

Les SCP sont généralement composés d'entités hétérogènes. Il sont plus particulièrement composés de capteurs et d'effecteurs reliés à des systèmes physiques, et de systèmes informatiques décisionnels. Ces différents éléments communiquent entre eux par le biais de réseaux

de communication, désormais quasiment toujours basés sur le très répandu protocole IP. Un exemple est proposé en Figure 1.1. **Ces réseaux IP ont un rôle essentiel dans les SCP, puisqu'ils sont l'organe central de ces systèmes.**

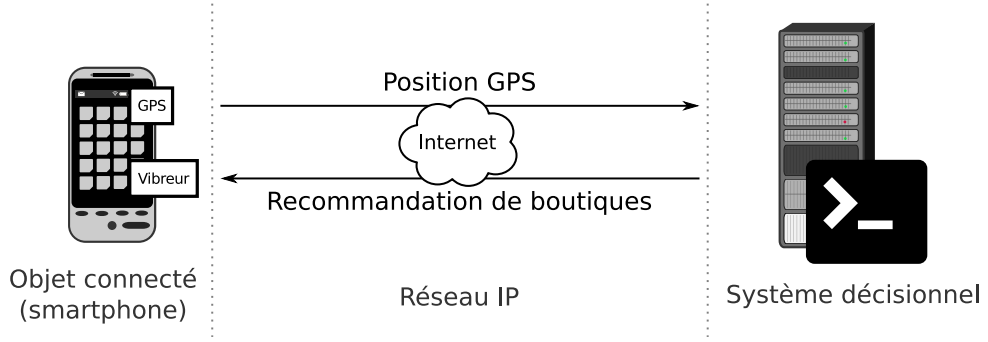



FIGURE 1.1 – Exemple simple d'objet connecté, avec un système de recommandations publicitaires utilisant la localisation GPS (capteur) proposée par un smartphone, pour être capable de vibrer (effecteur) à l'arrivée des nouvelles annonces, avec un réseau IP au centre.

En 2007, la National Science Foundation (NSF) a listé les SCP parmi les domaines de recherche prioritaires [Wolf, 2007]. Les exemples de cas d'utilisation pratiques sont nombreux, et vont de la voiture autonome sans conducteur, aux commandes automatiques de denrées alimentaires en fonction de l'état d'un réfrigérateur, en passant par les détecteurs de fumée communicants. Les SCP sont particulièrement cités dans le domaine de la santé et de l'assistance à la personne à domicile (e.g. à destination des personnes âgées vivant seules), avec de nombreuses propositions de systèmes de capteurs permettant de déterminer automatiquement si un humain est en difficulté, en levant une alerte auprès de qui de droit, le cas échéant.

Les réseaux électriques intelligents sont également des exemples de SCP auxquels s'intéressent la plupart des pays industrialisés. Ils correspondent au cas d'étude privilégié de cette thèse. C'est notamment l'objet de la section suivante.

### 1.1.2 Les réseaux électriques intelligents

Depuis la révolution industrielle qui a permis l'extraction massive de la houille, la France a progressivement adopté la solution du nucléaire pour produire jusqu'à 75% (2009) de l'électricité consommée sur l'ensemble du territoire. Suite au Grenelle de l'Environnement de 2007, elle a désormais pour objectif de produire jusqu'à 20% de son électricité sur la base d'énergies renouvelables (ainsi que réduire de 20% sa consommation électrique et de 20% ses émissions de gaz à effet de serre), diminuant d'autant les sources traditionnelles que sont le thermique et le nucléaire. Si certaines solutions comme l'hydraulique (avec des barrages ou des STEP [1]) sont déjà exploitées sans difficulté à hauteur de 11%, les sources d'énergies provenant de l'éolien ou du photovoltaïque (moins de 2% actuellement) posent plus de difficultés étant donnée l'absence de maîtrise sur la quantité d'énergie pouvant être produite pour un instant donné. Elles sont qualifiées d'*intermittentes*.

Cette part croissante des énergies renouvelables est certainement le phénomène le plus significatif de l'évolution du réseau électrique français depuis les cinq dernières années. En France,

1. Station de Transfert d'Énergie par Pompage

la capacité de production des fermes éoliennes en activité a augmenté de 3,5 GW en 2009 pour atteindre quasiment 8 GW fin 2013. Dans la même période, la capacité de production des fermes photovoltaïques en activité a été multipliée par 50 (4,5 GW en 2013).

D'après [Farhangi, 2010], presque 8% de l'électricité produite est perdue directement sur le réseau maillé durant le transport. Si on ajoute les 20% de capacité de production des centrales qui ne serviraient que 5% du temps pour répondre aux pics de consommation, on peut aisément comprendre que le modèle fortement centralisé des réseaux d'électricité que nous exploitons peut être amélioré.

Plusieurs solutions pourraient être envisagées :

1. prendre le contrôle sur la consommation ;
2. exploiter des accords transfrontaliers ;
3. rapprocher les sources de production des consommateurs.

D'autres objectifs sous-jacents se dégagent alors.

En particulier :
— la possibilité pour les clients de superviser directement leur consommation dans l'optique de faire des économies grâce aux compteurs intelligents (*smart meters*) ;
— la quasi-disparition des rendez-vous à prendre avec les agents de la compagnie de distribution pour les relevés de compteur, changements de tarification, etc. ;
— l'intégration facile des panneaux photovoltaïques de toit ou des petites éoliennes de particuliers ;
— la prévention des coupures d'énergies en cascade d'un pays à l'autre (cf. exemple du *blackout* de 2003 aux USA [2]) ;
— une meilleure intégration du marché des voitures électriques (un million d'ici 2020 selon RTE), qui déstabilisent les prévisions en déplaçant en permanence les demandes d'énergie, mais qui permettent de disposer d'un créneau de 20 à 22 heures d'immobilité par jour (d'après Bernard Bigot, vice-président du Conseil de surveillance d'Areva) pour répartir la charge.

L'émergence des énergies renouvelables et donc des points de production décentralisés, mais aussi les nouveaux usages de l'électricité (pompes à chaleur, voitures électriques, etc), ainsi que le contrôle des pics de consommation, nécessitent donc de développer des systèmes plus intelligents, en particulier au niveau des réseaux de distribution. De tels systèmes sont appelées des réseaux électriques intelligents (*smart grids*), et sont souvent considérés comme étant au cœur de la *troisième révolution industrielle* (illustration en Figure 1.2).

Pour atteindre les objectifs listés ci-dessus, un réseau électrique intelligent doit être composé de trois principales catégories d'éléments, caractéristiques des SCP :

1. des éléments physiques (i.e. électriques) ;
2. des systèmes d'information ;
3. des réseaux de communication (IP).

Chacune de ces catégories nécessite de mettre en œuvre des compétences différentes et correspond donc à un domaine d'expertise différent.

2. http://energy.gov/oe/services/electricity-policy-coordination-and-implementation/august-2003-blackout

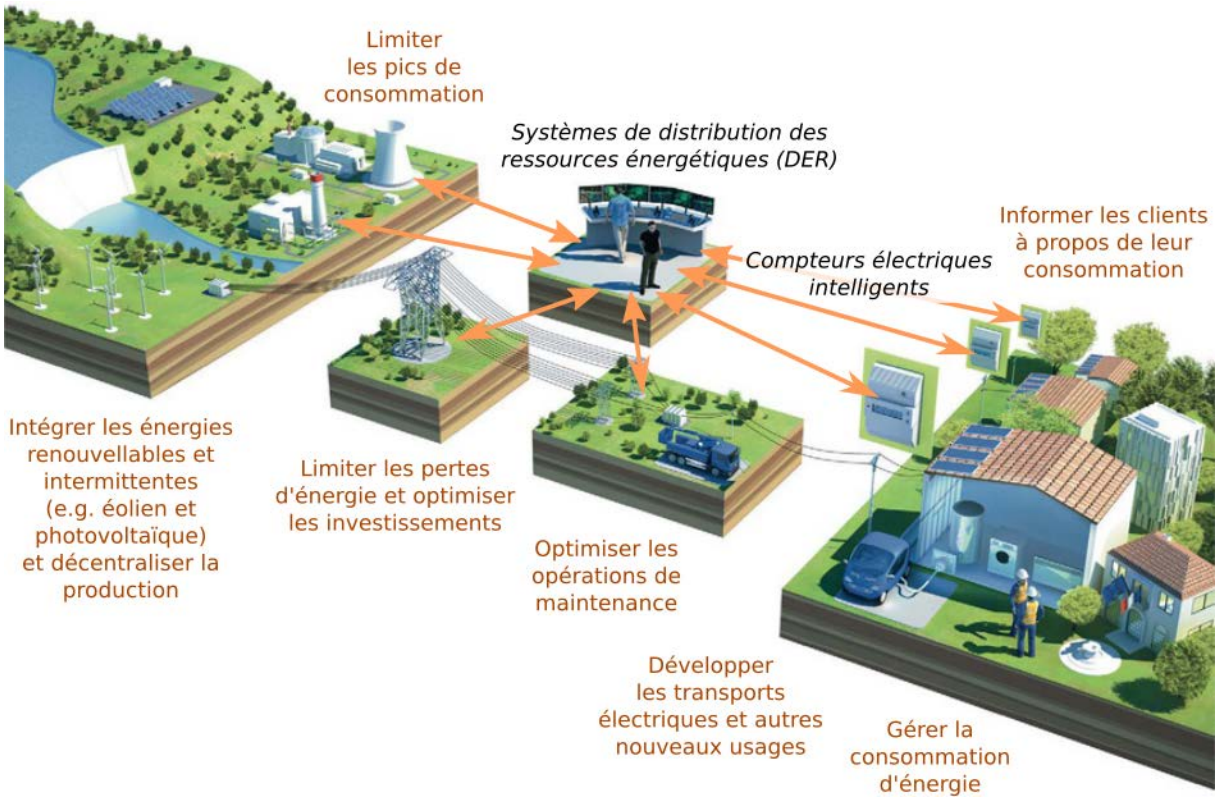



FIGURE 1.2 – Les principaux arguments en faveur des réseaux électriques intelligents (fond de carte ©EDF).

Aucun pays n'a encore déployé de réseau électrique intelligent à grande échelle. En France, le principal distributeur d'électricité ENEDIS (anciennement ERDF), dispose d'un réseau de distribution composé de 1 250 000 kilomètres de lignes électriques, principalement aériennes et en milieu rural [Vaubourg et al., 2015]. En attendant d'ajouter de l'intelligence à l'intégralité de son réseau électrique, ENEDIS teste depuis plusieurs années ces nouvelles technologies sur des démonstrateurs, à l'échelle d'un quartier ou d'une ville. Ainsi, par exemple, le démonstrateur VENTEEA, dans l'Aube près de Troyes, est utilisé pour tester les technologies innovantes pour les réseaux moyenne tension à un emplacement où Enel Green Power France opère de nombreuses fermes éoliennes. Le démonstrateur MILLENER à la Réunion permet de tester des cas de surcharge du réseau électrique, dans une région où le réseau manque d'interconnexions.

Malheureusement, en dépit des avantages de ces prototypes réels, il reste très compliqué de trouver de nouveaux lieux d'expérimentation, et le processus de recrutement de consommateurs industriels ou de particuliers reste long et très coûteux. Ainsi, la simulation des réseaux électriques intelligents est une solution technologique très attractive pour tester de nouveaux algorithmes distribués (e.g. gestion de la tension avancée) ou de nouveaux modes d'opération (e.g. îlotage de péninsules électriques), avant de les tester sur des prototypes réels, et à terme les vrais réseaux électriques.

Les réseaux électriques intelligents, comme les villes intelligentes, ou tout autre SCP de grande envergure, peuvent être considérés comme étant des systèmes complexes. Cette hypothèse de base permet d'orienter les outils nécessaires pour leur modélisation et simulation. La section

suivante a pour objectif d'introduire la notion de systèmes complexes.

### 1.1.3 Les systèmes complexes

D'après [Chavalarias et al., 2009], « *un système complexe désigne en général n'importe quel système comprenant un grand nombre d'entités hétérogènes, entre lesquelles des interactions locales créent de multiple niveaux de structures collectives ainsi que des organisations* ». Ces systèmes peuvent aussi bien être naturels (e.g. colonies d'insectes, cellules vivantes ou interactions humaines) qu'artificiels (e.g. villes, réseaux de transport ou même Internet).

La notion de complexité recouvre plusieurs aspects, et il n'est pas forcément intuitif de déterminer si un système peut être qualifié de complexe ou non. [Chazelle, 2012] propose une étude rapide du sens qu'on donne au terme « complexité », selon quatre points de vue différents :

**Sémantique :** Ce qui est dur à comprendre. Chazelle estime que pour 99,99% de l'humanité, complexité signifie difficile à comprendre.

**Épistémologique :** Ce qui est dur à prédire. Chazelle précise qu'il s'agit d'une notion totalement indépendante de la précédente : un système complexe peut être simple à comprendre, tandis qu'un mécanisme compliqué peut être simple à prédire.

**Opérationnel :** Ce qui est dur à calculer, en termes d'informatique théorique. [3]

**Linguistique :** Ce qui est dur à décrire. Décrire un algorithme ou un comportement peut nécessiter de prendre en considération énormément de variables différentes, qui évoluent en même temps.

Des auteurs comme [Camus, 2015] considèrent par conséquent qu'un système peut être qualifié de complexe dès lors qu'il est à la fois **difficile à comprendre, à prédire et à décrire**, dans le domaine de la simulation.

Certains SCP, comme les réseaux électriques intelligents, sont considérés comme des systèmes complexes par des auteurs comme [Karnouskos and de Holanda, 2009], [Varaiya et al., 2011] ou [Blumsack and Fernandez, 2012]. De manière générale, les SCP nécessitant par définition des interactions entre des entités hétérogènes, la multiplication de ces entités dans les projets de grande envergure (e.g. à l'échelle d'une ville ou d'un pays) les rend généralement comparables à des systèmes complexes.

Étudier ces systèmes est particulièrement difficile, parce que leur description peut demander de faire intervenir de nombreux experts, issus de nombreux domaines d'expertise différents, avec autant d'outils différents à utiliser [Vangheluwe, 2000a]. Ainsi, la description d'un réseau électrique intelligent peut demander de faire intervenir des spécialistes de l'électricité, des télécoms et des systèmes d'information, mais aussi des sociologues, des économistes, des météorologues, etc.

Lorsqu'on souhaite modéliser ces systèmes cyber-physiques complexes, différents modèles de simulation sont nécessaires et doivent être couplés entre eux. Le niveau de pluridisciplinarité requis pour manipuler et interconnecter ces modèles hétérogènes soulève un certain nombre de défis scientifiques, exposés dans la section suivante.

3. *« Instrumental : What is hard to compute, the province of theoretical computer science. », dans la version originale.*

## 1.2 Défis scientifiques et contributions

Lorsqu'on souhaite modéliser et simuler des SCP de grande envergure, plusieurs défis scientifiques et opérationnels spécifiques sont rencontrés. Comme nous l'avons vu dans la section précédente, certains de ces défis peuvent être rapprochés de ceux rencontrés plus généralement dans le cadre de la modélisation et simulation de systèmes complexes.

Les SCP étant généralement composés à la fois de systèmes physiques (pouvant correspondre à des objets de la vie de tous les jours comme à des équipements industriels), de réseaux informatiques et de systèmes décisionnels, les modéliser nécessite des compétences dans des domaines d'expertise très divers et qui ne sont pas habituellement regroupés. Ainsi, il n'existe pas, à notre connaissance, de simulateur de SCP global qui soit capable de représenter à la fois :

1. toute la diversité d'objets existants ;
2. des réseaux informatiques, en prenant en considération toutes les technologies existantes ;
3. tous les types de système décisionnel existants.

Pour tous ces objets, technologies de communication ou systèmes décisionnels, il existe pourtant de nombreux modèles mis à disposition par les communautés correspondantes. Réécrire tous ces modèles serait trop long, trop cher et nécessiterait surtout trop de compétences différentes. De plus, le risque d'introduction d'erreurs en reproduisant ces modèles serait trop important, alors que certains des modèles existants ont été prouvés ou au moins éprouvés au sein de leur communauté d'experts. De plus, des applications métiers de modélisation au sein d'entreprises spécifiques, imposent parfois un existant en termes de ressources, d'habitudes, et de savoir-faire, qui doit impérativement pouvoir être réutilisé en l'état.

Par conséquent, deux principaux défis se dégagent :

1. Parvenir à modéliser des SCP complets en utilisant des modèles déjà existants et issus de différentes communautés. Ces modèles sont généralement exécutables par des simulateurs incompatibles entre eux.
2. Parvenir à connecter ces modèles entre eux dans un même multi-modèle, en intégrant toutes les différentes formes d'hétérogénéité possibles (e.g. formalismes, syntaxe des données, langages, etc).

La solution étudiée dans ce manuscrit de thèse consiste à faire de la co-simulation : chaque simulateur est exécuté de façon indépendante, avec des communications entre les simulateurs et leurs modèles (synchronisation du temps et échange de données de simulation). Les problèmes liés à l'hétérogénéité entre les modèles sont résolus grâce au choix d'utilisation de la plateforme de co-simulation MECSYCO, utilisée pour la simulation de systèmes complexes [Camus, 2015]. Cette plateforme est basée sur l'exploitation du formalisme DEVS [Zeigler et al., 2000] et utilise la notion de wrappeurs DEVS [Quesnel et al., 2009] pour intégrer les modèles et leurs simulateurs, pour ensuite pouvoir les faire communiquer.

Alors que les réseaux informatiques sont au cœur du fonctionnement des SCP, il n'existe aucune solution actuellement pour intégrer des modèles de réseaux informatiques à la solution MECSYCO. De manière plus générale, il n'existe pas, à notre connaissance, de solution pour faire l'interface entre les simulateurs de réseaux IP répandus (le protocole IP étant largement dominant dans le domaine des réseaux informatiques) et le protocole de simulation de DEVS.

**La contribution principale de ce manuscrit consiste donc à proposer une démarche pour wrapper en DEVS des simulateurs IP, de façon à pouvoir les intégrer à des co-simulations basées sur DEVS.**

Deux modes d'interaction avec les modèles IP, qui conduisent à deux modes d'intégration différents, sont pris en considération :

1. interactions entre un modèle IP et un modèle issu d'un autre domaine d'expertise (transmissions de données applicatives à transmettre sur un réseau IP, ou à récupérer depuis un réseau IP) ;
2. et interactions entre deux modèles IP (représentation d'une topologie IP en décrivant des parties distinctes dans des modèles différents).

Les contributions portent par conséquent sur trois aspects plus spécifiques :

1. Être capable de contrôler l'exécution d'un modèle IP durant sa simulation, de façon à pouvoir les exécuter par l'intermédiaire du protocole de simulation DEVS (i.e. être capable d'intervenir dans l'exécution de sa dynamique).
2. Être capable d'injecter et de récupérer des données de simulation dans et depuis un modèle IP (i.e. être capable de définir des ports d'entrée et de sortie dans un modèle IP).
3. Être capable de décrire une topologie IP par l'intermédiaire d'un multi-modèle composé de différents modèles IP, qui soit équivalent à une description similaire qui n'utiliserait qu'un seul modèle IP (i.e. être capable de découper une topologie IP pour répartir sa description sur plusieurs modèles, sans influencer les résultats de simulation).

Ces travaux sont illustrés par l'intermédiaire de preuves de concept, consistant à wrapper en DEVS deux des simulateurs IP les plus répandus dans la communauté scientifique (NS-3 et OMNeT++/INET), ainsi que par la réalisation de cas d'étude industriels et par la proposition d'une démarche concrète pour la conception d'une co-simulation de SCP. Cette thèse a été réalisée dans le cadre du partenariat stratégique entre Inria et EDF R&D, pour le contrat MS4SG (*Multi-Simulation for Smart Grids*).

## 1.3 Organisation du manuscrit

Ce manuscrit est organisé comme suit.

Le Chapitre 1 présente le contexte, avec une introduction aux systèmes cyber-physiques (SCP). Il introduit également le principe des réseaux électriques intelligents, un type de SCP de grande envergure, qui sert de cas d'étude privilégié pour la suite du manuscrit. Les SCP de grande envergure sont comparés aux systèmes complexes, qui ont besoin d'être modélisés et simulés pour en étudier le comportement. Les défis scientifiques sont présentés, en précisant que ce manuscrit se concentre en particulier sur les problématiques liées à l'intégration de modèles et simulateurs de réseaux informatiques (en particulier ceux basés sur le protocole IP), qui sont des organes vitaux des SCP.

Le Chapitre 2 permet ensuite de parcourir les différentes notions utiles dans le cadre de la modélisation et simulation. Les différentes politiques d'exécution d'un modèle sont abordées, et le concept de co-simulation avec un multi-modèle est introduit. La co-simulation est nécessaire pour la simulation de SCP. Le formalisme DEVS, HLA et le standard FMI sont ensuite présentés,

en tant qu'outils pertinents pour la modélisation et simulation de SCP. Enfin, la plateforme de co-simulation MECSYCO est présentée : elle sera au cœur des propositions que nous ferons.

Le Chapitre 3 propose un état de l'art des travaux actuellement réalisés, qui impliquent l'intégration de modèles et simulateurs IP à des co-simulations. Les travaux sont classés selon le mode d'intégration des simulateurs, selon s'ils interagissent avec une autre instance d'eux-mêmes, s'ils interagissent avec une instance d'un autre simulateur IP, ou bien s'il interagissent avec des simulateurs issus d'un autre domaine d'expertise.

Le Chapitre 4 permet de situer les contributions de ce travail de thèse, notamment en explicitant les hypothèses et la proposition retenues. Puis, les Chapitres 5, 6 et 7 présentent les contributions.

Le Chapitre 5 propose des solutions pour concevoir un multi-modèle avec des modèles IP, avec une structuration des différents niveaux de problèmes pour l'intégration de modèles IP dans une co-simulation. Le Chapitre 6 détaille ensuite notre stratégie d'intégration de modèles IP et leurs simulateurs à une co-simulation hybride, en s'appuyant sur la notion de wrapping DEVS.

Enfin, le Chapitre 7 propose une méthode d'évaluation de ce travail, en présentant des exemples d'intégration réussie de deux simulateurs IP réputés dans la communauté scientifique (NS-3 et OMNeT++/INET), ainsi qu'une expérience tendant à prouver expérimentalement que les résultats de simulation ne sont pas impactés par nos méthodes. Ce chapitre présente également des cas d'étude industriels autour des réseaux électriques intelligents et des appartements intelligents. Certains de ces travaux ont été réalisés en partenariat avec EDF R&D, qui a utilisé une partie de nos contributions avec succès. Une démarche à adopter lorsqu'on souhaite co-simuler un SCP avec des réseaux IP en utilisant nos travaux, basée sur cette expérience industrielle, est aussi proposée dans ce chapitre.

# Chapitre 2

# Modélisation et simulation

**Sommaire**



## 2.1 Introduction

La démarche de modélisation et simulation permet d'étudier un système par l'intermédiaire d'une représentation de celui-ci, qu'on appelle un modèle.

L'objectif de ce chapitre est d'expliquer les principes fondamentaux de cette démarche. Après un rappel de la notion d'abstraction inhérente à la création d'un modèle, d'autres notions comme celle de formalisme et de paradigme sont présentées. Ces dernières notions sont autant d'outils nécessaires pour concevoir et créer un modèle, à partir de l'observation d'un système cible.

Ces modèles, qui font finalement l'objet de programmes informatiques opérationnels, sont ensuite exécutés par l'intermédiaire d'un logiciel de simulation, couramment appelé un simulateur. Cette exécution permet d'obtenir des résultats de simulation, qui permettent de tirer des conclusions sur le comportement du système cible à étudier. La notion de simulation est expliquée avec une présentation des principaux modes d'exécution de modèles, qui sont généralement utilisés.

Puisque ce manuscrit de thèse s'intéresse à la modélisation et simulation de SCP, et que ceux-ci imposent d'utiliser différents modèles qui doivent être interconnectés entre eux, ce chapitre introduit ensuite la notion de multi-modèle et de co-simulation. Enfin, différents outils pour la co-simulation, comme le formalisme DEVS, HLA ou le standard FMI sont présentés. La plateforme de co-simulation MECSYCO, basée sur la notion de wrapping DEVS, est enfin présentée et sera utilisée dans le cadre de nos propositions.

## 2.2 Modèle de simulation

### 2.2.1 Abstraction

L'étude d'un phénomène ou d'un objet nécessite de faire des expériences dessus, en le faisant évoluer selon des paramètres internes (changement de ses propriétés, i.e. de son *état*) et externes (changement des propriétés de l'environnement avec lequel il interagit). Le phénomène ou l'objet à étudier est appelé le système cible (qu'on peut également abréger par « système »). Mener les expériences directement sur le système cible peut parfois être dangereux, contraire à l'éthique ou simplement trop coûteux. Ainsi, on va souhaiter mener ces expériences sur des représentations du système cible, plutôt que sur le système lui-même : ces représentations sont appelées des modèles.

Le modèle est une représentation simplifiée d'un phénomène ou d'un objet. D'après [Minsky, 1965], pour un observateur B, un objet A* est un modèle d'un objet A, si A* permet à B de répondre aux questions qu'il se pose sur A. Dès lors, la question n'est pas de savoir à quel point le modèle est fidèle, dans le niveau de détails, au système cible, mais de savoir si le comportement du modèle est suffisamment proche de celui du système cible, dans les limites de l'objet de l'étude. La simplification d'un modèle par rapport au système cible permet également d'être plus à même de déduire des conclusions vis-à-vis des objectifs fixés, sans avoir à être parasité par un ensemble de données liées à l'évolution du modèle, qui ne sont pas intéressantes dans l'étude à mener.

Admettons que l'objet de notre étude soit de déterminer en combien de temps une bouilloire électrique est capable de faire bouillir de l'eau. Le niveau de précision attendu va varier en fonction de l'utilisation qu'on souhaite faire de ces résultats. Si on souhaite une estimation du temps, sans être à la milliseconde près, le modèle n'a par exemple pas besoin d'être complexifié en considérant la perte de chaleur qu'il peut y avoir durant le fonctionnement de la bouilloire. On pourra également considérer que l'énergie nécessaire pour augmenter de 1 °C la température de 1 kg d'eau est de 4,2 kJ/kg/K, bien que cette valeur soit supposée changer légèrement selon la température initiale de l'eau. Ainsi, une simple équation comme $t = 4.2m * (100 - T)/Psecs$ permet de décrire le comportement de la bouilloire, et de déterminer le temps nécessaire pour mettre à ébullition $m$ kg d'eau initialement à $T$ °C, en utilisant une bouilloire de puissance $P$ kW. La constante 100 indique la température à atteindre (en °C) pour que l'eau entre en

ébullition et est largement admise, alors qu'il s'agit également, dans l'absolu, d'une estimation. Selon si le but de la simulation est d'étudier la cinétique des gaz, ou d'étudier le temps moyen pour se préparer un thé, ces approximations n'auront pas la même importance.

Il est également intéressant de considérer que les modèles ne peuvent jamais réellement être comparés aux systèmes cibles eux-mêmes (qu'on pourrait alors appeler « la réalité »), mais à la représentation qu'on se fait de ce système. Ainsi, pour reprendre l'argumentaire de [Quesnel, 2006], l'observation d'une chaise nous permet de décrire la réalité en fonction de ce que notre œil renvoie comme image. Pourtant, si un insecte avait à décrire cette même chaise, avec son procédé de vision propre, sa version de la réalité serait différente. De façon plus générale, [Ferber, 1995] explique que les outils et les connaissances utilisés pour l'observation impliquent un filtrage de la réalité par un modèle. Déterminer si un modèle est suffisamment proche ou non du système cible dépend donc également de l'image que s'en fait l'observateur.

On peut en déduire qu'il n'existe pas de notion de bon ou de mauvais modèle, mais uniquement de capacité ou non de répondre à des questions qu'on se pose au moment de son utilisation (ce qu'on appelle le cadre expérimental [Zeigler et al., 2000]), et d'adéquation des réponses données par le modèle en fonction de celles qu'auraient permis d'obtenir les mêmes expériences sur le système cible lui-même. Il s'agit de la définition du modèle valide proposée par [Minsky, 1965] : A* est un bon modèle de A du point de vue de B, à partir du moment où les réponses apportées par A* sont celles qu'aurait pu observer B directement avec A, pour les questions qui sont importantes pour lui.

### 2.2.2 Paradigme

L'activité de description d'un objet ou d'un phénomène, quelle que soit la manière d'exprimer cette description, suppose systématiquement l'usage d'un certain nombre de lois et d'axiomes permettant à plusieurs personnes de se comprendre et de se mettre d'accord. Ce cadre de pensée correspond à ce qu'on appelle un paradigme. Dans le contexte scientifique, il correspond à la vision scientifique que le modélisateur a choisi d'adopter pour écrire son modèle.

[Kuhn, 1996] explique que les paradigmes ont pour objectif de normaliser une branche, pour servir de base de travail aux scientifiques qui ont choisi de travailler dedans. Ces derniers s'évertueront alors à tester et remettre en cause cette base, en soumettant systématiquement le paradigme à de nouvelles énigmes, leur permettant ainsi de renforcer le cadre paradigmatique. Dès lors qu'une énigme non ou partiellement résolue révèle des anomalies dans le paradigme, des dissidences dans le monde scientifique peuvent aboutir à la formation d'un nouveau paradigme dérivé.

À titre d'exemple, un des paradigmes majeurs en sciences modernes est celui qui concerne l'atome. La communauté scientifique s'est largement mise d'accord pour considérer que la matière de l'univers était composée d'atomes, C'est sur cette base de travail et en exploitant ce concept, que la théorie cinétique des gaz a pu émerger et donner une première explication convaincante à la notion de pression.

Dans le cadre de la modélisation, on peut citer le paradigme multi-agent, qui considère que le monde peut être décrit comme un ensemble d'entités autonomes (des agents) qui interagissent entre elles et avec leur environnement [Ferber, 1997].

Un même paradigme peut être décrit en utilisant plusieurs formalismes différents.

### 2.2.3 Formalisme

Une fois le paradigme choisi pour décrire l'objet et le phénomène, il faut réussir à traduire cette description en utilisant un langage formel. Ce langage formel permettra de décrire le système de manière non-ambiguë et de pouvoir développer un programme informatique correspondant au modèle.

Le choix du formalisme dépend de la nature du système à modéliser : un phénomène physique sur un système électrique pourra généralement être modélisé avec des équations différentielles, alors qu'un phénomène migratoire pour des groupes d'animaux utilisera plutôt la notion d'événements.

La classification des formalismes de É. Ramat (cf. Figure 2.1) indique que le choix du formalisme influence la façon dont seront représentés les changements d'états, le temps et l'espace. Le choix du formalisme va directement contraindre la façon dont le modèle devra être utilisé une fois qu'il sera exécuté sous forme de programme informatique.

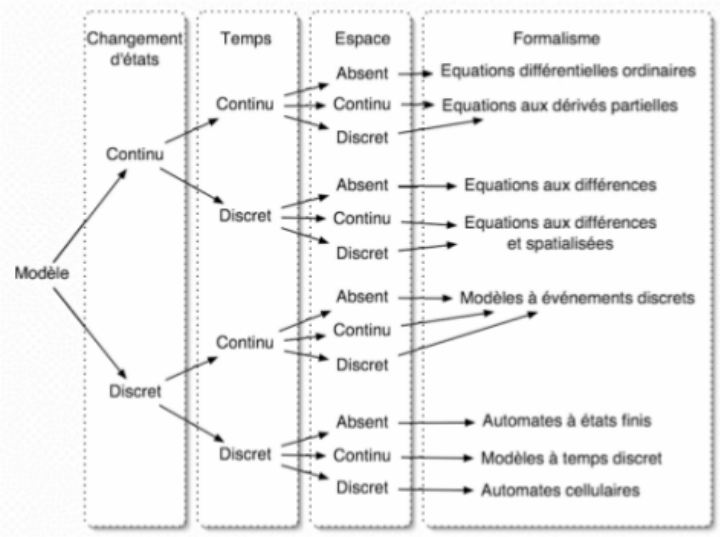



FIGURE 2.1 – Classification des formalismes selon l'aspect continu ou discret des variables, du temps et de l'espace de [Éric Ramat, 2006].

### 2.2.4 Démarche de modélisation

La conception d'un modèle depuis l'étude du système cible, peut se décomposer en plusieurs niveaux de spécification, qui sont empruntés à [Zeigler et al., 2000] :

— **Cadre d'observation (niveau 0) :** Le système cible est étudié en fonction de la question qu'on se pose, pour déterminer les conditions d'observation. Les variables de sortie faisant office d'indicateurs à observer sont identifiées, et la méthode pour les observer dans la durée est déterminée. On identifie également les stimuli appliqués sur le système, qui correspondront à des variables d'entrée.
Exemple : *Si on considère un étang comme système cible et qu'on se demande comment sa température va évoluer dans la journée, on va par exemple considérer qu'il s'agit d'une*

*journée d'été ensoleillée. La hauteur du soleil au cours du temps va influencer la température de l'étang dans la journée, et va donc être considérée comme une variable d'entrée intéressante. Une variable de sortie pertinente est évidemment la température de l'eau, et pourra être mesurée en plongeant un thermomètre dans l'étang, à partir duquel on va régulièrement noter la valeur.*

- **Comportement des entrées et des sorties (niveau 1) :** Des données sont collectées sur le système cible. Elles correspondent à des paires de valeurs indiquant la valeur d'une sortie en fonction d'une valeur d'entrée, associées à un temps précis.
  Exemple : *Dans l'exemple de l'étang, les différentes hauteurs atteintes par le soleil au cours de la journée seront associées à la températures atteinte par l'eau de l'étang au même instant.*
- **Fonctions d'entrées/sorties (niveau 2) :** Certains paramètres du système, et de son environnement, ont une influence durable sur les variables de sortie. On identifie ces paramètres, qu'on regroupe dans ce qu'on appelle un état initial. En fonction de ce que contient celui-ci, les valeurs d'entrée conduiront à produire des variables de sortie avec des valeurs uniques.
  Exemple : *La température de l'eau de l'étang au début de la journée influencera l'ensemble des prises de température de l'eau de la journée, quelle que soit la hauteur du soleil à l'instant de la prise de mesure.*
- **État de transition (niveau 3) :** L'observation des prises de mesure du niveau 1, et la connaissance du système, permettent de déterminer des relations entre les données d'entrée (faisant office de stimuli) et les données de sortie. On peut alors en déduire des raisonnements, qui en fonction des entrées, permettent de produire et donc de prédire les sorties. L'état calculé précédemment (qui sera l'état initial dans le premier cas) entre obligatoirement en compte dans ce raisonnement.
  Exemple : *En fonction de la hauteur du soleil, on pourra déterminer l'influence de son rayonnement sur la température de l'étang. La température actuelle de l'étang pourra être déterminée en fonction de cette information, mais également de sa température actuelle, correspondant à l'état précédent.*

On considère qu'un modèle n'est réellement simulable qu'à partir du moment où il a atteint ce niveau de spécification numéro 3. À ce stade de spécification, on parle de modèle atomique. Un dernier niveau de spécification, portant sur la possibilité d'interconnecter des modèles entre eux, est détaillé dans la Section 2.4.

D'après [Galán et al., 2009], le passage de l'observation d'un système cible à la production d'un modèle numérique exécutable passe par quatre grandes étapes, faisant intervenir différents acteurs (cf. Figure 2.2). Ces différents acteurs peuvent éventuellement être responsables de l'introduction d'erreurs dans le modèle final qui sera utilisé, qui pourront mettre en péril la qualité des résultats obtenus :

- **Le thématicien :** Il s'agit d'un expert du système cible, qui va définir comment représenter le système, en proposant un modèle conceptuel. Selon les choix d'abstraction du système en fonction de la question posée, des erreurs pourront être introduites dans sa description.
- **Le modélisateur :** La description proposée par le thématicien, généralement exprimée dans le langage naturel ou dans le langage métier, va ensuite être formalisée par le modélisateur. Des erreurs d'interprétation du discours du thématicien peuvent le conduire à introduire des erreurs dans la formalisation du modèle.

— **L'informaticien :** Le modèle formel va être traduit sous la forme d'un modèle exécutable, grâce à une notation algorithmique, pour pouvoir ensuite être implémenté. Le passage d'un modèle formel à un algorithme peut être de nouveau l'occasion d'introduire des erreurs.
— **Le programmeur :** Le modèle exécutable va être implémenté, permettant de l'utiliser dans le cadre de simulations numériques. Des erreurs d'implémentation peuvent conduire à de nouvelles erreurs vis-à-vis du comportement décrit par le modèle initial.

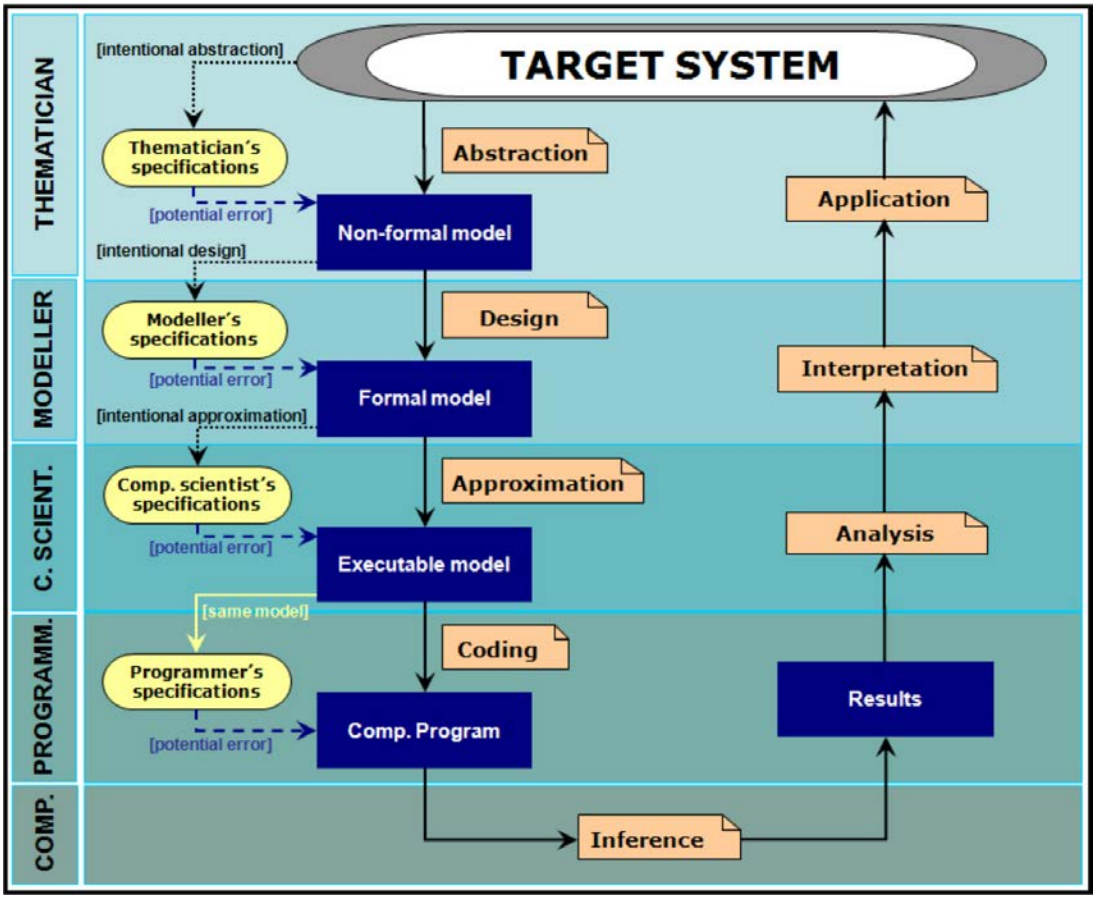



FIGURE 2.2 – Les différentes étapes du processus de modélisation, décrites par [Galán et al., 2009].

La démarche de modélisation et simulation consiste généralement à procéder par essais-erreurs. Ainsi, si les résultats fournis par le modèle final ne sont pas cohérents par rapport à ce qu'on peut constater directement sur le système cible, on tentera de remettre en question chacune de ces étapes, pour obtenir des résultats exploitables.

La section qui suit porte sur l'exécution des modèles, dans le cadre du processus de simulation.

## 2.3 Simulation

### 2.3.1 Exécution d'une simulation

Nous ne considérons que les simulations numériques (*simulations*) dans le cadre de cette thèse.

Exécuter une simulation consiste à exécuter un modèle de simulation. À chaque exécution du modèle, celui-ci maintient un état courant. Cet état est généralement un ensemble de propriétés qu'il a calculé durant sa dernière exécution, et qui ont permis de fournir des résultats de simulation. Ces résultats de simulation permettent de répondre à une question qui concerne le système cible, représenté par le modèle.

Chaque exécution du modèle se base sur son état précédent, et/ou sur des données d'entrée qui lui sont fournies. Les modèles évoluent en général en fonction d'un temps de simulation. Ce temps de simulation, qui fait partie des propriétés du modèle, représente le temps tel qu'il s'écoulerait pour le système cible dans la réalité. Il est à différencier du temps d'exécution, qui correspond au temps nécessaire pour exécuter le modèle. Ainsi, un temps de simulation peut parfaitement laisser s'écouler plusieurs dizaines d'années, pour un temps d'exécution de quelques secondes.

Le modèle de simulation est exécuté par le biais d'un logiciel de simulation (*simulateur*). Le simulateur est généralement un logiciel spécialisé dans la simulation, pour un domaine spécifique. Différents modèles peuvent être exécutés pour un même simulateur. Ce dernier a pour charge de faire évoluer le modèle qu'on lui fournit, en lui faisant calculer des résultats partiels, exécution après exécution, et en faisant évoluer son temps simulé. Le simulateur gère l'horloge représentant le temps simulé, durant la simulation.

Les données collectées au cours du temps, par l'intermédiaire de l'exécution du modèle, sont appelées des résultats de simulation. Ils sont utilisés, éventuellement après une étude approfondie par un expert du domaine, pour tirer des conclusions sur le comportement du système cible qui était à étudier, pour finalement répondre à la question qui était initialement posée. Les résultats de simulation peuvent être accessibles en direct, au fur et à mesure du déroulement de la simulation, ou en post-mortem, lorsque celle-ci est terminée, selon les possibilités offertes par le simulateur utilisé.

La façon dont le simulateur va faire évoluer le temps du modèle dépend fortement du mode d'exécution de celui-ci.

### 2.3.2 Différents modes d'exécution

La classification des types de modèle par formalisme de [Éric Ramat, 2006] (cf. Figure 2.1) permet d'obtenir l'intuition que tous les modèles n'offrent pas les mêmes possibilités en matière de résultats de simulation, et qu'ils ne sont pas conçus pour être exécutés de la même façon.

Cette diversité de modes d'exécution est un facteur important de l'hétérogénéité qu'il peut y avoir entre plusieurs modèles, et doit être prise en considération vis-à-vis des objectifs de ce manuscrit. Les deux catégories de modèles qui sont détaillées ci-dessous sont celles qui sont les plus importantes à étudier dans le cadre de la modélisation et simulation de SCP : les modèles à temps continu, qui proposent des changements d'état qui sont soit discrets, soit continus.

### Changements d'état continus

Un changement d'état continu signifie que, quel que soit le temps de simulation qu'on considère, on pourra toujours potentiellement trouver un résultat de simulation différent, si on interroge le modèle à un temps de simulation légèrement inférieur ou supérieur.

L'exécution d'un modèle qui possède cette caractéristique est soumise au choix de deux pas de temps différents : le pas de résolution et le pas de communication.

**Pas de résolution** D'après Ramat, ces modèles utilisent des formalismes basés sur des équations différentielles ordinaires ou des équations aux dérivés partielles. Ces équations doivent donc être résolues numériquement, et supposent donc l'utilisation d'un pas de résolution. C'est à dire que, au moment de l'exécution du modèle, on va être contraint de choisir, plus ou moins arbitrairement, des temps précis auxquels il faut faire une résolution de l'équation. On pourra par exemple décider de résoudre l'équation toutes les 100 millisecondes, en temps simulé. Le choix du pas de résolution doit correspondre à un compromis entre la précision des résultats de précision (qui dépendent de la nature de l'équation et de la question qu'on se pose sur le modèle), et la performance de la simulation (qui va plus ou moins allonger le temps d'exécution, en fonction du matériel informatique à notre disposition). Ce pas de résolution est censé être conseillé par le modélisateur, et peut varier au cours de la simulation.

**Pas de communication** Le pas de communication intervient lorsque le modèle est couplé à un autre modèle et qu'il doit lui transmettre des résultats au cours de la simulation (ce cas de figure étant plus largement détaillé dans la Section 2.4), et de façon plus générale pour la production de résultats de simulation. Il correspond aux temps auxquels on va décider d'observer l'état du modèle et de faire une copie des variables qui nous intéressent pour répondre à la question qu'on se pose sur le modèle, une fois la simulation achevée. Selon les moments où on décide d'observer cet état, les résultats de simulation peuvent être très différents, et peuvent éventuellement amener à des réponses biaisées. Le choix du pas de communication peut également influencer les performances du modèle, et conduire à des temps d'exécution plus ou moins acceptables. Il s'agit de nouveau d'un compromis entre la qualité des résultats attendus et le temps alloué, en fonction des ressources informatiques dont on dispose, pour l'exécution de la simulation.

La solution la plus simple consiste à décider d'un pas de résolution et d'un pas de communication fixes et identiques. Cette solution est viable tant que les performances à l'exécution restent acceptables, et que le pas unique choisi est validé par le modélisateur. Un exemple est illustré en Figure 2.3.

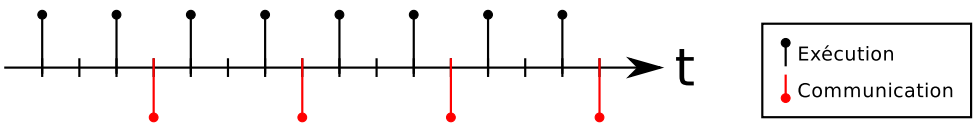



FIGURE 2.3 – Exécution avec des changements d'état continus, à l'aide de pas de résolution et de communication fixes.

Un pas de communication plus petit qu'un pas de résolution n'a aucun intérêt, puisque l'état ne peut pas avoir changé entre deux pas de résolution. Un pas de communication plus grand que

le pas de résolution peut parfois masquer des changements brusques de l'état, qui n'apparaitront pas dans les résultats de simulation. Ainsi, si une variable de l'état a la valeur 1 au dernier pas de communication, qu'elle passe ensuite à la valeur 0, puis 1, d'une résolution à l'autre, et que le pas de communication intervient lorsque la valeur est revenue à 1, les résultats de simulation indiqueront qu'il n'y a pas eu de changement de valeur sur toute la période.

Une solution pour ne pas avoir à faire ce choix, consiste à baser le pas de communication sur la notion d'événement d'état.

**Événement d'état** Plutôt que de transmettre des résultats à chaque résolution de l'équation, ou à des temps choisis de façon arbitraire, il est possible de définir à l'avance un ensemble de conditions. Ainsi, on pourra par exemple déterminer à l'avance que, d'après la question qu'on se pose sur le modèle, connaître les temps correspond à un changement de signe pour une valeur de l'état donnée, est suffisant. À chaque résolution de l'équation, on va donc vérifier la valeur de la variable correspondante : si elle est positive alors qu'elle était négative la fois précédente, ou inversement, on considère qu'il y a un événement d'état, et on ajoute la valeur aux résultats de simulation (cf. exemple en Figure 2.4). Le reste du temps, on ne communique aucun autre résultat. Les possibilités de condition sont infinies, et dépendent entièrement de l'objectif de la simulation.

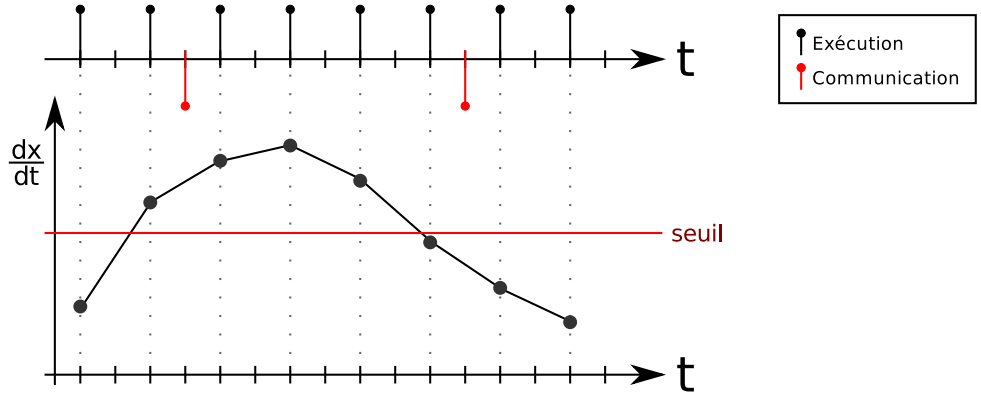



FIGURE 2.4 – Exécution avec des changements d'état continus, à l'aide d'un pas de résolution fixe et une communication basée sur une condition (franchissement du seuil) liée à une variable de l'état ($x$).

Le type d'événement d'état qu'on souhaite observer peut fortement influencer le choix du pas de résolution, et donc la précision de calcul dont on a besoin. Si les résultats de simulation attendus nécessitent d'avoir une résolution la plus fine possible de l'équation, un pas de résolution infiniment petit (toutefois limité à la capacité de précision dans la représentation du temps que peut atteindre le simulateur) doit être utilisé. Puisqu'un tel choix, selon le modèle, aura probablement pour conséquence d'aboutir à des temps d'exécution inacceptables, une solution intermédiaire consiste à utiliser des *rollbacks* (retours en arrière).

**Avec rollbacks** Un rollback consiste à faire avancer un modèle dans le temps, et de finalement lui demander de revenir à l'état correspondant au temps précédent. Le rollback est une fonctionnalité que le simulateur et son modèle peuvent ou non proposer. Ils nécessitent en général

de systématiquement garder une copie intégrale de l'état du modèle, avant de le faire évoluer. Un retour arrière consiste donc à réinitialiser l'état en fonction de cette sauvegarde et à reculer l'horloge du simulateur au temps correspondant. Une seconde stratégie consiste à être capable de demander au modèle de faire des opérations inverses : chaque événement exécuté depuis le temps auquel le simulateur souhaite reculer, jusqu'au temps actuel, doit être annulé un à un en appliquant l'opération inverse.

Une utilisation courante des rollbacks permet en général de faire plus ou moins disparaître la notion de compromis à trouver entre la finesse des résultats et le temps d'exécution. Ainsi, un modèle va pouvoir être résolu avec un pas de résolution fixe la majorité du temps, permettant d'obtenir des performances intéressantes. Quand un changement d'état entre deux résolutions indique qu'il y a eu un changement important dans le modèle, le simulateur va décider de demander au modèle de faire un retour arrière jusqu'au pas de résolution précédent. Puis, il va lui demander de résoudre de nouveau l'équation, avec un pas de résolution beaucoup plus petit. Ainsi, l'équation va être résolue finement lorsque l'état semble beaucoup évoluer, et va être résolue plus grossièrement, lorsque l'état semble stable. Cette méthode permet de détecter les événements d'état au temps le plus proche de leur apparition, plutôt qu'au pas de communication qui suit.

Il est important de remarquer que cette méthode n'est performante que si les changements dans le modèle ne sont pas trop fréquents, auquel cas sinon les rollbacks systématiques auront pour conséquence de très fortement dégrader les performances. Un exemple de recherche des temps approchés associés aux événements d'état est donné dans [Camus et al., 2016], avec une recherche bissectionnelle basée sur une stratégie dichotomique.

### Changements d'état discrets

Un changement d'état discret signifie que le modèle change d'état à des instants déterminés dans le temps. La classification de Ramat réduit la catégorie des modèles à temps continu mais à changements d'état discrets, à celle des modèles à événements discrets.

Les événements représentent les instants où l'état change. Ces événements peuvent intervenir n'importe quand dans l'espace du temps, mais systématiquement à des moments précis. L'état du modèle reste identique à partir du moment où un événement est intervenu, jusqu'à l'occurrence du prochain événement.

Exécuter le modèle consiste donc à demander au simulateur d'aller d'un événement à l'autre, en faisant des bonds plus ou moins grands dans le temps (cf. exemple en Figure 2.5). C'est l'équivalent du pas de résolution des modèles à changements d'état continus, à ceci près que c'est le modèle qui détermine lui-même le temps exact de sa prochaine exécution, correspondant au traitement de l'événement suivant. Cette spécificité permet d'avoir la certitude d'obtenir des résultats de simulation exacts, et de ne pas avoir à lire l'état alors qu'il n'y a eu aucun changement depuis la dernière fois (augmentant ainsi les performances de la simulation).

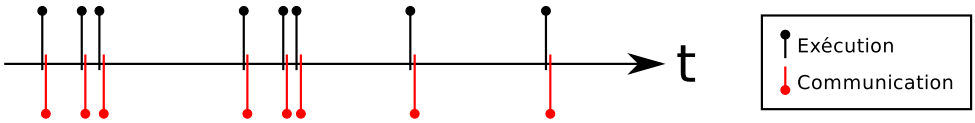



FIGURE 2.5 – Exécution avec des changements d'état discrets.

S'il est possible de calquer le pas de communication sur le temps du traitement des événements, pour obtenir l'intégralité des évolutions de l'état aux temps exacts de leur occurrences, il est également possible de se baser sur des conditions. Ainsi, on peut considérer que seuls certains changements caractéristiques de l'état sont intéressants pour les résultats de simulation, ou pour la transmission à un modèle couplé.

Les différents modes d'exécution abordés dans cette section correspondent à des modèles qui ne sont pas tous capables d'aussi bien décrire un objet ou un phénomène. Lorsqu'il s'agit de décrire plusieurs choses ou phénomènes qui interagissent ensemble, il peut être intéressant d'interconnecter des modèles différents. C'est l'objet de la section suivante.

## 2.4 Couplage de modèles

### 2.4.1 Multi-modèle

Les sciences modernes expliquent le monde et la matière comme des organisations hiérarchiques, qu'on peut toujours redécouper en plus petits éléments qui interagissent entre eux, de l'univers jusqu'aux particules élémentaires. Ainsi, considérer qu'un modèle est composé de sous-modèles interagissant entre eux, comme le système cible est composé de sous-systèmes interagissant entre eux, est tentant. Cette approche permet à la fois de construire un modèle complexe comme un LEGO, et de réutiliser des sous-modèles qui seraient déjà existants, pour décrire certaines parties du système. Il faut alors définir quels sont les composants du modèle, et de quelle façon ils interagissent entre eux.

Un modèle qui n'est pas lui-même décomposable en sous-modèles est appelé un modèle atomique. Un modèle qui est composé de sous-modèles est appelé un modèle couplé. Le modèle couplé correspond alors au dernier niveau de spécification proposé par [Zeigler et al., 2000] (niveau 4). La Figure 2.6 donne un exemple de modèle couplé, composé de trois sous-modèles atomiques. Les sous-modèles disposent de ports d'entrée et de sortie (*In* et *Out*), leur permettant d'échanger des données de simulation entre eux.

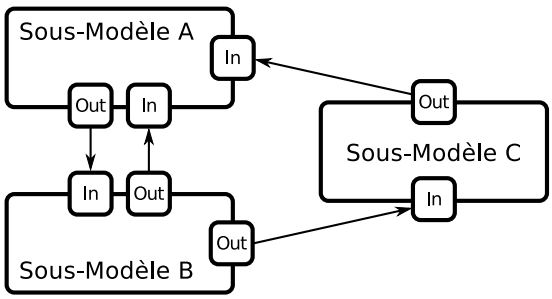



FIGURE 2.6 – Exemple de modèle couplé, composé de 3 sous-modèles atomiques (A, B et C).

Dès lors qu'on considère qu'un modèle couplé est formellement équivalent à un modèle atomique, il est possible d'utiliser la notion de fermeture par composition. Ainsi, un modèle couplé peut être composé à la fois de sous-modèles atomiques et de sous-modèles couplés. On peut ainsi décrire une hiérarchie de modèles à $N$ niveaux, uniquement avec ces deux concepts. La fermeture par composition permet de réduire la complexité d'un modèle, en le définissant par rapport à des composants existants, qui n'ont pas besoin d'être détaillés. Du point de vue de la simulation, les

modèles couplés étant formellement équivalents aux modèles atomiques, il peuvent être utilisés de façon indifférenciée.

Parce qu'il est censé décrire de nombreuses interactions entre de nombreuses entités, un système complexe (cf. Section 1.1.3) peut rarement être décrit à l'aide d'un seul et unique modèle. Ainsi, ce type de système sera généralement représenté à l'aide de différents modèles couplés entre eux, qui peuvent généralement être utilisés de façon indépendante et qui n'ont pas obligatoirement été conçus pour communiquer entre eux. On parlera dans ce cas plutôt de multi-modèle, composé de modèles atomiques et/ou couplés, eux-mêmes composés de sous-modèles atomiques et/ou couplés, et ainsi de suite (cf. exemple de la Figure 2.7).

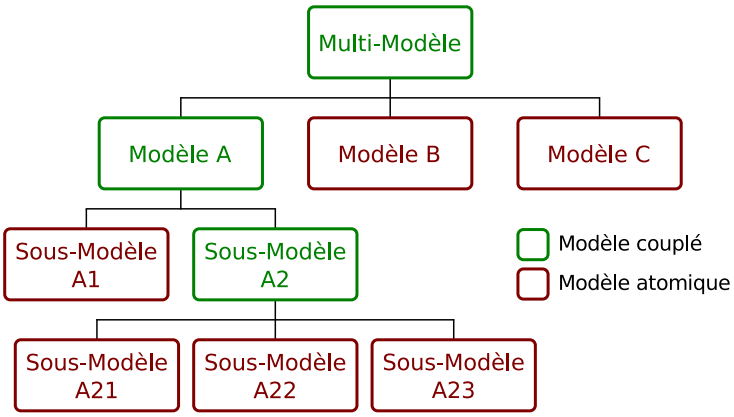



FIGURE 2.7 – Exemple de multi-modèle hiérarchique, composé à la fois de modèles couplés et de modèles atomiques.

À titre d'exemple, la modélisation de l'étang introduite précédemment (cf. Section 2.2.4) pourrait être complexifiée, en introduisant la possibilité d'avoir une éclipse partielle du soleil durant la journée étudiée. L'impact de l'éclipse sur le rayonnement du soleil au cours du temps peut être déterminé en utilisant un modèle déjà conçu et qui est dédié à ce phénomène. Il faudrait alors coupler notre modèle d'étang avec celui-ci, et déterminer comment ils communiquent ensemble. On utiliserait alors désormais un multi-modèle pour étudier l'évolution de la température de l'eau au cours de la journée.

Lorsque les modèles sont de natures différentes, on parle alors de multi-modèle hybride.

### 2.4.2 Multi-modèle hybride

Le choix du mode d'exécution dépend de la nature du modèle.

La Section 2.3.2 a permis de présenter deux grandes catégories de mode d'exécution : à changements d'états continus (principalement avec des modèles équationnels) et à changements d'état discrets (principalement avec des modèles à événements discrets).

Le choix du formalisme dépend fortement de la nature du système cible. Si on souhaite modéliser l'élévation du niveau de la mer au cours du temps, en fonction de la fonte des glaces, il n'y a pas de notion d'événements à proprement parler mais il y a un phénomène continu. Ainsi, on pourra probablement le représenter dans un modèle en s'appuyant sur un formalisme équationnel. Par contre, si on souhaite modéliser le passage de clients à une caisse de supermarché,

on peut supposer que l'arrivée et le départ de chaque client à la caisse sont des événements. Entre deux clients, il n'y a rien à observer. On utilisera par conséquent plutôt un formalisme à événements discrets.

Cependant, selon la question qu'on se pose vis-à-vis du système cible, et donc du niveau d'abstraction souhaité, le choix du formalisme peut être différent. Par conséquent, la modélisation d'un péage autoroutier pourrait parfaitement correspondre à un cas d'utilisation de modèle à événements discrets. Mais si on souhaite simplement déterminer une cadence de passage de voitures, la représentation de celles-ci à l'aide d'une équation différentielle, peut être plus appropriée.

Dans certains cas, le choix d'un formalisme par rapport à un autre peut ne jamais être satisfaisant. L'exemple du péage autoroutier peut donner des résultats satisfaisants avec un modèle équationnel la plupart du temps, et aura l'avantage de permettre l'utilisation d'un modèle simple et potentiellement rapide à exécuter. Cependant, puisqu'il cache la complexité qu'il peut y avoir dans ce genre de système, il ne pourra pas prendre en compte des événements particuliers, comme la panne d'un automate ou un carambolage à l'entrée de l'autoroute, qui pourraient pourtant avoir une importance particulière sur les résultats de simulation. On peut par conséquent souhaiter utiliser un modèle équationnel, représentant ainsi une vue macroscopique du système, et le remplacer ponctuellement par un modèle à événements discrets, pour représenter le système d'un point de vue microscopique, en prenant en compte des événements précis.

Le multi-modèle formé par le couplage de ces différents modèles hétérogènes est appelé un multi-modèle hybride. Interconnecter des modèles de natures différentes dans un même multi-modèle, peut mener à devoir résoudre des problèmes liés à l'hétérogénéité, aussi bien au niveau de la représentation du temps, qu'au niveau de la représentation des données échangées (ainsi que sur d'autres aspects, dont purement logiciels, qui seront détaillés par la suite).

Un multi-modèle est exécuté dans le cadre d'une co-simulation.

### 2.4.3 Co-simulation

La notion de co-simulation intervient dès lors qu'il s'agit de simuler un multi-modèle, composé de modèles différents, qui pourront être exécutés par des simulateurs différents. On dit qu'elle est hybride dès lors qu'elle fait intervenir des modèles qui se reposent sur des formalismes différents.

Puisque les systèmes complexes sont difficiles à décrire (cf. Section 1.1.3), il n'est pas possible de les représenter avec un simple modèle équationnel. La simulation à événements discrets permet de représenter les différentes (ou une partie des) entités de façon atomique, et de modéliser les interactions qu'elles ont entre elles. Ça n'est que lors de l'exécution du modèle, grâce à la simulation et aux résultats de simulation, qu'on pourra comprendre de quelle façon il se comporte d'un point de vue macroscopique.

Dans un système complexe, les entités sont nombreuses et hétérogènes. Selon l'objet de la simulation et le niveau d'abstraction nécessaire, il est peu probable qu'un seul simulateur soit capable de permettre d'écrire un modèle pour représenter toutes celles qui sont à prendre en considération. Cette hétérogénéité se retrouve généralement au niveau du choix des modèles, et peut conduire à devoir utiliser des modèles qui représentent chacun un sous-système avec un formalisme différent. Simuler un système complexe consiste donc généralement à devoir recourir à une co-simulation hybride.

Les différents simulateurs qui participent à la co-simulation ont tous leur propre notion du

temps, et leurs horloges doivent être synchronisées pour qu'ils soient en capacité de s'attendre les uns les autres.

### 2.4.4 Synchronisation du temps

Dans le cadre d'une co-simulation, chaque simulateur participant utilise sa propre horloge. Le temps simulé, pris en considération par les modèles durant la simulation, pourra donc être différent d'un modèle à l'autre, selon le simulateur qui l'exécute. Pour que les modèle puissent s'échanger des données de simulation, ces horloges doivent être synchronisées, notamment pour respecter la contrainte de causalité.

Respecter la contrainte de causalité signifie qu'il faut s'assurer que chaque simulateur (et son modèle) reçoit toujours des données de simulation dans l'ordre croissant du temps. Ainsi, une donnée de simulation qui est associée au temps de simulation $y$ sera obligatoirement reçue après une donnée de simulation associée au temps de simulation $x$, si $y > x$.

D'après [Fujimoto, 2001], il existe deux grandes catégories d'algorithmes de synchronisation du temps, entre simulateurs : optimistes et conservatifs.

Les algorithmes conservatifs assurent à l'intégralité des simulateurs impliqués dans la co-simulation, que la contrainte de causalité ne sera jamais violée. Ainsi, les simulateurs devront généralement arrêter leur exécution, lorsque leur horloge aura atteint un temps simulé à partir duquel il ne peut plus être garanti que d'autres simulateurs ne sont pas susceptibles de lui transmettre des données de simulation associées à ce temps. Une fois que les autres simulateurs auront avancé leur propre horloge interne, et que l'algorithme déterminera qu'il n'y a plus de risque de recevoir de nouvelle donnée dans une certaine tranche de temps simulé, le simulateur pourra reprendre l'avancement de sa simulation jusqu'à une nouvelle date limite.

La solution conservative peut éventuellement imposer aux simulateurs de constamment s'attendre les uns les autres. Selon le type de simulation, et le type de modèles impliqués, cette contrainte peut très fortement ralentir la co-simulation.

Afin d'obtenir de meilleurs temps d'exécution de la co-simulation, les algorithmes optimistes préfèrent prendre le risque de temporairement violer la contrainte de causalité. Les simulateurs pourront ainsi avancer leur horloge, sans garantie de ne pas recevoir ensuite une donnée de simulation qu'il auraient dû intégrer à un temps qui est déjà passé pour eux. Les simulateurs qui utilisent cette stratégie doivent impérativement être capables de résoudre cette situation, en ayant la capacité de faire des rollbacks (cf. Section 2.3.2).

La solution optimiste permet parfois aux simulateurs d'avancer plus rapidement, en n'étant pas obligés de systématiquement s'attendre les uns les autres. Cependant, si les modèles génèrent beaucoup de données de simulation à se transmettre, à des temps très rapprochés, les simulateurs auront fréquemment besoin de faire des rollbacks. Ces derniers pouvant être très coûteux en temps d'exécution, la solution optimiste peut par conséquent parfois être moins performante que la solution conservative, selon les modèles utilisés.

Des plateformes de co-simulation existent, et proposent différentes solutions pour gérer cette synchronisation du temps entre les modèles. La section suivante présente quelques outils utiles pour la co-simulation.

## 2.5 Outils pour la co-simulation

### 2.5.1 Introduction

Cette section a pour objectif de présenter quelques outils qui sont utiles pour mettre en œuvre des co-simulations, de préférence hybrides.

Nous y verrons :

— **DEVS :** Un formalisme permettant d'intégrer tous les autres formalismes, qui est accompagné d'un protocole de simulation. Quelques implémentations, sous forme de plateformes de co-simulation, sont basées sur ce formalisme et son protocole. Cette présentation sera l'occasion d'aborder la notion d'événements internes et externes et de revenir sur les modèles couplés et atomiques en les formalisant.
— **HLA :** Un standard industriel répandu permettant de réaliser des co-simulations. HLA se situe au niveau logiciel et donc principalement au niveau exécution des co-simulations (uniquement composées de modèles événementiels). Cette présentation sera l'occasion d'aborder la notion de lookahead.
— **FMI :** Un autre standard industriel ayant pour but de fournir une interface logicielle universelle, actuellement principalement utilisé avec des modèles de type équationnel.
— **MECYSCO :** Une plateforme permettant de réaliser des co-simulations hybrides. Elle permet de décrire un multi-modèle du niveau formel au logiciel, de l'exécuter d'une façon similaire à celle de HLA, profite de la capacité d'intégration de tous les formalismes de DEVS, et intègre (entre autres) les modèles du standard FMI.

### 2.5.2 DEVS

#### Généralités

DEVS (*Discret EVent System specification*) [Zeigler et al., 2000] est un formalisme de modélisation événementiel.

Comme le démontre [Vangheluwe, 2000a], notamment avec la Figure 2.8, la plupart des formalismes connus peuvent être intégrés à DEVS. Grâce à cette propriété et au principe de wrappeur DEVS [Quesnel, G. et al., 2005] (permettant d'encapsuler un modèle non DEVS, avec des fonctions DEVS), les plateformes basées sur DEVS peuvent potentiellement être capables de co-simuler des multi-modèles hybrides.

#### Événements internes et externes

Deux types d'événéments existent en DEVS : internes et externes.

Les événements internes sont utilisés pour changer l'état du modèle et le faire évoluer dans le temps, de la façon qui est décrite par la dynamique du modèle. Lorsqu'un modèle DEVS est simulé seul en dehors d'un multi-modèle, son exécution correspond simplement à une pile d'événements internes à exécuter séquentiellement jusqu'à ce qu'il n'y en ait plus de planifiés.

Les événements externes correspondent aux entrées et aux sorties des modèles, lorsqu'ils sont intégrés à un multi-modèle. Les événements externes d'entrée arrivent via les ports d'entrée du modèle, et correspondent aux stimuli extérieurs (émis par d'autres modèles du multi-modèle),

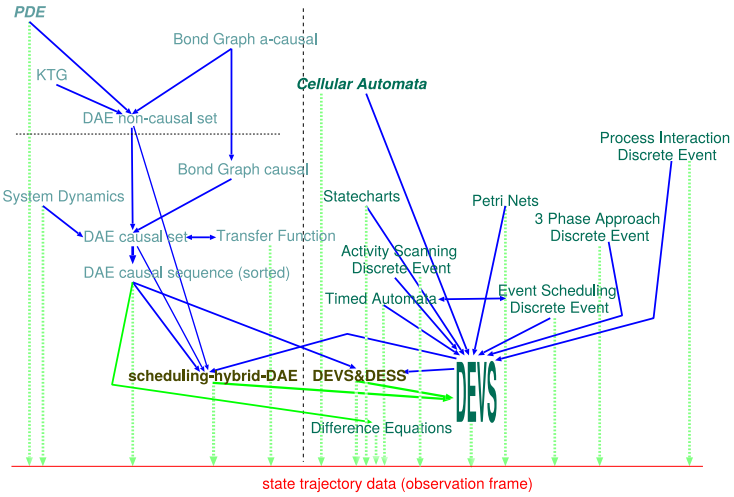



FIGURE 2.8 – Graphe de transformation des formalismes (source : [Vangheluwe, 2000b] à partir de [Vangheluwe, 2000a]). Une flèche bleue indique la possibilité de transformer un formalisme en un autre.

qui viennent influencer l'état du modèle. Ces événements externes d'entrée doivent être exécutés par le modèle dès leur réception, et peuvent éventuellement conduire à l'ajout de nouveaux événements internes dans la pile. À l'inverse, les événements externes de sortie sont des événements produits par le modèle, et envoyés par ses ports de sortie, à destination d'autres modèles du multi-modèle. Un événement externe de sortie est le fruit de l'exécution d'un événement interne.

### Formalisation des modèles

Un modèle DEVS peut être atomique ou couplé (cf. Section 2.4.1).

Un modèle atomique DEVS permet de définir le comportement d'un système, et est formellement défini par l'équation suivante :

$$M_i = (X_i, Y_i, S, \delta_{ext}, \delta_{int}, \lambda, ta) \tag{2.1}$$

pour laquelle :

$X_i = \{(p, v) | p \in InPorts_i, v \in X_i\}$ est l'ensemble des ports d'entrée, avec leurs valeurs respectives ;

$Y_i = \{(p, v) | p \in OutPorts_i, v \in Y_i\}$ est l'ensemble des ports de sortie, avec leurs valeurs respectives ;

$S$ est l'ensemble des variables composant l'état du modèle ;

$\delta_{ext} : Q \times X_i \to S$ est la fonction de transition externe, qui détermine comment le modèle évolue en fonction de la réception d'événements externes d'entrée, et pour laquelle :

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

$e$ correspond au temps écoulé depuis la dernière transition ;

$\delta_{int} : S \to S$ est la fonction de transition interne, qui décrit la dynamique du modèle en exécutant des événements internes ;

$\lambda : S \to Y_i$ est la fonction de sortie, qui détermine si des événements externes doivent être produits, en fonction de l'état du modèle ;

$ta : S \to \mathbb{R}^+_{0,\infty}$ est la fonction d'avancement du temps, qui détermine jusqu'à quand l'état du système restera inchangé, s'il n'y a pas d'événement externe d'entrée qui arrive.

La dynamique d'un modèle atomique DEVS est décrite dans la Figure 2.9.

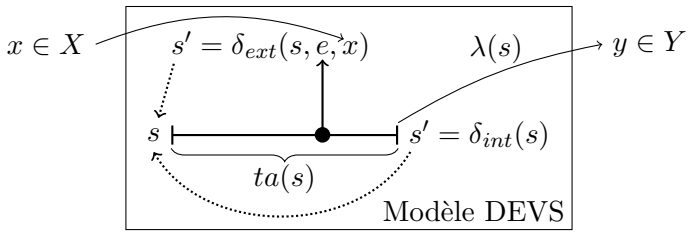



FIGURE 2.9 – Dynamique d'un modèle DEVS, d'après [Zeigler et al., 2000].

Un modèle couplé DEVS permet de définir la structure du système, et est formellement défini par cette équation :

$$N = (X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC) \tag{2.2}$$

pour laquelle :

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

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

$D$ est l'ensemble des identifiants des modèles atomiques ;

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

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

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

[Zeigler et al., 2000] a prouvé qu'en DEVS, les modèles atomiques et les modèles couplés étaient formellement équivalents, permettant ainsi d'exploiter la notion de fermeture par composition (cf. Section 2.4.1). Pour exécuter ces modèles, DEVS propose un protocole de simulation, qui est par conséquent commun aux modèles atomiques et couplés.

### Protocole de simulation

Le protocole de simulation DEVS permet d'exécuter un modèle couplé DEVS. Les fonctions du protocole de simulation DEVS sont présentées en tant que messages, qui transitent entre les modèles et les simulateurs.

Ces messages sont de quatre types différents (illustration avec la Figure 2.10) :

— **Message d'initialisation `(i,t)` :** ordre d'initialiser le modèle au temps `t`.
— **Message de transition interne `(*,t)` :** ordre d'exécuter les événements internes jusqu'au temps `t` (et donc de mettre à jour l'état du modèle).
— **Message d'entrée `(x,t)` :** signalement de l'arrivée d'un événement externe d'entrée, à intégrer au temps `t`.
— **Message de sortie `(y,t)` :** signalement de la disponibilité d'un événement externe de sortie, produit au temps `t`.

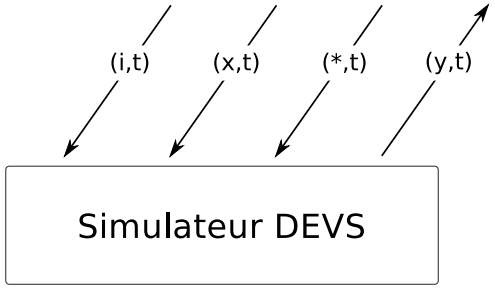



FIGURE 2.10 – Messages associés au protocole de simulation DEVS, d'après [Zeigler et al., 2000].

Le détail du fonctionnement d'un simulateur DEVS et de ses interactions avec les modèles est expliqué dans des publications comme [Camus, 2015], ou directement [Zeigler et al., 2000].

### Limitations

La principale limitation de DEVS est d'offrir un nombre très important de concepts théoriques, qui permettent théoriquement de résoudre un grand nombre de problèmes, mais qui sont parfois trop généraux et donc difficiles à implémenter. Certaines plateformes de co-simulation comme [Quesnel et al., 2009] et [Kim et al., 2009] essaient de palier ce problème, en proposant des cadriciels DEVS basés sur son formalisme et son protocole de simulation.

### Conclusion

L'utilisation du formalisme DEVS pour concevoir des multi-modèles, et de son protocole de simulation pour contrôler des co-simulations, permet de pouvoir se reposer sur un cadre de pensée et de travail complet et rigoureux. La popularité de DEVS permet également d'avoir l'assurance d'avoir à sa disposition de nombreux algorithmes compatibles qui sont prêts à l'emploi.

La capacité de DEVS à pouvoir théoriquement intégrer n'importe quel autre formalisme en fait un candidat de choix pour concevoir une plateforme permettant de définir des multi-modèles hybrides.

### 2.5.3 HLA

#### Généralités

L'objectif du standard industriel HLA (*High Level Architecture*) [Dahmann et al., 1997] est de fournir des spécifications logicielles, permettant d'intégrer des simulateurs existants à des co-simulations. La motivation derrière ce travail est de permettre d'ajouter deux propriétés qui manqueraient à des modèles déjà existants : la réusabilité et l'inter-fonctionnement [Zacharewicz, 2006].

L'ensemble des simulateurs impliqués dans une même co-simulation HLA est appelé une fédération, et chacun de ses simulateurs est appelé un fédéré. Le RTI (*Run-Time Infrastructure*) est le composant qui coordonne le déroulé de la co-simulation, en gérant l'avancement du temps à l'aide d'une horloge centralisée, et en organisant les échanges de données entre fédérés, à l'aide de services.

HLA est défini en fonction de trois principaux composants :

— **Les règles :** détaillent les conditions à respecter pour qu'un simulateur puisse rejoindre une fédération HLA, et pour qu'un logiciel puisse être identifié comme un RTI HLA.
— **L'OMT (*Object Model Template*) :** standardise la description des échanges possibles entre les fédérés.
— **Spécification d'interface :** détermine les services proposés par le RTI aux fédérés.

Afin de donner un aperçu général des possibilités de HLA, les types de services de la spécification d'interface sont décrits ci-après.

#### Services proposés aux fédérés

Les services HLA sont proposés par le RTI, et sont disponibles pour tous les fédérés. Un fédéré communique avec le RTI par l'intermédiaire de ses services, en utilisant un ambassadeur (sorte de connecteur logiciel). Un simulateur ne peut rejoindre une fédération HLA que s'il dispose de l'implémentation d'un ambassadeur, qui lui est spécifique.

Les services définis par le standard sont les suivants :

— **Gestion de la fédération :** Permet de créer, rejoindre ou détruire une fédération. Lorsqu'un fédéré se connecte à un RTI, il indique la fédération qu'il souhaite rejoindre, déterminant ainsi la co-simulation à laquelle il participera.
— **Gestion des déclarations :** Permet d'annoncer les classes d'objet et d'interaction, pour lesquelles les fédérés souhaitent jouer un rôle de publieur (production de données à échanger) ou d'abonné (réception des données, lorsqu'elles sont publiées par d'autres fédérés).
— **Gestion des objets :** Permet de créer, de mettre à jour ou de supprimer des objets ou des interactions, en accord avec les déclarations qui ont été faites (cf. point précédent).
— **Gestion de la propriété :** Permet de déterminer qui est le propriétaire actuel d'un objet ou d'une interaction, de façon à éviter les modifications concurrentes.
— **Gestion de la distribution de données :** Permet d'optimiser les échanges entre fédérés, en affinant le système de publication/souscription de la gestion des déclarations.
— **Gestion du temps :** Permet de faire avancer le temps des fédérés en fonction de l'horloge du RTI, et du type de synchronisation qui a été choisi (e.g. conservatif ou optimiste, cf.

Section 2.4.4).

Si le standard n'impose pas le fonctionnement de l'algorithme de synchronisation du temps, [Dahmann et al., 1997] introduit la notion de lookahead pour les algorithmes conservatifs, qui est à configurer pour chaque modèle.

### Utilisation d'un lookahead

Le lookahead est la durée pendant laquelle un fédéré garantit qu'il ne produira pas de données à transmettre à d'autres fédérés, dans le cadre d'une synchronisation du temps de type conservatif (cf. Section 2.4.4). Cette valeur obligatoirement positive, est indispensable pour permettre au RTI de respecter la contrainte de causalité durant la simulation. Au niveau logiciel, cette contrainte impose qu'un fédéré ne puisse pas recevoir une donnée de la part d'un autre fédéré, qui soit estampillée à une date inférieure à celle où son simulateur considère qu'il est déjà.

À un instant $t_1$, pour chaque fédéré *f*, le RTI utilise le lookahead annoncé par tous les fédérés qui sont susceptibles d'envoyer des données à *f*, pour calculer le LBTS de *f*. Le LBTS (*Lower Bound Time Stamp*) correspond à la date $t_2$ ($t_2 > t_1$) jusqu'à laquelle un simulateur peut faire avancer le temps de son modèle, sans risquer de recevoir ensuite une donnée qui lui ferait violer la contrainte de causalité. Les valeurs de lookahead sont régulièrement transmises par les fédérés, sous forme de *messages nuls*.

Le lookahead est spécifique au modèle, et peut être calculé de façon statique et constante pour toute la co-simulation, ou évoluer au cours de son avancée. Devoir systématiquement évaluer le lookahead d'un modèle peut être contraignant et parfois difficile à accomplir. Choisir un lookahead trop grand peut faire échouer la co-simulation, parce qu'une violation de la contrainte de causalité aura été détectée. Mais choisir un lookahead trop bas, par sécurité, a pour conséquence de ralentir toute l'exécution de la simulation. Dans ce cas, les fédérés sont en effet contraints d'avancer très lentement, en échangeant énormément de messages nuls, afin de se synchroniser très régulièrement.

Un exemple de choix facile de valeur d'un lookahead concerne les réseaux IP (illustration en Figure 2.11). Admettons qu'un modèle IP soit utilisé dans une co-simulation pour simuler le transfert de données d'un ordinateur à l'autre, qui ne sont reliés que par un lien simple. Le modèle fonctionne ainsi : les données à envoyer sur le réseau simulé proviennent de la fédération, leur transfert d'un ordinateur est simulé dans le modèle, et elles sont renvoyées dans la fédération. Dès lors que des données ont été reçues par le modèle IP, le simulateur est en capacité d'affirmer qu'il ne produira pas de données avant le temps correspondant à la fin du transfert des données sur le réseau simulé. Si le lien simulé est configuré dans le modèle pour utiliser des cartes réseau avec un délai de 2 ms, alors le temps de transfert ne pourra pas être inférieur à 4 ms. Ainsi, le lookahead du modèle peut être statiquement configuré à 4 ms, en permettant de modéliser n'importe quel transfert de données, sans jamais risquer de violer la contrainte de causalité.

Un lookahead qui a la valeur zéro est qualifié de nul. Ce cas correspond à la possibilité pour un modèle de simuler des actions instantanées, c'est à dire qu'il estime que le temps qui aurait été nécessaire dans le système pour effectuer le traitement de la donnée qui est reçue, est infinitésimale (ou simplement d'une échelle de temps inférieure à ce qu'il est capable de représenter). Dans le cadre des algorithmes conservatifs, les lookaheads nuls sont généralement incompatibles et donc proscrits.

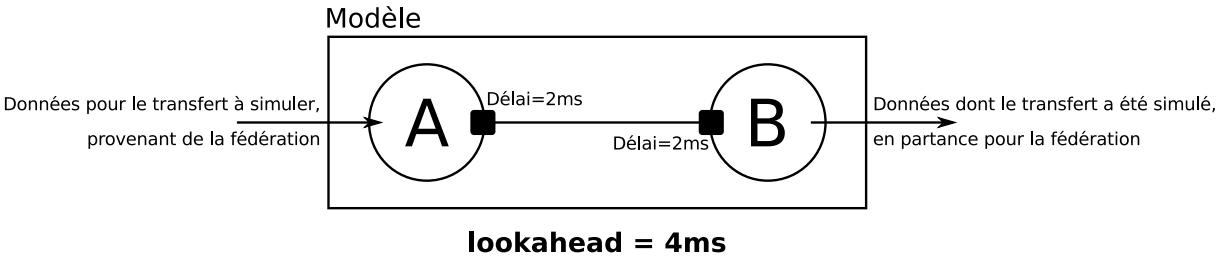



FIGURE 2.11 – Exemple de détermination de la valeur d'un lookahead, à partir d'un modèle IP simple.

### Limitations

Malgré la popularité de HLA, trois principales limitations peuvent être observées d'après [Camus, 2015] :

1. Les spécifications HLA détaillent le fonctionnement général d'un RTI, mais aucunement de son implémentation. Par conséquent, très peu de RTI sont développés de la même façon, et la compréhension de leur fonctionnement par rapport aux spécifications du standard peut être problématique. Ils sont également liés à un langage de programmation, et éventuellement à un système d'exploitation particuliers. De plus, le standard est très ambitieux et très complexe à implémenter entièrement. Par conséquent, la plupart des implémentations de RTI HLA offrent un nombre limité de services. Pour ces raisons, les ambassadeurs des simulateurs sont souvent développés pour une implémentation de RTI en particulier, et ne sont pas compatibles avec les autres versions disponibles. L'interopérabilité entre les simulateurs « compatibles HLA » n'est donc pas assurée.
2. Les interactions qui peuvent avoir lieu dans le multi-modèle sont uniquement définies au niveau logiciel, et donc au niveau syntaxique. Par contre, les interactions ne sont pas définies au niveau sémantique. Deux fédérés qui utilisent le même service HLA pourront donc être en mesure d'échanger des données, sans pour autant être capables de correctement les interpréter. Deux fédérés conçus séparément pour être interopérables via HLA, pourront donc ne pas être interopérables entre eux.
3. HLA n'étant pas capable de représenter formellement un multi-modèle, en se plaçant uniquement au niveau de son exécution, il ne peut pas apporter de réponse viable pour permettre l'exécution d'un multi-modèle composé de modèles utilisant des formalismes hétérogènes. Son utilisation est alors limitée aux formalismes événementiels.

### Conclusion

HLA est un standard incontournable, particulièrement utilisé dans l'industrie. Il est pertinent en tant que cadre de travail, pour réaliser l'intégration logicielle de simulateurs existants, et pour décrire le fonctionnement d'une co-simulation. Par contre, sa limitation aux formalismes événementiels ne lui permet pas d'exécuter des multi-modèles hybrides.

Quelques projets, comme [Zeigler, B. P. et al., 1998, Zacharewicz, 2006], proposent de lever cette limitation en combinant les capacités de HLA au niveau de l'exécution des multi-modèles,

avec les possibilités de DEVS au niveau de l'intégration de formalismes hétérogènes (cf. Section 2.5.2).

### 2.5.4 FMI

#### Généralités

Le standard FMI (*Functional Mockup Interface*) [Blockwitz et al., 2012] propose des spécifications logicielles permettant d'uniformiser la description des modèles et de définir leurs ports d'entrée et de sortie. Lorsqu'il est utilisé pour faire de la co-simulation, il permet également de standardiser la gestion de l'évolution du temps de simulation, à l'exécution du modèle. Ces spécifications ont pour but de rendre les modèles interopérables entre eux et de pouvoir ainsi éventuellement les utiliser au sein d'une même co-simulation. Les modèles deviennent alors des FMU (*Functional Mockup Unit*), qui peuvent rejoindre un master de co-simulation.

L'intégration d'un modèle à un master de co-simulation nécessite que le logiciel de modélisation utilisé soit capable de l'exporter au format FMU. La section suivante présente les deux types de FMU qui sont proposés par le standard.

#### Le format FMU

Concrètement, une FMU correspond à une archive qui contient à la fois un fichier XML, et une bibliothèque C partagée au format binaire. Le fichier XML contient la liste de tous les ports du modèle avec leur identifiant, en précisant le type de donnée (e.g. entier, double ou chaîne de caractères) auquel ils sont associés et s'il s'agit d'un port d'entrée ou de sortie. Un port d'entrée permet d'autoriser la modification de la valeur de la variable correspondante dans le modèle, tandis qu'un port de sortie ne permet que la consultation de sa valeur.

Le rôle de la bibliothèque C dépend du type de FMU dont il s'agit :

— **FMU pour l'échange de modèles :** la bibliothèque C ne contient que le modèle, c'est à dire des fonctions permettant de faire évoluer un ensemble de variables d'état en fonction des valeurs données aux ports d'entrée, et de fournir des valeurs aux ports de sortie. Elle devra donc être exécutée avec un solveur extérieur et adapté à la nature du modèle. Les modèles utilisés peuvent être discrets, continus, ou hybrides, selon le type de solveur utilisé.
— **FMU pour la co-simulation :** la bibliothèque C contient le modèle et le solveur qui permet de l'exécuter. Le modèle intègre donc également une horloge interne, et toutes les fonctions nécessaires pour le simuler. Dans ce cas, les modèles utilisés doivent être de type continu, et l'exécution de la simulation nécessitera d'utiliser des pas de communication (cf. Section 2.3.2).

#### Limitations

La principale limitation du standard FMI est qu'il propose une solution pour décrire des modèles discrets, mais pas de solution pour les simuler. De plus, le standard ne donne aucune indication concernant la réalisation d'un master de co-simulation ou la gestion de l'hétérogénéité au sein d'un multi-modèle.

#### Conclusion

FMI est un standard particulièrement intéressant pour rendre interopérables entre eux des modèles qui ont été créés avec des outils de modélisation différents, qui sont tous les deux capables de réaliser des exports au format FMU. Il est également beaucoup utilisé dans l'industrie, pour sa capacité à permettre d'utiliser un modèle tout en cachant son code source dans le binaire de la bibliothèque partagée. Ainsi, les FMU deviennent des boîtes noires capables de limiter la divulgation d'un secret industriel.

Cependant, l'absence de spécifications pour intégrer des modèles discrets à un master de co-simulation FMI, ne permet pas d'envisager de co-simuler des multi-modèles hybrides.

### 2.5.5 MECSYCO

#### Généralités

MECYSCO (*Multi-agent Environment for Complex SYstems CO-simulation*) est une plateforme de co-simulation réalisée par [Camus, 2015], en se basant sur sa prédécesseuse AA4MM (*Agents & Artefacts for Multi-Modeling*, [Siebert, 2011]). Il est particulièrement adapté pour la simulation de systèmes complexes.

La plateforme MECSYCO propose différents concepts et outils permettant de décrire un système, et de le simuler comme un ensemble de modèles (et donc de simulateurs) en interaction. Ainsi, chaque couple modèle/simulateur correspond à un agent, et les données échangées entre les simulateurs correspondent aux interactions qui ont lieu entre les agents. L'originalité de MECSYCO par rapport aux autres approches multi-modèles à base d'agents, est de ne pas considérer les interactions de façon directe, mais en se reposant sur le paradigme Agents et Artéfacts (A&A) de [Omicini et al., 2008, Ricci et al., 2007].

#### Fonctionnement général

Dans le paradigme A&A, les artéfacts prennent en charge les interactions entre les modèles et les processus externes, et les expriment indépendamment des fonctions internes des modèles. Par conséquent les problèmes d'interopérabilité sont gérés par les artéfacts eux-mêmes. De la même façon, les problèmes de correspondance entre les différents modes de représentation des données ou du temps, sont gérés comme des services de transformation, proposés par les artéfacts qui sont en charge d'assurer les interactions entre les modèles concernés.

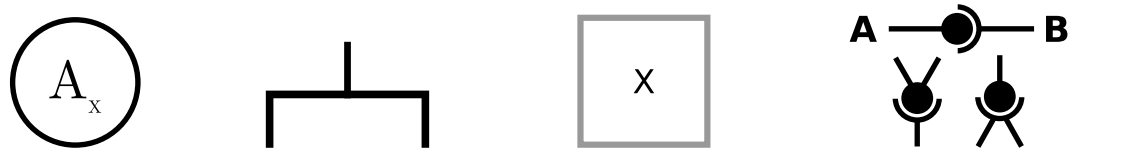
MECSYCO propose une approche méta-modèle basée sur la métaphore multi-agent, pour décrire des multi-modèles hétérogènes. Les concepts multi-agent de MECSYCO ont une représentation graphique propre, qui est associée à des contraintes sémantiques et syntaxiques qui garantissent des descriptions non ambiguës. Ces concepts sont formalisés à l'aide des spécifications opérationnelles de DEVS (cf. Section 2.5.2) permettant de produire un multi-modèle exécutable directement à partir de sa représentation graphique. MECSYCO bénéficie donc également de tous les avantages de DEVS en terme d'intégration de formalismes hétérogènes, et s'appuie sur la notion de wrappeur DEVS [Quesnel, G. et al., 2005] pour intégrer n'importe quel modèle non DEVS au multi-modèle.

Seule une implémentation de MECSYCO en Java existe [4], et est disponible librement en ligne [5] sous licence AGPL v3.

### Concepts

MECSYCO s'appuie sur quatre concepts pour décrire un multi-modèle.

Un modèle $m_X$ est une représentation partielle du système cible, implémentée dans un simulateur (symbole en Figure 2.12c). Il peut avoir un ensemble de ports d'entrée et de sortie. Les modèles correspondent aux parties pré-existantes qu'on souhaite interconnecter, pour construire un multi-modèle. Un m-agent $A_X$ gère un modèle $m_X$ et a pour charge de régir ses interactions avec les autres modèles (symbole en Figure 2.12a). Le comportement des m-agents est spécifié par le protocole de simulation DEVS, afin de permettre leur bonne coordination. Un artéfact de modèle $I_X$ réifie les interactions entre un m-agent de simulation $A_X$ et son modèle de simulation $m_X$ (symbole en Figure 2.12b). Une interaction de $A_A$ vers $A_B$ est réifiée par un artéfact de couplage $C_B^A$ (symbole en Figure 2.12d). Un artéfact de couplage a deux rôles : pour $A_A$, il s'agit d'un artéfact de couplage de sortie, alors que pour $A_B$, c'est un artéfact de couplage d'entrée. Les artéfacts de couplage peuvent transformer les données qu'ils transportent, qui sont échangées entre les modèles, en utilisant des opérations de transformation. Ces opérations permettent également d'effectuer des conversions d'unité de temps sur les événements externes échangés.




(a) M-agent $A_X$. (b) Artéfact de modèle $I_X$. (c) Modèle $m_X$. (d) Artéfact de couplage $C_B^A$ avec A transmettant à B (en haut), et en versions multiplexeur (en bas).

FIGURE 2.12 – Symboles des composants MECSYCO pour décrire un multi-modèle.

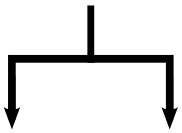
MECSYCO dispose également d'une plateforme d'observation en parallèle, qui peut être raccordée à un multi-modèle MECSYCO pour organiser la récolte et la visualisation des résultats de simulation. Le raccord entre les deux est possible grâce à l'utilisation de concepts similaires à ceux de MECSYCO, mais dédiés à l'observation (symboles en Figure 2.13). Un agent d'observation a notamment comme particularité de ne pas pouvoir être connecté à un artéfact de couplage de sortie (i.e. le modèle d'observation associé peut recevoir des données en entrée mais ne peut pas en produire pour la co-simulation).

Dans l'implémentation de MECSYCO, la coordination entre les modèles est faite de façon purement décentralisée, grâce au comportement des m-agents. Ce comportement correspond à celui d'un simulateur DEVS parallèle et conservatif, basé sur l'algorithme de Chandy-Misra-Bryant (CMB) [Chandy and Misra, 1979]. Le détail de l'intégration de l'algorithme CMB avec le paradigme multi-agent de MECSYCO est donné dans [Camus et al., 2015]. Il se base notamment

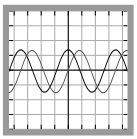
4. La suite de ce manuscrit indiquera que l'une de nos contributions est d'avoir proposé une seconde implémentation de MECSYCO, écrite en C++.

5. `http://mecsyco.fr`

(a) M-agent d'observation.



(b) Artéfact de modèle d'observation.



(c) Modèle d'observation.

FIGURE 2.13 – Symboles des composants de la plateforme d'observation de MECSYCO.

sur l'utilisation d'un lookahead (cf. Section 5.4.3), qui doit obligatoirement être non nul, et qui est à configurer pour chaque artéfact de modèle, selon le modèle utilisé. D'un point de vue logiciel, les communications entre machines sont assurées par un intergiciel de communication.

Les artéfacts de modèle ont été conçus pour pouvoir être utilisés en tant que wrappeurs DEVS. Cette fonctionnalité permet théoriquement d'intégrer n'importe quel simulateur, avec sa bibliothèque de modèles.

#### Intégration d'un simulateur non DEVS

Un modèle peut être intégré à un multi-modèle MECSYCO uniquement si son simulateur a déjà été intégré auparavant. Si c'est le cas, un artéfact de modèle pour ce simulateur, permettant d'intégrer ses modèles, est disponible et peut être directement utilisé (à condition toutefois que le type de couplage souhaité soit similaire à celui pour quoi il a été réalisé). Sinon, il est nécessaire de créer l'artéfact de modèle adéquat.

Un artéfact de modèle étant un wrappeur DEVS, celui-ci implémente des fonctions similaires à celles du protocole de simulation DEVS (cf. Section 2.5.2). D'après les spécifications MECYSCO proposées par [Camus, 2015], chaque m-agent $\mathcal{A}_i$ peut manipuler $m_i$ comme un modèle atomique DEVS, en utilisant les primitives logicielles du protocole de simulation DEVS proposées par son artéfact de modèle $\mathcal{I}_i$. Ainsi, les fonctions listées ci-dessous doivent donc être définies dans chaque artéfact de modèle :

- $init()$ initialise le modèle $m_i$, c'est à dire démarre le logiciel de simulation du modèle, fixe les paramètres de ce dernier et son état initial.
- $processExternalInputEvent(ein_i, t_i, x_i^k)$ exécute l'événement externe d'entrée $ein_i$ arrivant au temps de simulation $t_i$ dans $x_i^k$, le $k$-ième port d'entrée de $m_i$.
- $processInternalEvent(t_i)$ exécute l'événement interne du modèle $m_i$ planifié au temps de simulation $t_i$.
- $getExternalOutputEvent(y_k^i)$ retourne $eout_i^k$, l'événement externe de sortie présent dans le $k$-ième port de sortie de $m_i$, $y_k^i$ (ou *null* si aucun événement de sortie n'est présent dans ce port).
- $getNextInternalEventTime()$ retourne le temps du prochain événement interne de $m_i$.

#### Limitations

La principale limitation de MECSYCO est son nombre actuellement réduit de simulateurs déjà intégrés. Intégrer un nouveau simulateur IP à MECSYCO nécessite d'écrire son artéfact de modèle, et donc de réussir à calquer ses propres concepts sur ceux de DEVS, tout en parvenant

à implémenter les fonctions du protocole de simulation DEVS au niveau de son simulateur. Pour l'instant, aucun simulateur de réseaux IP n'a été intégré à MECSYCO.

**Conclusion**

Grâce à l'utilisation du paradigme A&A et du formalisme DEVS, combiné avec la notion de wrappeur DEVS, MECYSCO est une plateforme de co-simulation particulièrement âpte à intégrer la plupart des formes d'hétérogénéité possibles.

Ainsi, la plateforme permet :

1. d'intégrer des modèles existants à un même multi-modèle, et d'utiliser les simulateurs pour lesquels ils ont été implémentés dans une même co-simulation ;
2. de faire interagir ces modèles (et leurs simulateurs) ensemble, pour leur faire échanger des événements externes ;
3. de passer d'une représentation graphique contrainte du multi-modèle, à une version exécutable.

Elle est particulièrement adaptée pour l'exécution de multi-modèles hybrides, puisque ses concepts lui permettent d'intégrer de nombreuses formes d'hétérogénéité, tant au niveau des modèles que de leur environnement logiciel :

— utilisation de formalismes différents ;
— représentation du temps et des données différentes ;
— utilisation de simulateurs différents et non interopérables entre eux ;
— utilisation de modèles écrits avec des langages de programmation différents ;
— exécution sur des plateformes différentes (i.e. systèmes d'exploitations variés).

Grâce à son exécution purement décentralisée, la plateforme permet également de réduire les temps d'exécution des co-simulations, en répartissant l'exécution des modèles sur des machines différentes.

## 2.6 Conclusion

Ce chapitre a été l'occasion de parcourir les différentes notions indispensables pour pouvoir appréhender la démarche de modélisation et simulation.

Nous avons notamment pu passer en revue les deux principaux modes d'exécution d'une simulation, avec des modèles à changements d'état continus et des modèles à changements d'état discrets. Nous avons vu que ces deux modes d'exécution nécessitaient une gestion de l'avancement du temps différentes, et utilisaient tous les deux la notion de pas de communication, qui détermine quand l'état du modèle va être consulté, pour compléter les résultats de simulation. Ces deux modes d'exécution sont à prendre en considération dans le cas de la modélisation et simulation de SCP : les modèles généralement utilisés pour représenter des comportements physiques sont à changements d'état continus (en utilisant un formalisme équationnel), tandis que les modèles généralement utilisés pour représenter les réseaux IP sont à changements d'états discrets (en utilisant un formalisme à événements discrets).

Nous avons également abordé la notion de multi-modèle, exécutable dans le cadre d'une co-simulation, et permettant de relier des modèles existants entre eux, pour représenter un système

complet. Représenter un SCP complet nécessitant d'utiliser différents modèles issus de différentes communautés, modéliser un SCP consiste généralement à co-simuler un multi-modèle. Puisque ces modèles utilisent différents formalismes, le multi-modèle et sa co-simulation sont qualifiés d'hybrides. Dès lors, des problèmes liés à l'hétérogénéité des modèles apparaissent, à la fois pour permettre aux modèles d'échanger des données de simulation compréhensibles par l'un et l'autre, et pour exécuter la simulation en utilisant des horloges parfaitement synchronisées (qui permettent de toujours respecter la contrainte de causalité, c'est à dire l'ordre d'arrivée des données de simulation échangées).

Différents outils pour la co-simulation ont été présentés. Notamment le formalisme DEVS, qui permet d'obtenir un « langage commun » entre modèles utilisant des formalismes différents. Le protocole de simulation de DEVS permet d'exécuter un multi-modèle DEVS, et la notion de wrappeur DEVS permet théoriquement d'intégrer formellement et logiciellement n'importe quel modèle existant à un multi-modèle DEVS.

Enfin, nous avons détaillé le fonctionnement de la plateforme de co-simulation MECSYCO, conçu pour la simulation de systèmes complexes. Par l'intermédiaire de wrappeurs DEVS et d'artéfacts de couplage, MECSYCO permet de résoudre la plupart des problèmes d'hétérogénéité rencontrés avec un multi-modèle hybride. Cependant, pour qu'un modèle puisse être intégré au multi-modèle, il faut soit que celui-ci utilise le formalisme DEVS, soit qu'il dispose de son propre wrappeur DEVS. L'environnement logiciel de MECSYCO propose notamment un wrappeur DEVS pour FMI, permettant d'intégrer n'importe quel modèle équationnel au format FMU à ses multi-modèles.

Dans ce manuscrit de thèse, nous nous intéressons en particulier à l'intégration des modèles IP (composants essentiels des SCP) à des multi-modèles hybrides. Le prochain chapitre s'intéresse donc aux travaux qui ont déjà été menés, qui impliquent l'intégration d'au moins un modèle IP à un multi-modèle co-simulé.

# Chapitre 3

# Simulateurs IP et co-simulations

**Sommaire**



## 3.1 Introduction

La littérature apporte de nombreux exemples d'utilisation de simulateurs IP dans le cadre de co-simulations. Nous verrons dans ce chapitre qu'il existe plusieurs façons d'intégrer leurs modèles à un multi-modèle co-simulé, avec des modes d'interaction avec les autres modèles, qui peuvent différer d'un cas à l'autre. Ces travaux permettent d'obtenir un bon aperçu des différentes problématiques qui sont soulevées dès lors qu'on souhaite créer une co-simulation qui intègre des simulateurs de réseaux IP.

Les travaux présentés dans ce chapitre sont classés en trois principales catégories :

1. Distribution et parallélisation de simulateurs IP : dans cette première catégorie, les co-simulations sont exclusivement composées de simulateurs IP. Les différents simulateurs IP correspondent dans ce cas à plusieurs instances d'un même simulateur, permettant une distribution dudit simulateur et donc la parallélisation de l'exécution de ses modèles.
2. Interconnexion de simulateurs IP hétérogènes : les co-simulations faisant parties de cette seconde catégorie sont également exclusivement composées de simulateurs IP. Cette fois-ci, les différents simulateurs IP impliqués correspondent à logiciels différents. Des problématiques liées à l'intégration de l'hétérogénéité et au souci d'interopérabilité s'ajoutent par rapport à la catégorie précédente.
3. Interconnexion de simulateurs IP avec des simulateurs d'autres domaines : les co-simulations de cette dernière catégorie sont composées à la fois de simulateurs IP et de simulateurs issus d'autres domaines d'expertise. Le mode d'interaction entre les simulateurs est différent des deux autres catégories et les problématiques liées à l'hétérogénéité sont accrues.

Nous considérons dans ce chapitre que les travaux concernant le couplage de simulateurs IP avec des humains, du matériel ou des logiciels non-simulés sont hors sujet, bien qu'ils pourraient légitimement faire l'objet d'une quatrième catégorie. Dans la suite de ce manuscrit, nous nous intéresserons en particulier à la deuxième et la troisième catégorie. Toutefois, la première catégorie pouvant être considérée comme un cas particulier de la seconde, avec un sous-ensemble des problématiques induites qui sont similaires, les travaux qui y sont liés nous intéressent également dans le cadre de cet état de l'art.

Avant de passer à la présentation des travaux concernant les simulations IP, ce chapitre est introduit par un rapide aperçu des différentes notions liées aux protocoles Internet (*suite TCP/IP*) qui seront requises pour comprendre les différents travaux présentés dans ce manuscrit.

## 3.2 Suite des protocoles Internet (TCP/IP)

### 3.2.1 Introduction

Le but de cette section est de donner un aperçu rapide de l'essentiel des notions relatives aux réseaux IP qui doivent être comprises, pour comprendre les travaux qui seront présentés dans ce manuscrit de thèse.

Pour obtenir plus d'informations sur les protocoles qui seront cités, une référence à la norme sera chaque fois fournie. Elle peut être désignée sous la forme d'un numéro de RFC (*Request For Comments*) produit par l'IETF (*Internet Engineering Task Force*), ou de référence aux travaux de l'IEEE (*Institute of Electrical and Electronics Engineers*) ou bien de l'IEC (*International Electrotechnical Commission*) ou encore de l'ISO (*International Organization for Standardization*).

La suite des protocoles Internet désigne un ensemble de protocoles qui sont utilisés dans les réseaux informatiques modernes, et notamment sur Internet. Elle est souvent désignée sous le nom *suite TCP/IP*, d'après le nom de ses deux premiers protocoles : IP et TCP. Cette suite introduit la notion de modèle TCP/IP, utilisé pour décrire et catégoriser ses différents protocoles. Le modèle TCP/IP a été structuré en couches par l'IETF, dans la RFC 1122.

Dans la suite de ce manuscrit, nous parlerons de « structure en couches de la suite Internet », plutôt que de « modèle TCP/IP », à la fois parce que le terme de « modèle » porte à confusion

dans le cadre de la modélisation et simulation, et parce que le nom « TCP/IP » est trompeur, dans la mesure où le modèle n'est pas restreint à TCP.

Bien que l'analogie ne soit pas exacte, nous acceptons de considérer ici que les couches de la suite Internet correspondent aux quatre couches les plus basses du très réputé modèle OSI, combinées à une cinquième couche *application* au niveau le plus haut.

### 3.2.2 Modèle en cinq couches

Les cinq couches de la suite Internet sont :

**5 (application) :** C'est notamment à ce niveau que se trouvent tous les logiciels destinés aux utilisateurs finaux, et au traitement de données sur les serveurs. C'est le point d'accès aux services réseau, grâce à la notion de *sockets* qui est utilisée en programmation orientée réseau, et qui permet de transmettre des données à la couche inférieure.

**4 (transport) :** Gestion des connexions de bout en bout, avec un contrôle des flux. C'est notamment à ce niveau qu'intervient la notion de port, qui permet aux données qui arrivent via le réseau d'être associées à la bonne application (i.e. socket) à la couche supérieure.

**3 (réseau) :** Gestion du parcours des données sur le réseau et de l'adressage logique des cartes réseau (adresses IP).

**2 (liaison de données) :** Gestion des communications au sein d'un même réseau IP et de l'adressage physique des cartes réseau (adresses MAC). Bien que cette couche soit parfois divisée en deux sous-couches différentes, nous n'utiliserons pas ce niveau de détail dans ce manuscrit.

**1 (physique) :** Transformation des signaux sous forme numérique ou analogique (impulsions électriques ou lumineuses).

Les protocoles de la suite Internet sont catégorisés en fonction de la couche à laquelle ils interviennent.

### 3.2.3 Protocoles les plus utilisés

Les deux protocoles les plus utilisés en couche 4 sont :

**UDP :** Protocole non-connecté et non-fiable. Les données transmises via UDP peuvent être perdues, dupliquées et transmises dans le désordre, sans aucune possibilité de contrôle. Ce manque de fiabilité est acceptable pour certaines applications, comme les logiciels de diffusion de la voix (VoIP), et permet d'obtenir d'obtenir des transmissions rapides (protocole simple) – RFC 768

**TCP :** Protocole connecté et fiable. Les données transmises via TCP sont vérifiées par le destinataire, et elles sont réémises par l'expéditeur dès lors qu'elles sont perdues ou altérées. Cette garantie de recevoir exactement ce qui a été envoyé a pour conséquence que TCP est potentiellement plus lent que UDP, notamment à cause d'un système de messages d'acquittement qui peut encombrer le réseau – RFC 793

Le protocole le plus utilisé en couche 3 est IP (*Internet Protocol*), pour lequel deux versions cohabitent actuellement :

**IPv4 :** C'est la version historique du protocole IP. Sauf cas spécifiques ou anciens réseaux, quasiment tous les réseaux informatiques du monde supportent actuellement au minimum

IPv4. Cette version de IP propose un système d'adressage des cartes réseau qui est sur 32 bits, et qui permet de produire des adresses IP du type `203.0.113.42`. Il permet de faire passer les données d'un réseau à l'autre, notamment sur Internet. Il transmet les données d'une machine à l'autre, en s'aiguillant grâce à des tables de routage, qui indiquent la direction à prendre (i.e. la carte réseau par laquelle sortir et à qui il faut transmettre les données) en fonction de l'adresse du destinataire – RFC 791

**IPv6 :** C'est la version évoluée de IP, qui remplace peu à peu IPv4, qui est lui-même amené à totalement disparaître à terme. IPv6 fonctionne de façon similaire à IPv4, bien qu'il apporte également énormément d'améliorations. IPv4 et IPv6 ne sont pas compatibles mais peuvent cohabiter sur une même machine avec un double système d'adressage. Les adresses IP utilisées par IPv6 sont sur 128 bits et sont du type `2001:db8::42` – RFC 2460

Le protocole le plus utilisé en couche 2 est Ethernet, qui est dépendant des protocoles ARP et NDP dans le cadre des réseaux IP :

**Ethernet :** Protocole de réseau local utilisé sur les câbles à paires torsadées et sur fibre optique. Les adresses physiques sur 48 bits qu'il utilise sont appelées des adresses MAC et sont du type `1A:22:F3:AA:BB:4E`. Sa version sans fil la plus connue est le Wifi – ISO/IEC 8802-3

**ARP :** Protocole permettant de trouver une adresse MAC à partir d'une adresse IPv4 – RFC 826

**NDP :** Protocole permettant, entre autres, de trouver une adresse MAC à partir d'une adresse IPv6 – RFC 4861

D'un point de vue matériel, les couches permettent également de classer les équipements réseau selon le niveau où ils interviennent.

### 3.2.4 Équipements réseau courants

Nous considérons que toutes les machines informatiques sont des ordinateurs. Ce qui les différencie est le niveau le plus haut des couches de la suite Internet qu'elles supportent.

Ainsi, on peut distinguer quatre principaux équipements, pour quatre couches différentes (il n'existe pas d'équipement qui se limite à la couche 4) :

**Station (couche 5) :** Équipement final, de type client (e.g. utilisateurs humains) ou serveur (traitement automatisés).

**Routeur (couche 3) :** Équipement de transit, permettant d'aiguiller les données d'un réseau IP à l'autre.

**Commutateur (couche 2) :** Équipement de transit également (*switch* en anglais), mais permettant d'aiguiller les données au sein d'un même réseau IP.

**Concentrateur (couche 1) :** Similaire à une multi-prise (*hub* en anglais), puisqu'il ne s'agit pas réellement d'un ordinateur mais d'un simple réplicateur de signaux électriques ou lumineux.

Lorsqu'on décrit la topologie d'un réseau IP, les différents équipements sont couramment qualifiés de nœuds.

Le passage d'une donnée applicative d'un équipement à l'autre nécessite de faire passer cette donnée d'une couche à l'autre, grâce aux mécanismes de l'encapsulation et de la désencapsulation.

### 3.2.5 Encapsulation et désencapsulation

La structure de la suite Internet repose sur l'utilisation des cinq couches qui ont été décrites ci-avant. Lorsqu'une donnée applicative est envoyée d'une application à l'autre, d'une station à l'autre, on dit qu'elle « descend » les couches, puis les « remontent ». Quand la donnée passe d'une couche à l'autre, elle est modifiée pour ajouter ou retirer de l'information spécifique à la couche. On parle alors d'encapsulation ou de désencapsulation, et la donnée change de nom selon la couche au niveau duquel elle se trouve :

**Couche 5 :** Données applicatives
**Couche 4 :** Datagrammes (ou segments)
**Couche 3 :** Paquets
**Couche 2 :** Trames
**Couche 1 :** Bits

Un exemple de descentes et remontées des couches, donnant lieu à des encapsulations et des désencapsulations, est donné dans la Figure 3.1. Un exemple concret de transmission pour cet envoi pourrait être : un niveau de température (données applicatives en couche 5) est envoyé sur du TCP (couche 4), le tout sur un réseau IPv6 (couche 3), avec des liaisons Ethernet à 1 Gb/s (couche 2), sur des câbles en cuivre de catégorie 6a (couche 1).

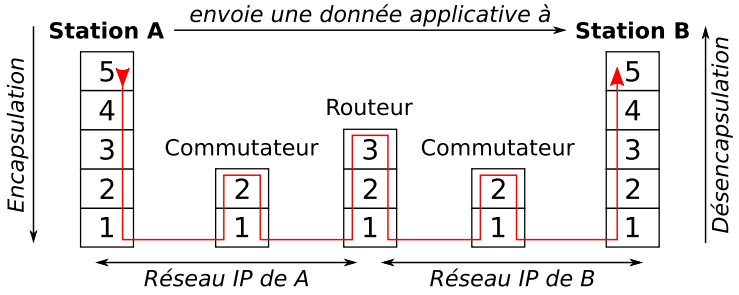



FIGURE 3.1 – Illustration du fonctionnement de la structure en couches de la suite Internet, avec un envoi d'une machine A vers une machine B, au travers de deux commutateurs et un routeur.

Sauf exception, les différentes couches de la suite Internet sont indépendantes. Dans le cadre de l'envoi en direct d'une donnée applicative, d'une machine A à une machine B, le détail des interactions entre les couches serait donc le suivant (on choisit d'ignorer les différentes mémoires tampons, et protocoles intermédiaires, dans cet exemple simplifié) :

— La couche 5 (le logiciel) de A génère la donnée applicative, et la transmet à sa couche 4.
— La couche 4 de A encapsule (ajoute des informations autour de) la donnée applicative et la transforme ainsi en datagramme. Elle ajoute notamment des numéros de port et éventuellement des informations liées à une session de bout en bout. Elle transmet ensuite ce datagramme à sa couche 3.
— La couche 3 de A encapsule le datagramme, pour en faire un paquet. Elle ajoute notamment des adresses IP. Elle transmet ensuite ce paquet à sa couche 2.
— La couche 2 de A encapsule le paquet, pour en faire une trame. Elle ajoute notamment des adresses physiques (e.g. Ethernet). Elle transmet ensuite cette trame à la couche 1.
— La couche 1 de A se charge de transformer cette trame en bits, puis en impulsions électriques ou lumineuses, qui sont envoyés à travers le média physique.

— La couche 1 de B réceptionne les impulsions, et les transforme en bits. Elle les transmet à sa couche 2.
— La couche 2 de B extrait une trame du flux de bits, et la désencapsule (utilise les informations spécifiques à la trame et les retire), pour en faire un paquet. Elle transmet ce paquet à sa couche 3.
— La couche 3 de B désencapsule le paquet, pour en faire un datagramme, qu'elle transmet à sa couche 4.
— La couche 4 de B désencapsule le datagramme, et obtient ainsi la données applicative seule. Elle la transmet enfin à sa couche 5.
— La couche 5 (le logiciel) de B exploite la donnée applicative, à sa discrétion.

### 3.2.6 Aller plus loin

Pour plus d'informations, [Kurose, James F, 2005] offre une présentation complète du fonctionnement des réseaux IP et des protocoles de la suite Internet.

Cet aperçu du monde des réseaux IP permet à présent de pouvoir étudier les différents travaux de la littérature qui concernent leur modélisation dans le cadre de co-simulations. Ces notions seront également largement réutilisées lors des Chapitres 5 et 6.

## 3.3 Distribution et parallélisation de simulateurs IP

Les travaux présentés dans cette section concernent des co-simulations composées uniquement de plusieurs instances d'un même simulateur IP. L'objectif est de paralléliser l'exécution des modèles IP supportés par ces simulateurs, correspondant à une distribution desdits simulateurs.

Nous distinguons deux sous-catégories dans les travaux retenus pour cette section de l'état de l'art :

1. Les simulateurs IP qui ont directement été conçus pour être distribués.
2. Les logiciels de surcouche qui permettent de distribuer des simulateurs IP existants qui n'ont pas été conçus pour ça initialement.

La première catégorie permet d'étudier les concepts mis en œuvre pour la distribution dans le cas le plus favorable, i.e. dans le cas où les auteurs sont libres d'implémenter les solutions qu'ils jugent les mieux adaptées, sans avoir à supporter un existant. La seconde catégorie permet de commencer à percevoir les problèmes qui apparaîssent dès lors qu'on souhaite intégrer un existant, qui ne répond pas obligatoirement aux exigences qu'on aurait souhaité.

D'autres travaux de la même nature nature sont listés dans l'Annexe A.2.

### 3.3.1 Simulateurs IP distribués par conception

#### GloMoSim

GloMoSim [Zeng et al., 1998] (vendu commercialement sous le nom de QualNet) est un simulateur de réseaux IP spécialisé dans les réseaux sans-fil.

Le simulateur a été écrit de façon modulaire, en séparant distinctement les couches de la suite Internet (cf. Section 3.2.2) en différentes fonctions. Ainsi, dès lors qu'une couche a été écrite en

respectant son API, elle peut remplacer une couche de niveau équivalent dans la simulation, sans n'avoir rien d'autre à changer au niveau du modèle. À titre d'exemple, TCP a été introduit dans GloMoSim en reprenant directement le code utilisé par le noyau de FreeBSD, avec une simple adaptation logicielle de façon à ce qu'il puisse communiquer de la façon décrite par l'API.

GloMoSim est conçu pour pouvoir distribuer l'exécution de ses modèles sur plusieurs processeurs (mais pas plusieurs machines) en utilisant la mémoire partagée. Il s'appuie pour ce faire sur le langage de simulation Parsec, qui propose plusieurs algorithmes de synchronisation exclusivement conservatifs (cf. Section 2.4.4). On retrouve ainsi le protocole basé sur des *messages nuls* (donc utilisant des lookaheads, cf. Section 5.4.3), un protocole basé sur des conditions d'événements, et un protocole qui mélange les deux précédents, qualifié de protocole basé sur des messages nuls « accéléré ».

Les calculs sont répartis entre les processeurs grâce au système de partitions de GloMoSim, qui réduit par ailleurs drastiquement les temps d'exécution. Sa stratégie de distribution est basée sur le regroupement de nœuds en partitions. Une partition est un rectangle qui représente une zone géographique et qui englobe tous les nœuds qui sont localisés dans cette zone (tous les nœuds devant être associés à des coordonnées géographiques dans les modèles).

À l'exécution, GloMoSim n'instancie qu'un objet par couche et par partition. Ainsi, l'empreinte mémoire d'une partition ne dépendra pas directement du nombre de nœuds qu'elle est chargée de modéliser. Au niveau charge des processeurs, l'exécution est également optimisée, par exemple dans le cas des messages de diffusion (largement utilisés dans le cas des protocoles sans-fils) : plutôt que de transmettre le message à tous les nœuds qui sont censés être accessibles par le nœud émetteur (en ayant la contrainte de devoir calculer cette liste), ou de transmettre le message à l'ensemble des nœuds du simulateur (en ayant la contrainte de devoir ensuite calculer si le message doit être reçu ou non pour chacun des nœuds), il est transmis uniquement aux partitions qui sont dans le champ de réception. Ensuite, les partitions ont la charge de ne transmettre le message qu'aux nœuds qu'elle contiennent et qui sont censés être capables de le recevoir.

#### GTNetS

GTNetS [Riley, 2003] est un simulateur de réseaux IP créé pour concevoir des simulations de réseaux IP potentiellement composés de millions de nœuds.

Après avoir constaté qu'un simulateur devait de préférence avoir été conçu dans cette optique pour atteindre un tel passage à l'échelle [Riley and Ammar, 2002], les auteurs de GTNetS ont proposé un simulateur qui s'appuie sur un certain nombre d'optimisations qui permettent d'obtenir d'excellentes performances et un passage à l'échelle important, tout en utilisant des machines disposant de capacités limitées. Contrairement à GloMoSim, GTNetS peut être distribué sur des machines distantes, sans utiliser de mémoire partagée.

Il représente les couches de la suite Internet (cf. Section 3.2.2) de façon distinctes, en les séparant logiciellement par des fonctions. Les paquets IP simulés contiennent les informations liées à chacune des couches correspondant aux protocoles qu'ils transportent.

La principale originalité de GTNetS est d'utiliser un ordonnanceur d'événements en $\mathcal{O}(\log N)$ qui permet de limiter la taille de la pile des événements, et donc le coût de traitement de ces événements et l'empreinte mémoire de la pile. Le temps de calcul pour prendre les décisions de routage des paquets est également fortement diminué grâce à l'utilisation de NIX-Vectors [Riley

et al., 2001b], qui permettent d'ajouter des informations de routage dans les packets, et donc de s'affranchir de la consultation de tables de routage potentiellement très volumineuses. D'autres optimisations originales, comme l'absence totale de calcul du routage pour les nœuds qui ne sont reliés qu'à un seul routeur (feuilles), permettent d'optimiser encore davantage l'exécution de ses simulations. Enfin, GTNetS est également fortement optimisé pour limiter autant que faire se peut l'empreinte mémoire et les écritures disques lors de l'export des journaux de simulation.

La distribution des modèles se fait de façon explicite, en créant des liens distants [Riley, 2003]. Le modélisateur a donc à charge de découper la topologie du réseau IP qu'il souhaite simuler, en différents modèles, qui en représentent différentes parties. En indiquant simplement l'adresse IP et le masque de réseau du nœud distant qui est censé être connecté à un modèle (et qui existe dans un autre modèle exécuté par une autre instance de GTNetS), GTNetS transporte les paquets simulés d'un modèle à l'autre.

Une instance du simulateur ne connaît pas obligatoirement la topologie de l'ensemble du réseau simulé, et n'a pas connaissance des nœuds et des liens qui sont censés suivre un lien distant défini dans son modèle. Il résulte de cette contrainte un certain nombre de problèmes lorsqu'il est nécessaire de calculer des informations liées au routage des paquets. Les deux solutions proposées sont (1) de définir un ensemble de routes statiques qui permettent au simulateur de connaître la liste des préfixes IP qui peuvent être routés via le lien distant ou (2) de créer un ensemble de liens fantômes. Ces derniers représentent dans le modèle local l'ensemble des nœuds qui sont définis dans les modèles distants, sans pour autant être utilisés dans la simulation pour autre chose que le calcul des routes. Cette seconde solution, qui permet à chaque simulateur d'avoir conscience de l'intégralité de la topologie du réseau simulé, a l'inconvénient d'augmenter l'empreinte mémoire du modèle, dans le cas où la topologie est composée de millions de nœuds.

#### SSFNet

SSFNet est un simulateur de réseaux IP qui exécute des modèles SSF [Cowie et al., 1999]. L'objectif de SSFNet est de permettre de modéliser le réseau Internet, et donc de très grands réseaux de façon générale. Les modèles SSF ont pour objectif de rester très simplistes, pour permettre d'obtenir des temps d'exécution réduits, et pour permettre aux modélisateurs de facilement s'approprier l'API proposée pour en concevoir de nouveaux. Un DML (*Domain Modeling Language*) [Yoon and Kim, 2009] est également mis à disposition des modélisateurs pour décrire les réseaux simulés très facilement.

Les différentes instances de SSFNet sont synchronisées grâce à un algorithme de type conservatif (Quanta Synchronization), réputé pour passer à l'échelle [Nicol, 1998].

Pour distribuer la simulation, le modélisateur a pour charge de regrouper les nœuds de son réseau, afin d'indiquer ceux qui sont susceptibles de communiquer souvent et très rapidement ensemble. Ce partitionnement permet au simulateur de répartir les calculs sur les différents processeurs de la machine, au moment de l'exécution. À l'instar de GloMoSim, l'utilisation de mémoire partagée ne permet pas à SSFNet de distribuer l'exécution de ses modèles sur plusieurs machines.

Les différentes couches de la suite Internet (cf. Section 3.2.2) sont représentées séparément dans des objets distincts (appelées *packages*), qui peuvent éventuellement être partagés en mémoire entre différents paquets. Les événements peuvent contenir des trains de paquets, qui ne sont donc pas représentés individuellement au niveau de la simulation.

### 3.3.2 Surcouches pour la distribution de simulateurs IP séquentiels existants

#### PDNS

PDNS [George Riley, 2014] est une surcouche au simulateur de réseaux IP NS-2, permettant de distribuer ses simulations. NS-2 est un simulateur de réseaux IP généraliste, particulièrement bien adopté dans le domaine scientifique et disposant d'une bibliothèque de modèles conséquente.

Les modifications apportées à NS-2 pour le distribuer sont mineures et consistent principalement à définir un nouvel ordonnanceur, ainsi qu'un nouveau type de lien, qualifié de « distant ». Les instances de NS-2 sont regroupée sous la forme d'une fédération, similaire à ce que propose le standard HLA (cf. Section 2.5.3). La synchronisation entre les instances du simulateur est assurée grâce à un algorithme de type conservatif (cf. Section 2.4.4).

De la même façon que pour GTNetS, pour distribuer une simulation, le modélisateur doit écrire plusieurs modèles, chacun représentant une partie de la topologie IP. Les connexions entre les modèles sont définies grâce à des liens distants, qui identifient un nœud distant par son adresse IP. Ce nœud distant est défini localement dans un autre modèle, lui-même doté d'un lien distant défini avec l'autre IP du nœud de l'autre extrémité. Le routage est assuré grâce aux routes statiques, que le modélisateur doit obligatoirement définir dans chacun des modèles, pour indiquer les réseaux accessibles de l'autre côté des liens distants. Ainsi, aucun modèle n'a connaissance de l'ensemble de la topologie, et les problématiques liées au routage sont similaires à celles rencontrées par GTNetS.

NS-2 représente séparément les différentes couches des paquets IP, et ceux-ci sont représentés de façon atomique dans le simulateur, ce qui permet de les échanger facilement d'une instance à l'autre.

#### NS-3/MPI

La distribution de NS-3 [Henderson et al., 2006] n'a pas été pensée dès sa conception, mais le travail de [Pelkey and Riley, 2011] a été totalement intégré depuis au projet.

NS-3 a la particularité d'être l'héritier de NS-2, qui a longtemps été une référence incontournable dans le domaine de la simulation de réseaux IP. Bien qu'il ne soit pas compatible avec la volumineuse bibliothèque de modèles de son prédécesseur, NS-3 dispose d'ors et déjà de nombreux modèles et fait l'objet de nombreux travaux de recherche.

La parallélisation de NS-3 est réalisée grâce à la bibliothèque MPI, qui permet de faire passer des messages facilement entre processeurs ou ordinateurs. Pour distribuer une simulation, l'écriture d'un seul modèle, représentant l'intégralité de la topologie IP, est suffisante. Il est par contre nécessaire de préciser explicitement les endroits de la topologie IP qui pourront faire l'objet d'un découpage automatique du simulateur, pour simuler une partie sur une machine et l'autre partie sur une autre machine. La seule contrainte est que cet endroit corresponde à un lien pair-à-pair. Il faut donc obligatoirement que des liens pair-à-pair soient utilisés dans le réseau IP à simuler.

Par conséquent, toutes les instances de NS-3 ont connaissance de l'intégralité de la topologie IP, ce qui permet de s'affranchir des problèmes liés au calcul des routes. Par contre, l'empreinte mémoire de chacune des instances peut être très importante si le réseau simulé contient des milliers de nœuds.

Il suffit ensuite d'exécuter la simulation en utilisant une implémentation de MPI, pour que ce dernier répartisse tout seul les processus sur les ressources qu'on lui a mis à disposition. La synchronisation étant assurée par un algorithme conservatif (cf. Section 2.4.4), le lookahead (cf. Section 5.4.3) est déterminé automatiquement en fonction du délai configuré pour le sous-modèle de lien pair-à-pair.

Afin de faciliter les interconnexions entre NS-3 et des réseaux non-simulés, les paquets IP sont représentés par NS-3 de la même façon qu'ils sont représentés dans un système GNU/Linux. Les différentes couches de la suite Internet d'un paquet sont traités par une chaîne de fonctions, chacune représentant l'une de ces couches (cf. Section 3.2.2).

#### OMNeT++/INET

OMNeT++ [Varga and Hornig, 2008] est un simulateur à événements discrets qui n'est pas spécialisé dans les réseaux IP. Associé avec la bibliothèque de modèles INET, il a toutefois toutes les qualités d'un simulateur de réseaux IP et est particulièrement utilisé dans cette combinaison.

Plusieurs travaux [D. Wu et al., 2002, Sekercioglu et al., 2003] ont permis d'aboutir à une version distribuable de OMNeT++. Comme pour NS-3, la communication entre les processus est assurée par MPI, et les algorithmes de synchronisation sont de type conservatif (cf. Section 2.4.4). Afin de limiter le surcoût de ces communications et de limiter le volume des données échangées, l'algorithme complémentaire SSM est proposé. Plutôt que de transmettre des paquets IP d'une instance de OMNeT++ à l'autre, SSM envoie des statistiques, qui permettent de recréer les paquets sur l'instance distante. Toutefois, les auteurs précisent que l'utilisation de SSM introduit des erreurs dans les résultats de simulation, qui sont alors moins précis qu'en utilisant l'algorithme conservatif avec échange de paquets.

Comme pour la version MPI de NS-3, cette version distribuée de OMNeT++ impose à chaque instance du simulateur de connaître l'intégralité de la topologie IP. La répartition de la topologie sur les différentes instances est à la charge du modélisateur, qui doit regrouper lui-même les différents nœuds en partitions, dans un fichier de configuration. Les partitions seront alors potentiellement simulées sur des machines ou des processeurs différents, lors de l'exécution.

Cependant, d'après [Stoffers et al., 2014], les modèles IP disponibles dans la bibliothèque INET ne sont pas conçus pour être exécutés en parallèle. Les auteurs proposent des recommandations pour optimiser ces modèles, et une solution nommée DMSI qui permet de résoudre la plupart des problèmes liés à l'initialisation. Ainsi par exemple, ils permettent aux autoconfigurateurs de INET de correctement assigner des adresses MAC et IP aux différents nœuds répartis sur plusieurs instances, en ayant l'assurance qu'il n'y aura pas de collision. L'inclusion de DMSI dans OMNeT++ nécessite des modifications importantes dans ce dernier.

Les différentes couches de la suite Internet (cf. Section 3.2.2) sont représentées de façon distinctes dans les modèles INET, et les paquets IP sont représentés individuellement dans un format spécifique à OMNeT++/INET.

## 3.4 Interconnexion de simulateurs IP hétégorènes

Les travaux présentés dans cette section concernent les co-simulations qui ont pour objectif d'intégrer des simulateurs IP séquentiels existants, et qui ne sont généralement pas prévus pour

communiquer avec d'autres simulateurs. Les nouvelles problématiques soulevées relèvent par conséquent principalement de différentes formes d'hétérogénéité rencontrées, à intégrer ensemble (e.g. au niveau de la représentation du temps et des paquets IP, de la dynamique, etc).

D'autres travaux de la même nature nature sont listés dans l'Annexe A.3.

### 3.4.1 Dynamic Simulation Backplane

Le Dynamic Simulation Backplane [Riley et al., 1999, Riley et al., 2001a] (DSB) est une plateforme de co-simulation qui a pour vocation de permettre à n'importe quel simulateur de réseau IP d'échanger des paquets IP simulés avec d'autres simulateurs de réseau IP.

La plateforme utilise le principe de la fédération avec un RTI, en se reposant sur les spécifications propres à HLA (cf. Section 2.5.3). Chacun des simulateurs participant à la co-simulation est considéré comme un fédéré, et doit disposer à ce titre d'un connecteur logiciel approprié pour rejoindre la fédération. Les simulateurs sont synchronisés grâce à un algorithme de type conservatif (cf. Section 2.4.4).

Durant la simulation, le DSB a pour charge de résoudre les problèmes de représentation des paquets IP simulés, qui peuvent différer d'un simulateur à l'autre. Ainsi, lorsqu'un simulateur souhaite transmettre un paquet IP à un autre simulateur, il le confie au DSB qui se charge de le convertir dans un format universel grâce au connecteur. Le paquet IP sera de nouveau converti dans le format spécifique du simulateur distant, lorsque le DSB lui transmettra.

Le DSB a également pour vocation de gérer l'absence de certaines fonctionnalités d'un simulateur à l'autre. Ainsi, si un simulateur transmet un paquet IP avec du HTTP (protocole applicatif pour le web) dans la partie applicative mais que le simulateur distant ne dispose pas d'implémentation de ce protocole, la partie concernant HTTP sera considérée soit comme *rebus* soit comme *baggage*. Dans le premier cas, le simulateur qui reçoit le paquet aura le droit de simplement ignorer la partie HTTP quand il transcrira le paquet fourni par le DSB dans son propre format de représentation. Dans le second cas, le simulateur devra considérer qu'il s'agit de données binaires, et qu'elle doivent être transportées dans le paquet, sans pour autant avoir à en comprendre la signification.

Les données inconnues sont considérées comme pouvant être mises au rebus ou comme devant nécessairement être transportées, par le simulateur qui a transmis le paquet comportant ces données. Le DSB a connaissance de cette information grâce à une phase d'initialisation de la co-simulation. Avant le démarrage de celle-ci, chaque simulateur a pour charge de communiquer au DSB la liste de tous les protocoles et les données associées qu'il supporte (cf. exemple illustré par la Figure 3.2). Pour chaque protocole et chaque donnée, il doit préciser s'ils sont optionnels ou indispensables. Cette première étape permet également au DSB de détecter les incompatibilités fortes : par exemple, si deux simulateurs sont censés communiquer entre eux, mais que l'un indique que IPv6 est obligatoire alors que l'autre ne fourni pas ce protocole dans sa liste, le DSB refuse de démarrer la co-simulation.

Les premiers tests ont été réalisés en utilisant une version modifiée de PDNS, qui permet de distribuer le simulateur NS-2. Cette version modifiée utilise le DSB pour faire communiquer et synchroniser les instances de NS-2. En comparant les temps d'exécution et les résultats pour l'exécution d'un même modèle avec et sans utilisation du DSB dans PDNS, les auteurs ont déterminé que l'utilisation du DSB ne modifiait pas les résultats de simulation et que son impact sur les temps d'exécution était négligeable. Une autre preuve de concept illustre l'interconnexion

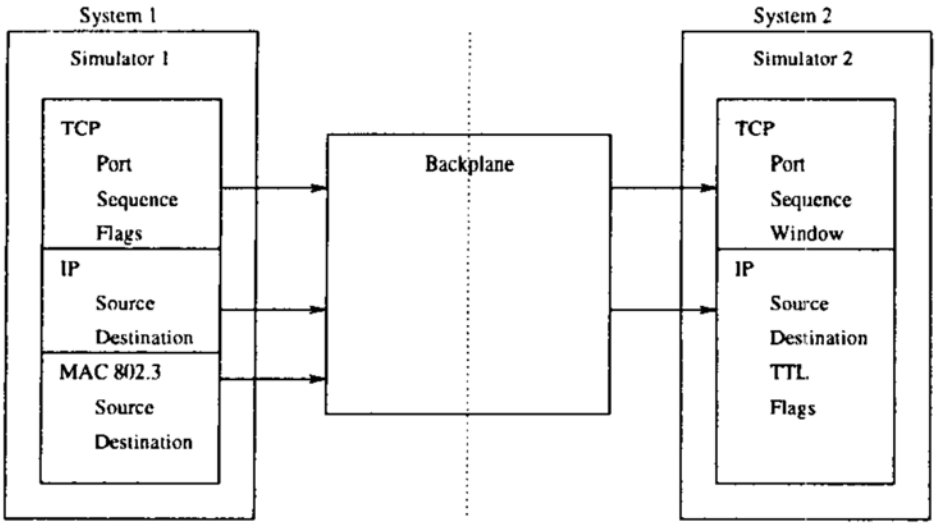



FIGURE 3.2 – Exemple de protocoles et de données associées pour deux simulateurs différents (source : [Riley et al., 2001a]).

réussie de NS-2 avec GloMoSim.

Un second travail [Xu et al., 2001] propose une utilisation légèrement différente du DSB. Plutôt que de considérer que les paquets sont transmis entre les simulateurs à leur niveau le plus bas (c'est à dire que l'intégralité du paquet est transmis, de la couche physique à la couche applicative), les échanges peuvent avoir lieu entre deux couches de suite Internet qui sont adjacentes (cf. Section 3.2.2). Ainsi, par exemple, NS-2 peut être utilisé pour simuler les couches transport et application, tandis que GloMoSim peut être utilisé pour simuler les couches inférieures. Cette nouvelle possibilité permet d'utiliser le simulateur le plus approprié pour chaque couche d'un même paquet.

Le DSB fournit déjà tous les outils nécessaires pour connecter les simulateurs de cette façon. Le problème spécifique qui se pose dans cette situation concerne le lookahead (cf. Section 5.4.3), utilisé dans le cadre de l'algorithme de synchronisation conservatif utilisé par le DSB. Dans le cas où un simulateur se contente de générer les entêtes liés à une couche spécifique pour les fournir à un autre simulateur, le lookahead devient potentiellement nul, parce que le temps nécessaire pour générer ces données dans le système est lui-même considéré comme nul. Ainsi, la contribution de ce travail complémentaire est principalement de considérer qu'un simulateur (maître) peut être directement dépendant d'un autre simulateur (esclave) pour une tâche donnée : le maître est celui qui est connu du DSB comme faisant partie de la co-simulation et l'esclave se contente de communiquer avec son maître, de façon purement séquentielle.

Les travaux gravitant autour du DSB ne précisent pas comment les simulateurs ont été intégrés à la fédération. Les auteurs précisent qu'ils souhaitent à terme intégrer le simulateur commercial OpNet [Chang, 1999], mais que l'absence de code-source disponible complique son intégration.

### 3.4.2 Maya

Maya est une plateforme de co-simulation qui a pour objectif d'intégrer des simulateurs de réseaux IP qui utilisent des formalismes hétérogènes, tout en prenant en considération des contraintes liées au temps réel.

Ainsi, comme illustré dans la Figure 3.3, Maya est capable de simuler un réseau IP complet, en représentant un tronçon de sa topologie avec Qualnet (événements discrets) et un autre tronçon avec un modèle basé sur des flux de fluides (équationnel). Cette simulation est également reliée à des interfaces réseau réelles, avec lesquelles les simulateurs doivent échanger des paquets IP.

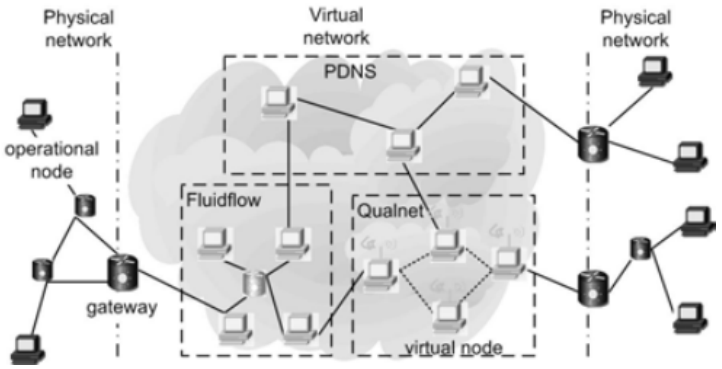



FIGURE 3.3 – Exemple de multi-modèle hybride exécuté par Maya (source : [Zhou et al., 2003]).

La synchronisation utilise un algorithme de type conservatif (cf. Section 2.4.4), avec une fédération et un RTI, à l'instar de HLA (cf. Section 2.5.3). Les paquets IP échangés entre les différents composants sont convertis d'un format à l'autre de façon similaire au DSB de Riley, qui est cité en référence. L'architecture de Maya (cf. Figure 3.4) intègre également un ordonnanceur interne et un collecteur de statistiques, principalement utilisés pour l'intégration du modèle équationnel.

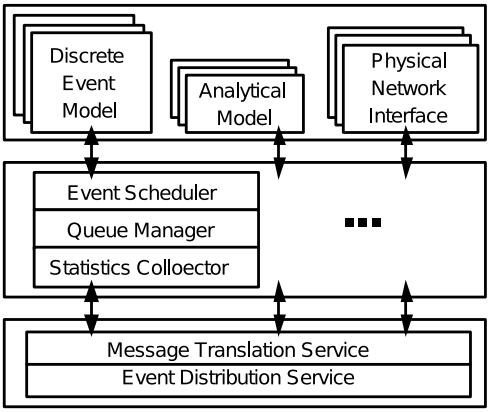



FIGURE 3.4 – Architecture de la plateforme Maya (source : [Zhou et al., 2003]).

Lorsqu'un paquet IP est à destination du tronçon de réseau représenté par le modèle équationnel, les informations liées à ce paquets sont fournies au collecteur de statistiques. Des événements

demandant l'exécution du solveur permettant de résoudre le modèle équationnel sont régulièrement ajoutés à l'ordonnanceur interne. Lorsque ceux-ci se déclenchent, les statistiques de flux de trafic calculées par le collecteur sont fournies en données d'entrée au modèle équationnel, juste avant son exécution. Une fois l'exécution terminée, un temps de transmission des paquets est récupéré dans les ports de sorties du modèle, et stocké dans le collecteur de statistiques. Ce temps est utilisé pour simuler la transmission des paquets IP qui sont censés traverser la partie du réseau représentée par le modèle équationnel.

Les contraintes pour intégrer Qualnet à la fédération de Maya ne sont pas évoquées dans les travaux.

### 3.4.3 NS-3 / Manifold

Les travaux de [Stoffers and Riley, 2012] s'intéressent à l'interconnexion entre NS-3 et le simulateur d'architecture informatique Manifold. Les deux simulateurs sont intégrés à une même co-simulation, de façon à pouvoir utiliser les modèles de NS-3 pour simuler les couches logicielles des paquets IP, et Manifold pour pouvoir simuler des algorithmes complexes de gestion des files d'attente.

Le couplage entre les deux simulateurs ne repose pas sur une plateforme de co-simulation existante, et les choix sont spécifiques aux deux simulateurs choisis. Ainsi, le choix de MPI pour faire communiquer les simulateurs entre eux vient principalement du fait que les deux simulateurs intègrent déjà des fonctionnalités de ce protocole. Comme illustré dans la Figure 3.5, des wrappeurs sont ajoutés pour chacun des simulateurs, avec des interfaces permettant à chacun d'entre eux de pouvoir connaître le temps du prochain événement de l'autre simulateur, et d'en récupérer le paquet éventuellement associé. Les fonctionnalités natives de MPI pour la synchronisation du temps n'ont pas pu être utilisées, parce que les structures utilisées par les deux simulateurs pour représenter le temps sont différentes. Le protocole de synchronisation a donc été réimplémenté, sans être documenté dans la publication.

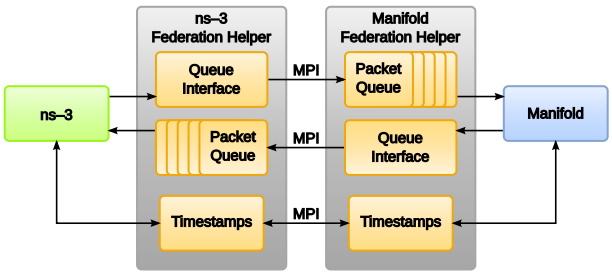



FIGURE 3.5 – Composants principaux utilisés pour relier NS-3 à Manifold (source : [Stoffers and Riley, 2012]).

Le modèle réseau de Manifold a été entièrement implémenté pour l'exemple. Du côté NS-3 comme Manifold, des connecteurs ont été ajoutés aux simulateurs et à leurs modèles (cf. Figure 3.6). Ainsi, dans NS-3, la classe représentant la couche de liaison de données a été surchargée, pour être en capacité de récupérer les paquets générés et pour pouvoir en ajouter de nouveaux dans la file d'attente du nœud associé. Puisque Manifold n'utilise pas le contenu des paquets dans son modèle, les auteurs ont choisi de ne transmettre que les informations utiles des paquets à destination de Manifold. NS-3 doit alors garder une copie du paquet initialement

envoyé, pour pouvoir le retransmettre après son passage dans Manifold, ou le supprimer si ce dernier lui indique qu'il a finalement été perdu.

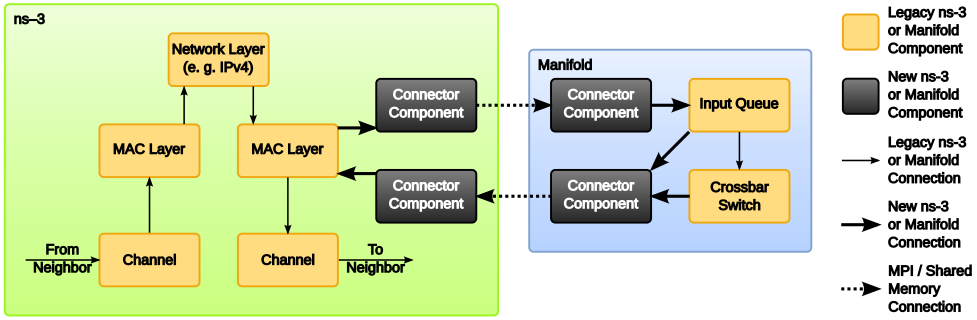



FIGURE 3.6 – Intégration de connecteurs à NS-3 et Manifold (source : [Stoffers and Riley, 2012]).

Comme dans le cas du DSB, un problème de lookahead nul intervient, puisque la génération des paquets dans NS-3 prend un temps suffisamment négligeable pour ne pas pouvoir être représenté sur la ligne de temps simulée. La simulation s'en trouve très fortement ralentie. D'après les auteurs, la seule solution serait alors de simuler une partie du réseau physique dans NS-3, pour obtenir un lookahead non-négligeable, ou de faire en sorte d'utiliser un couplage à mémoire partagée lorsque NS-3 n'est couplé avec Manifold que pour ses fonctionnalités de génération de paquets (solution similaire à celle proposée dans le cas du DSB).

## 3.5 Interconnexion de simulateurs IP avec des simulateurs d'autres domaines

Les travaux présentés dans cette section concernent les co-simulations qui interconnectent des simulateurs de réseaux IP avec des simulateurs d'autres domaines. Ainsi, les nouvelles problématiques soulevées sont généralement liées à l'intégration de formalismes hétérogènes (e.g. événements discrets vs. temps continu).

D'autres travaux de la même nature nature sont listés dans l'Annexe A.4.

### 3.5.1 EPOCHS

EPOCHS [Hopkinson et al., 2006] est une plateforme de co-simulation destinée à intégrer à la fois des simulateurs de réseau IP et des simulateurs de systèmes électriques. Ses auteurs la définissent comme la première du genre.

Les travaux sur la plateforme ont porté sur l'intégration de NS-2 comme simulateur IP, de PSCAD/EMTDC pour les modèles de transition électromagnétique et de PLSF pour ceux de transition électromécanique. Les auteurs précisent toutefois qu'ils n'ont jamais fait d'expérience qui réunisse les trois simulateurs à la fois, parce que les différences en terme d'échelles de temps sont trop importantes entre le domaine électromagnétique et le domaine électromécanique, entraînant des simulations beaucoup trop longues à exécuter.

EPOCHS utilise un RTI avec une fédération, proche de l'esprit de HLA (cf. Section 2.5.3). Toutefois, puisque les simulateurs ne proposent pas de connecteurs HLA, ils ont été intégrés en

fonction du niveau de personnalisation qu'ils proposent. Ainsi, de nouveaux modèles d'équipements électriques ont été ajoutés à PSCAD/EMTDC et PLSF, pour permettre à la plateforme de lancer des actions en récupérant des informations sur l'état des modèles, chaque fois qu'il est exécuté. Du côté de NS-2, un nouveau protocole de communication a été ajouté, pour récupérer et injecter des paquets IP, ainsi que pour contrôler l'exécution de la simulation.

PSCAD/EMTDC et PLSF utilisent des modèles à changements d'état continus, et doivent donc être exécutés à l'aide de pas de résolution et de communication (cf. Section 2.3.2). À l'inverse, NS-2 utilise des modèles à changements d'état discrets et doit donc être exécuté avec des événements discrets. Afin de synchroniser le temps entre ces différents simulateurs, EPOCHS consulte l'état des modèles de NS-2 à l'aide d'un pas de communication, comme s'il s'agissait également de modèles à changement d'état continus (d'autres travaux, comme [Fuller et al., 2013], ont fait ce choix).

Pour y parvenir, des événements sont programmés de façon artificielle dans NS-2, pour chaque pas de communication, afin que le simulateur puisse se mettre en pause. En utilisant cette facilité, EPOCHS perd le principal intérêt des simulations à base d'événements discrets, à savoir la détection des événements à leur temps exact. Ainsi, des événements générés par NS-2 au milieu d'un pas de communication ne seront pris en compte par les autres simulateurs qu'au temps correspondant à la fin dudit pas de communication. Comme pour les modèles à changements d'état continus, utiliser un pas de communication nécessite de faire choisir une valeur au modélisateur, qui ne soit ni trop petite pour ne pas nuire aux performances, ni trop grande pour ne pas trop dénaturer les résultats de simulation.

Les auteurs précisent également que les simulateurs dont le code-source n'est pas disponible, est un obstacle pour leur intégration à leur co-simulations.

### 3.5.2 ORNL Power System Simulator

Le simulateur de réseaux électriques de l'ONRL [Nutaro, 2011] est une plateforme de co-simulation qui poursuit des objectifs similaires à ceux de EPOCHS, en proposant d'intégrer des simulateurs de réseaux électriques avec des simulateurs de réseaux IP.

Le simulateur de l'ONRL est basé sur adevs [Nutaro, 2010], une bibliothèque qui permet d'exécuter des modèles utilisant le formalisme DEVS (cf. 2.5.2). Les modèles électriques sont ceux proposés par la bibliothèque THYME, qui a été conçue pour fonctionner avec adevs. Des modèles de réseaux IP sont intégrés au simulateur, mais le projet propose également un connecteur permettant de communiquer avec un simulateur de réseaux IP externe (cf. 3.7). Cette fonctionnalité a été testée avec NS-2 et OMNeT++/INET.

La principale problématique résolue par le simulateur de l'ONRL concerne le problème de précision dans les résultats de simulation rencontré par EPOCHS, lorsque des événements discrets interviennent entre deux pas de communication. Ainsi, grâce à un système de rollbacks (cf. Section 2.3.2) et des fonctions de détection, le simulateur est capable de détecter le temps exact des événements d'état dans les modèles de THYME. Il est également capable d'intégrer des événements programmés à l'avance, ou provenant du simulateur IP, aux temps exacts auxquels ils sont censés intervenir. La solution utilisée est similaire à celle décrite dans [Camus et al., 2016].

La méthode d'intégration de NS-2 et OMNeT++ n'est pas précisée, mais certaines interactions entre le simulateur IP et adevs sont décrites dans [Nutaro et al., 2007]. Les auteurs estiment

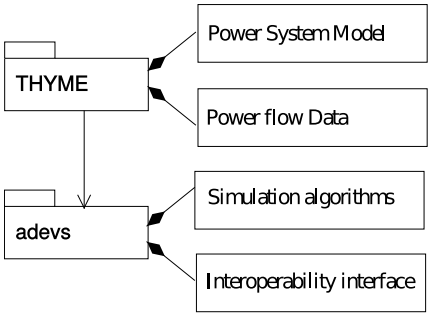



FIGURE 3.7 – Architecture logicielle du simulateur de l'ONRL, avec un connecteur (*Interoperability Interface*) permettant de communiquer avec un simulateur de réseaux IP externe (source : [Nutaro, 2011]).

également que les logiciels libres ont un grand rôle à jouer dans le domaine de la recherche, parce qu'ils permettent de répondre à de nouveaux besoins qui ne sont pas satisfaits par les solutions commerciales.

### 3.5.3 FNCS

FNCS [Ciraci et al., 2014] est une plateforme de co-simulation, destinée à intégrer des simulateurs de réseaux IP, des simulateurs de transmission électrique et des simulateurs de distribution électrique.

Sans pour autant implémenter le standard HLA (cf. Section 2.5.3), FNCS est basé sur le principe de la fédération. Les détails concernant les simulateurs intégrés et la façon de les intégrer ne sont pas indiqués. Le projet FNCS est principalement centré sur les problématiques liées à la synchronisation du temps, et les performances de l'exécution de la co-simulation qui en découlent.

Alors que la plupart des travaux dans le domaine de la co-simulation intégrant des simulateurs IP se contentent d'utiliser un algorithme de synchronisation conservatif (sauf quelques exceptions notables comme [Szymanski et al., 2002]), FNCS propose une alternative originale, basée sur deux algorithmes utilisés conjointement.

Le premier algorithme est conservatif (cf. Section 2.4.4). Chaque simulateur électrique calcule le temps de son prochain événement et le minimum des temps calculés est utilisé comme prochain temps maximal autorisé jusqu'auquel les simulateurs peuvent exécuter leurs modèles. Le prochain événement est caractérisé en fonction des messages que les simulateurs ont à recevoir de la part du réseau IP. Grâce à un système de compteurs, FNCS permet en effet à chaque simulateur électrique de savoir s'il y a actuellement un message pour lui, émis par un autre simulateur électrique, en transit dans le modèle IP. Si c'est le cas, le simulateur estime le temps de son prochain événement au temps courant plus l'unité de temps minimale qu'il peut lui ajouter. Tant qu'il a ce message en transit qui n'est pas arrivé, il prédit le temps de son prochain événement de cette façon, en avançant le plus lentement possible (un système de temporisateurs permet de débloquer la situation dans laquelle le message serait considéré comme perdu par le modèle IP). S'il n'y a pas de message en transit pour lui, il calcule le temps de son prochain événement en ajoutant simplement son pas de communication au temps courant.

Grâce au système de compteurs de messages en transit, le simulateur de réseaux IP n'a

pas besoin de participer au calcul du temps maximal autorisé. Puisque les modèles IP utilisent généralement des échelles de temps très petites et utilisent de nombreux événements internes très rapprochés (en particulier dans le cas des protocoles sans-fils), cette première optimisation a pour effet de grandement accélérer l'exécution de la co-simulation. Toutefois, si les simulateurs électriques ont eux-mêmes des échelles de temps très différentes (cf. le problème de EPOCHS avec les échelles de temps des simulateurs électromagnétiques et électromécaniques), la co-simulation continue d'être ralentie, avec des simulateurs qui passent beaucoup de temps à attendre.

Pour corriger ce problème, FNCS propose d'exécuter une seconde stratégie de synchronisation en parallèle, qu'on pourrait qualifier d'optimiste (cf. Section 2.4.4) : la stratégie de basculement spéculatif. Avant de laisser un simulateur électrique s'exécuter en mode conservatif, FNCS prend l'initiative de le *forker* (dupliquer le processus). Le processus fils démarre alors une exécution optimiste, en faisant fortement avancer son modèle dans le temps, en fonction d'un temps d'exploration défini à l'avance par le modélisateur. Pendant ce temps, le processus père continue d'exécuter le modèle en mode conservatif. Si le processus fils constate que son modèle est censé échanger un message avec un autre simulateur électrique avant la fin du temps d'exploration, il prend l'initiative de se tuer lui-même et laisse au processus père la charge continuer la simulation (tout en prenant soin de lui communiquer le temps auquel il a échoué, permettant également à l'algorithme conservatif d'avancer plus rapidement). Si par contre le processus père obtient l'autorisation d'avancer (au minimum) jusqu'au temps correspondant au temps d'exploration, le processus père est tué et le fils prend officiellement la main.

Contrairement aux simulations optimistes avec retours en arrière (qui sont rarement possibles pour les simulateurs IP, cf. Section 2.3.2), cette méthode permet d'obtenir de meilleures performances dans le cas où l'exploration échoue, puisqu'il suffit de laisser la main au processus père, sans n'avoir rien d'autre à faire. Le choix du temps d'exploration autorisé pour chaque simulateur électrique influence beaucoup les performances obtenues à l'exécution, selon les modèles utilisés.

La solution proposée ne semble pas donner directement de solution en ce qui concerne la détection du temps exact auquel un événement d'état (e.g. changement de valeur censée déclencher l'envoi d'un message) est détecté dans un modèle électrique.

Les auteurs estiment que leur solution améliore le temps d'exécution des co-simulations de ce type d'environ 20%.

## 3.6 Conclusion

Les différents travaux présentés dans cette section illustrent plusieurs méthodes différentes pour intégrer des modèles IP à une co-simulation. Quel que soit l'objectif de cette intégration, ils soulèvent des problématiques communes, avec parfois des propositions de solution :

1. Comment synchroniser le temps des simulateurs de façon optimale (e.g. GloMoSim, FNCS) ?
2. Comment réduire le temps de simulation de la co-simulation pour simuler de grands réseaux (e.g. GTNetS, SSFNet) ?
3. Comment réduire le surcoût des communications entre les instances de simulateurs (e.g. OMNeT++/INET, NS-3/Manifold) ?
4. Comment éviter l'utilisation de lookaheads nuls (e.g. DSB, NS-3/Manifold) ?

5. Comment intégrer des simulateurs non-libres (e.g. EPOCHS, ONRL, FNCS, DSB) ?

L'interconnexion de simulateurs IP entre eux ajoutent des problématiques spécifiques, les trois premiers concernant ce qu'on appellera ensuite la co-initialisation :

1. Comment découper une topologie IP à simuler en plusieurs tronçons et les répartir sur plusieurs instances de simulateurs (e.g. GloMoSim, GTNetS, SSFNet, DSB) ?
2. Comment calculer les tables de routage lorsque les instances n'ont pas connaissance de l'intégralité de la topologie IP (e.g. GTNetS, PDNS) ?
3. Comment autoconfigurer les adresses physiques et logiques des nœuds, sans provoquer de collision, lorsque les instances n'ont pas connaissance de l'intégralité de la topologie IP (e.g. OMNeT++/INET) ?
4. Comment échanger des paquets IP simulés lorsque les simulateurs utilisent des représentations différentes (e.g. DSB, Maya) ?

Dès lors qu'on souhaite interconnecter ces simulateurs IP avec des simulateurs d'autres domaines (ou que les simulateurs IP interconnectés utilisent des formalismes différents, comme dans le cas de Maya), de nouvelles problématiques s'ajoutent à nouveau :

1. Comment faire communiquer des simulateurs à événements discrets avec des simulateurs équationnels (e.g. EPOCHS, ONRL, FNCS) ?
2. Comment intégrer les événements discrets au temps exact auquel ils correspondent, et comment détecter le temps exact d'événements discrets détectés sur des modèles équationnels (e.g. EPOCHS, ONRL) ?
3. Comment conserver des temps d'exécution efficaces quand les simulateurs utilisent des échelles de temps très différentes (e.g. EPOCHS, FNCS) ?

À quelques exceptions près, la plupart des solutions présentées utilisent des algorithmes de synchronisation de temps de type conservatif. Ceci est principalement lié au fait que les simulateurs IP ont rarement la capacité d'effectuer des rollbacks. Certains projets comme FNCS proposent un contournement du problème, en dupliquant les processus.

Si certains travaux comme le DSB insistent sur le problème des implémentations incomplètes des protocoles IP dans les différents simulateurs, on peut toutefois noter que ceux-ci ont généralement un fonctionnement relativement similaire. Ainsi, l'utilisation du modèle en couches de la suite Internet utilisé dans les réseaux IP (cf. Section 3.2.2) est souvent pris en compte dans l'architecture-même du simulateur. De la même façon, hormis quelques exceptions comme GloMoSim, les paquets IP sont généralement représentés de façon atomique dans les simulateurs. Ces généralités sur les simulateurs IP permettent de grandement faciliter l'échange de paquets simulés entre simulateurs IP, voire entre couches adjacentes comme dans le cas du DSB.

Les projets portent généralement sur l'interconnexion de simulateurs spécifiques, et l'intégration de ceux-ci est généralement ad-hoc. Le détail de l'intégration des simulateurs est également rarement fourni, et il semble dès lors compliqué d'imaginer une communauté se créer autour de ces plateformes. Cette dernière est pourtant indispensable pour créer les connecteurs nécessaires aux nombreux simulateurs existants, et ainsi permettre l'utilisation de nombreuses bibliothèques de modèles différentes dans une même co-simulation.

Enfin, on peut noter qu'aucune solution ne semble apporter de solutions à la fois pour les problèmes liés à l'interconnexion de simulateurs IP entre eux, et ceux liés à l'interconnexion de simulateurs IP à des simulateurs d'autres domaines.

# Chapitre 4

# Positionnement

**Sommaire**



## 4.1 Introduction

L'objectif de ce chapitre est de mieux préciser les problématiques auxquelles répondront les contributions présentées dans les chapitres suivants.

Nous avons pris connaissance dans le Chapitre 1 de la définition d'un système cyber-physique (SCP). Nous avons notamment vu que les SCP avaient pour caractéristiques :

1. de nécessiter l'interconnexion d'équipements hétérogènes ;
2. d'être fortement dépendants des réseaux IP ;
3. de pouvoir être qualifiés de systèmes complexes lorsqu'ils sont de grande envergure.

Nous avons également abordé l'intérêt de les modéliser et les simuler, en réutilisant des modèles (et donc des simulateurs) déjà existants. Le Chapitre 2 a permis d'établir la nécessité d'utiliser un multi-modèle hybride pour simuler des SCP grâce à une co-simulation, impliquant plusieurs simulateurs différents. Ce chapitre a également été l'occasion de découvrir différents outils utiles pour la co-simulation, comme le formalisme DEVS et la plateforme de co-simulation MECSYCO.

Étant donnée l'importance des réseaux IP pour les SCP, nous avons fait le choix dans ce manuscrit de thèse, de nous intéresser particulièrement à l'intégration des modèles et simulateurs IP, à des multi-modèles hybrides. Dans cette optique, nous avons étudié, dans le Chapitre 3, un panel de différents travaux existants, pour comprendre comment ils intégraient des modèles et simulateurs IP à des multi-modèles, et quelles ont été les difficultés rencontrées.

Ces études préliminaires nous permettent à présent de mieux cadrer les hypothèses et contraintes qu'on retient pour ce manuscrit de thèse.

## 4.2 Hypothèses

À partir de l'étude menée dans le Chapitre 3, et grâce à notre connaissance des simulateurs IP, nous choisissons de nous limiter aux simulateurs IP qui respectent les hypothèses suivantes :

- Ils reposent sur le formalisme des événements discrets.
- Leur exécution dépend d'une pile d'événements.
- Leurs modèles s'appuient sur et respectent les *Internet Standard*, définis dans le cadre des *Request For Comments* (RFC) émis par l'*Internet Engineering Task Force* (IETF) ; ce qui correspond à une représentation des couches logicielles des équipements réseaux très proche du système.
- Leurs modèles représentent les trames et paquets de façon atomique, et les nœuds et liens (avec leurs interfaces) composant la topologie de réseau sont individuellement représentés.
- Des adresses IP au format standardisé (IPv6 et/ou IPv4) peuvent être associées aux interfaces.

Nous savons que ces hypothèses sont valides au moins pour les simulateurs IP les plus utilisés par la communauté scientifique, tels que NS-3, OMNeT++/INET ou encore OpNet. Ces hypothèses nous permettront par la suite de pouvoir proposer un discours le plus générique possible, en ayant la possibilité d'éliminer les simulateurs IP dont le fonctionnement fait office d'exception.

## 4.3 Problématiques

D'après ce qui a été vu dans les chapitres précédents, intégrer un modèle IP à un multi-modèle hybride nécessite d'être capable d'intégrer :

1. l'hétérogénéité du multi-modèle, au niveau des formalismes, de la représentation des données de simulation et des implémentations logicielles utilisées ;
2. des modèles IP de façon à ce qu'ils soient en capacité de représenter le transport de données applicatives, produites par des modèles issus d'autres domaines d'expertise ;
3. des modèles IP de façon à ce qu'ils soient en capacité de se compléter, pour représenter ensemble une topologie de réseau IP complète.

Nous ajoutons également deux contraintes sur la façon dont les modèles IP doivent êtres intégrés à un multi-modèle :

1. Les modèles IP doivent pouvoir être interchangeables, de façon à pouvoir passer d'un modèle ou d'un simulateur IP à l'autre, sans modifier le reste du multi-modèle. Cette contrainte permet de s'assurer que l'intégration n'est pas ad-hoc, et qu'il est possible de comparer rapidement plusieurs modèles ou simulateurs IP, en les intégrant dans un même scénario.
2. Le code du simulateur lui-même ne doit pas être modifié, autant que faire se peut. Ainsi, une instance déjà existante d'un simulateur IP devrait pouvoir exécuter un modèle qui est impliqué dans un multi-modèle (i.e. participer à une co-simulation) sans avoir à subir de modifications préalables. Une version spécifique du simulateur ne devrait pas avoir à être maintenue.

Nous considérons qu'intégrer un modèle IP à un multi-modèle implique systématiquement d'également intégrer le simulateur associé, à la co-simulation.

## 4.4 Proposition

Nous avons vu dans le Chapitre 2 que la plateforme de co-simulation MECSYCO, grâce à l'utilisation de wrappeurs DEVS et d'opérations de transformation, était capable de nous aider à répondre à la première problématique.

Cependant, aucun simulateur IP n'a déjà fait l'objet d'une intégration à une co-simulation MECSYCO, et aucune solution n'est proposée pour réussir à wrapper en DEVS un simulateur IP et ses modèles. Pour parvenir à réaliser ce type de wrapping, nous avons besoin de définir un cadre général permettant d'éviter l'utilisation de solutions ad-hocs limitées.

Notre proposition est double :

1. Structuration des différents niveaux de problèmes pour l'intégration de modèles IP dans une co-simulation, permettant de délimiter les objectifs et contraintes du wrapping.
2. Proposition d'une stratégie de wrapping DEVS de modèles IP et leurs simulateurs.

Notre contribution permettra d'intégrer des modèles IP existants à des co-simulations hybrides, permettant ainsi de simuler en même temps les trois principaux domaines d'expertise, caractéristiques des systèmes cyber-physiques. Elle permettra en outre de donner la possibilité de représenter des réseaux IP, en mixant des sous-modèles issus de bibliothèques IP incompatibles entre elles. Le contexte industriel de la thèse implique que l'objectif de notre proposition soit plutôt orientée applications pratiques (i.e. études préliminaires pour la conception de projets industriels), que recherche expérimentale (e.g. pas de conception de protocoles réseau innovants).

Le Chapitre 5 développe le premier point de notre proposition. Les concepts proposés dans ce chapitre servent ensuite à aborder la seconde partie de notre proposition, dans le Chapitre 6. Le Chapitre 7 utilise les apports de l'ensemble de notre proposition, pour présenter des exemples concrets d'intégration de modèles IP à des co-simulations hybrides de SCP, en se basant sur deux simulateurs IP réputés.

# Chapitre 5

# Conception d'un multi-modèle avec des modèles IP

**Sommaire**

## 5.1 Introduction

Concevoir un multi-modèle nécessite de lister les modèles qui seront impliqués dans la co-simulation, et d'anticiper les connexions qui sont à établir entre eux.

Cette étape fondamentale est une étape préliminaire de niveau théorique. Concevoir un multi-modèle qui intègre des modèles IP nécessite de prendre en considération certaines spécificités liées au domaine, que nous allons étudier dans ce chapitre.

La modélisation d'un réseau IP est réalisée à partir du schéma de la topologie IP du système qui est à représenter. Selon les besoins, lorsqu'on modélise la partie réseau IP d'une co-simulation hybride, deux façons de procéder sont à distinguer :

— Si l'intégralité du réseau est représenté dans un seul modèle, il suffit de choisir un simulateur IP (et donc une bibliothèque de modèles IP) et de composer avec les sous-modèles pour le « reconstituer ». La question qui se pose est alors : *Comment ce modèle IP va-t-il interagir avec les autres modèles, qui sont issus d'autres domaines d'expertise ?* Cette question implique de comprendre ce que représenteront les données issues – et à destination de – ces autres modèles, vis-à-vis du domaine des réseaux IP.
— Comme évoqué dans le Chapitre 3, deux principales raisons peuvent motiver à utiliser plusieurs modèles distincts, dans le multi-modèle, pour représenter le réseau IP : soit parce que les (meilleurs) sous-modèles permettant de représenter les différentes technologies utilisées dans le réseau ne sont pas tous disponibles pour le même simulateur IP, soit parce qu'il est nécessaire de distribuer le modèle d'une façon non permise par le simulateur IP utilisé. Une question s'ajoute alors à celle énoncée dans le point précédent : *Comment les différents modèles IP vont-ils interagir entre eux ?* Cette nouvelle question implique de comprendre la relation entre la découpe d'une topologie IP, et l'interconnexion des différents modèles IP qui seront produits.

Le premier objectif de ce chapitre est d'identifier toutes les façons dont on pourrait souhaiter « éclater » la représentation d'un réseau IP en plusieurs modèles, en nous appuyant sur la représentation en couches des protocoles IP. Nous les catégoriserons ensuite en deux grands modes de couplage : structurel et spatial, que nous définirons précisément. Nous illustrerons dans un premier temps ces couplages pour le domaine IP, de façon naïve, sans considérer la moindre contrainte.

Dans la suite du chapitre, nous listerons les différentes contraintes qui sont à prendre en considération dès lors qu'on souhaite représenter des réseaux IP ou connecter des modèles IP. Elles nous amènerons notamment à introduire la notion de port pour les modèles IP, à comprendre à quoi ils correspondent d'un point de vue IP et où ils devraient être situés vis-à-vis de la topologie IP représentée. L'étude progressive de ces contraintes nous permettra de faire évoluer la représentation naïve des couplages, de préciser la nature du lien entre le réseau simulé et le multi-modèle, et de finalement obtenir des multi-modèles qui peuvent être transcrits du papier au code, sans difficulté.

Nous tirerons de cette étude deux principaux bénéfices :

1. une liste restreinte de modes de couplage et de leurs ports, sémantiquement définis, qui sont à prendre en considération pour les modèles IP ;
2. des solutions permettant de découper des réseaux IP, en sachant exactement à quoi correspondront les différents modèles IP à concevoir.

L'étude permettra notamment de dresser une liste d'outils qui doivent être à la disposition du modélisateur, pour qu'il soit capable de réaliser les modèles IP à interconnecter. Nous proposerons également une représentation graphique des différents concepts utilisés, afin d'être capable de les représenter au niveau méta-modèle, tout en ayant à terme une possibilité de correspondance directe au niveau du code.

La liste des contraintes présentées ne peut pas être exhaustive, et correspond uniquement à celles que nous avons identifiées en travaillant sur l'intégration de modèles IP. Elles sont toutefois suffisantes pour réussir à intégrer des simulateurs IP très utilisés, et n'ont jamais été abordées sous cet angle pragmatique dans la littérature. Ce chapitre peut par ailleurs sembler aride sous certains aspects, notamment à cause des notions IP avancées qui sont parfois nécessaires pour comprendre les limitations (les notions présentées dans la Section 3.2 doivent notamment avoir été comprises). Il permet en particulier de fournir le détail de la justification de nos choix, sur lesquels se repose notamment le chapitre suivant, qui les concrétise.

Nous avons organisé la présentation des contraintes en trois niveaux :

— **Niveau modélisation :** Ce niveau permettra de comprendre comment les propriétés des modèles (et en particulier le respect des standards Internet, hérités du système cible) limitent les possibilités sur le plan des couplages.
— **Niveau simulation :** Ce niveau s'intéressera aux modifications qui sont à faire sur le multi-modèle, pour s'assurer que les différents modèles IP à concevoir (avec leurs ports), pourront effectivement être exécutés par les simulateurs. Un certain nombre de concepts seront proposés.
— **Niveau logiciel :** Ce niveau proposera de évolutions des concepts proposés, notamment pour pouvoir contourner les limitations courantes imposées par les API des simulateurs IP.

La section suivante s'intéresse aux différents modes de couplage possibles, impliquant un modèle IP.

## 5.2 Différents modes de couplage

### 5.2.1 Introduction

Le Chapitre 4 a permis de prendre pour hypothèse que nous intégrons des modèles IP exécutés par des simulateurs qui respectent les standards Internet, définis dans le cadre des RFC de l'IETF. Nous considérons par conséquent qu'ils utilisent la structure en couches de la suite Internet, tel que décrit dans la Section 3.2.

Cette structure en couches nous permet de considérer plusieurs solutions pour concevoir un multi-modèle qui intègre des modèles IP. En prenant le cas le plus extrême, on s'intéresse d'abord au cas où l'on souhaite représenter chaque couche dans un modèle séparé.

Cet exemple nous permettra de faire ressortir deux principaux modes de couplage possibles, que nous définirons.

### 5.2.2 Deux principaux modes de couplage

#### Représentation d'une couche par modèle

D'un point de vue de la modélisation, on peut souhaiter utiliser un modèle différent pour représenter chacune des couches de la suite Internet. Les modèles seraient alors reliés entre eux au sein d'un même multi-modèle, et s'échangeraient des données de simulation de la même façon que les couches s'échangeraient des données dans le système. Ainsi, on obtiendrait un multi-modèle équivalent au schéma proposé en Figure 5.1.

Sur ce schéma, les modèles qui correspondent purement à des couches de la suite Internet sont en rouge (2-4). Ce sont les modèles qui seront en principe (mais pas obligatoirement) exécutés par des simulateurs IP. La couche 5 (applicative, en vert) sera plus généralement représentée par un modèle tiers, qui génère des données quelconques. La couche 1 (physique, en bleu), à l'autre extrémité, sera généralement représentée par un modèle équationnel, dès lors qu'on souhaite finement la représenter.

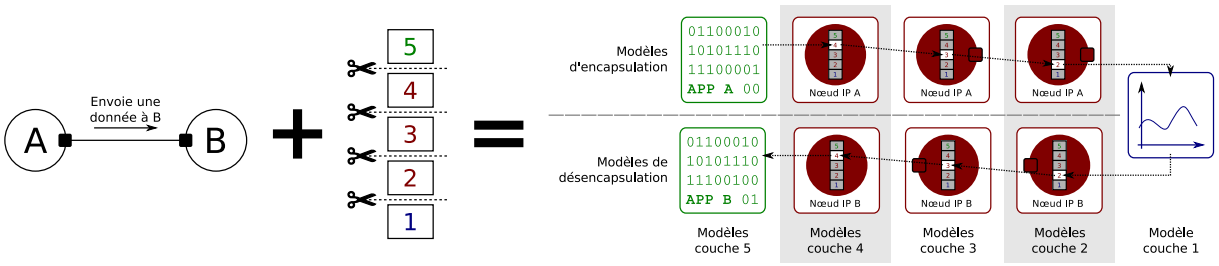



FIGURE 5.1 – Exemple simplifié de multi-modèle représentant un transfert de données d'un nœud IP à un autre, en utilisant un modèle pour représenter chacune des couches de la suite Internet, pour les différents équipements.

Les données de simulation qui seront transmises d'un modèle à l'autre seront donc progressivement des données applicatives, des datagrammes, des paquets, des trames, puis des bits, dans le sens de l'encapsulation, et inversement dans le sens de la désencapsulation. Toutefois, la transmission d'une donnée applicative d'un nœud à un autre, ne peut pas se réduire à transmettre l'information d'une couche à l'autre.

Selon la couche et le protocole modélisé, les modèles peuvent être amenés à effectuer des échanges intermédiaires, à leur propre initiative : par exemple pour trouver une adresse MAC à partir d'une adresse IP, ou pour découvrir le MTU du plus petit du chemin à parcourir (cf. RFC 1191), des trames intermédiaires devront être émises.

Le nombre de paquets ou de datagrammes reçu par le modèle qui désencapsule peut aussi être supérieur à celui envoyé par le modèle qui encapsule, par exemple en cas de segmentation (couche 4) ou de fragmentation (couche 3). À l'inverse, ce qui est envoyé par un modèle ne sera pas obligatoirement reçu par un autre, par exemple si le modèle de la couche 3 détermine qu'il n'y a aucune route correspondant au paquet qu'elle aurait à transmettre, si aucune adresse MAC ne correspond à l'adresse physique, ou si le lien en couche 1 est considéré comme étant inutilisable.

Enfin, si un nœud a plusieurs interfaces réseau à sa disposition, les modèles de couche 3 peuvent être interconnectés à plusieurs modèles de couche 2, et les modèles de couche 2 peuvent

être interconnectés à plusieurs modèles de couche 1.

Cet exemple simplifié, mais extrême, permet d'identifier différentes façons de coupler des modèles IP entre eux, et avec des modèles tiers.

#### Regroupement par type de couplage

Dans l'exemple de la Figure 5.1, on peut distinguer trois parties différentes :

1. le groupe de modèles qui à eux tous représentent l'intégralité des fonctionnalités de A (les quatre du haut) ;
2. le modèle qui représente la couche physique du lien qui relie A et B ;
3. la groupe de modèles qui à eux tous représentent l'intégralité des fonctionnalités de B (les quatre du bas).

Les deux groupes de modèles et le modèle du lien physique représentent trois équipements réseaux différents : les nœuds A et B et le lien. On comprend dès lors que le couplage entre les différents modèles d'un même groupe, et le couplage entre les groupes et le lien, ne sont pas de même nature. Les couplages des modèles au sein des groupes servent à représenter un même équipement, composé de plusieurs niveaux de fonctionnalités. Les couplages entre les groupes et le modèle de lien servent à représenter des connexions physiques entre équipements, qui existent de la même façon dans le système cible.

Dans le cadre de ce manuscrit, on choisit de distinguer ces deux types de couplage, qu'on nommera *structurels* (au sein d'un groupe) et *spatiaux* (entre les groupes). Les sections suivantes ont pour objectif de proposer une définition claire de ces deux types de couplage, qui peuvent intervenir dans un multi-modèle impliquant des modèles IP.

### 5.2.3 Couplages structurels

#### Définition proposée

Un couplage structurel entre modèles permet de représenter un même système, au même instant, mais selon des points de vue différents. Les points de vue peuvent être différents en fonction de l'échelle choisie (e.g. macroscopique versus microscopique), du mode de perception choisi (e.g. en couleur ou en noir et blanc) ou encore du domaine d'expertise pris en considération (e.g. modélisation focalisée sur les propriétés thermiques du système, plutôt que sa mobilité).

Si les différents points de vue ne sont pas indépendants les uns des autres, les modèles ont besoin de s'échanger régulièrement des données sur l'état du système représenté, afin de pouvoir évoluer ensemble et s'influencer mutuellement durant la co-simulation. Une illustration abstraite de multi-modèle utilisant des couplages structurels est proposée dans la Figure 5.2.

Cette notion est inspirée des couplages structurels utilisés par [Desmeulles, 2006], et initialement proposés par [Francisco, Varela, 1989].

#### Application aux modèles IP

Dans le contexte des réseaux IP, les couplages structurels permettent de représenter les différentes couches de la suite Internet, dans des modèles séparés. Les modèles s'échangent alors

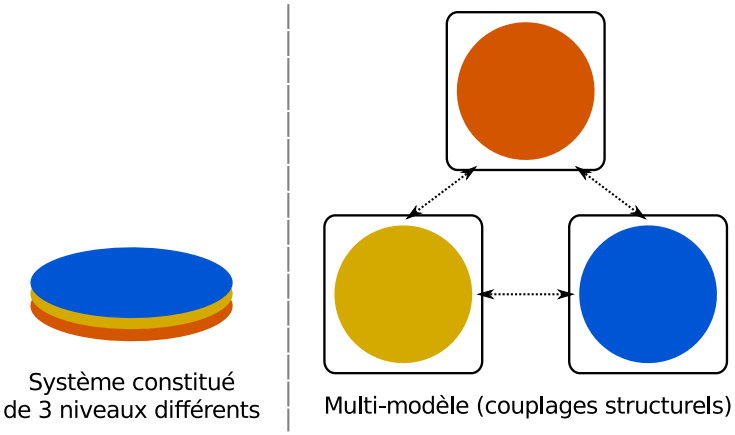



FIGURE 5.2 – Multi-modèle (à droite) composé de trois modèles, tous structurellement couplés entre eux et représentant chacun un point de vue du système (à gauche).

des données lorsqu'il est nécessaire de simuler une encapsulation, une désencapsulation ou le passage de bits sur un lien.

Ce type de couplage est particulièrement utile pour simuler le transfert sur un réseau IP de données générées par un modèle quelconque, comme illustré dans l'exemple de la Figure 5.3 (le réseau donné dans cet exemple est volontairement minimaliste). Dans cet exemple, les couches 5 (applicatives) des deux nœuds qui communiquent ensemble, sont séparées. Le modèle central (en rouge) est un modèle IP qui représente l'intégralité de la topologie de réseau IP (les deux nœuds et le lien). Les modèles latéraux (en vert) représentent uniquement la partie applicative des nœuds A et B. Les nœuds A et B sont donc représentés deux fois, avec deux points de vue différents, correspondant à deux domaines d'expertise différents : en tant qu'équipements IP dans le modèle central, et en tant qu'applications (de façon plus générale en tant que « générateurs ou consommateurs de données applicatives », pouvant se reposer sur des capteurs) dans les modèles latéraux.

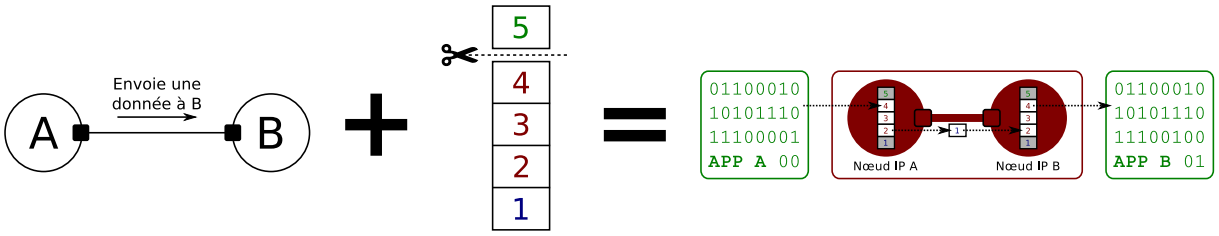



FIGURE 5.3 – Exemple de couplage structurel, avec la topologie de réseau IP représentée par un modèle IP (en rouge) et la partie application des nœuds représentée par des modèles tiers (en vert).

Dans cet exemple, les données sur l'état du système qui sont échangées entre les modèles correspondent à des données applicatives. Le modèle IP prend connaissance des changements d'état de la partie applicative des nœuds A et B lorsque ceux-ci ont un impact sur la partie réseau du système (e.g. simulation de l'envoi de données dans une socket réseau) et les modèles applicatifs prennent connaissance des changements d'état du réseau quand ceux-ci ont un impact

sur les applications (e.g. simulation de l'arrivée de données via une socket réseau).

Les couplages effectués dans les travaux présentés dans la Section 3.5 pourraient également être qualifiés de structurels.

### 5.2.4 Couplages spatiaux

#### Définition proposée

Un couplage spatial entre modèles permet de représenter un système complet à partir de modèles qui ne contiennent chacun que la représentation d'un fragment physique de celui-ci. Durant la co-simulation, les différents modèles vont s'échanger des données correspondant aux échanges entre les différents fragments, qui auraient lieu dans le système aux mêmes instants.

Par exemple, si un multi-modèle représente un pays et que les différents modèles sous-jacents (spatialement couplés) représentent des villes, et qu'on s'intéresse au trafic urbain, les modèles de deux villes adjacentes s'échangeront des données durant la co-simulation dès lors que des voitures seraient censées passer d'une ville à l'autre, dans le système. Une illustration abstraite de multi-modèle utilisant des couplages spatiaux est proposée dans la Figure 5.4.

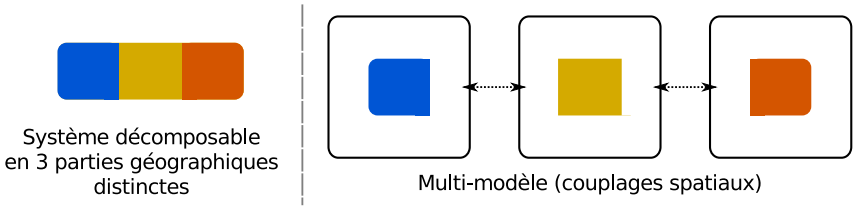



FIGURE 5.4 – Multi-modèle (à droite) composé de trois modèles, représentant chacun un fragment physique du système (à gauche), et spatialement couplés lorsque les fragments représentés sont adjacents dans le système.

#### Application aux modèles IP

Dans le contexte des réseaux IP, les couplages spatiaux permettent de découper la topologie du réseau à représenter, pour répartir les morceaux dans différents modèles. Ces modèles seront donc couplés de la même façon que les équipements représentés à l'intérieur sont reliés dans le système.

Dans l'exemple de la Figure 5.5, on sépare chaque équipement du réseau, pour tous les représenter dans des modèles individuels (cas extrême). Le modèle de gauche représente la machine correspondant au nœud A, le modèle du milieu représente le lien entre les deux machines (avec ses cartes réseau), et le modèle de droite représente la machine correspondant au nœud B.

Lorsque A envoie une donnée à B, le modèle de A transmet un paquet simulé au modèle de lien, qui simule son transfert sur le lien. Une fois le transfert terminé, dans le temps simulé, le modèle de lien transmet une trame simulée au modèle de B.

Les couplages effectués dans les travaux présentés dans la Section 3.3 et 3.4 pourraient également être qualifiés de spatiaux.

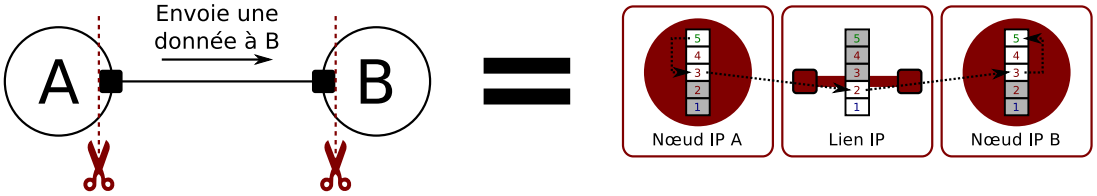



FIGURE 5.5 – Exemple simple de couplage spatial, avec un modèle différent pour représenter de façon séparée chacun des deux nœuds interconnectés et le lien qui les relie.

### 5.2.5 Conclusion

À partir de la représentation en multi-modèle d'un réseau IP minimaliste, et en prenant soin d'utiliser un modèle individuel pour représenter chaque couche de la suite Internet, nous avons pu constater intuitivement qu'il existait au moins deux sortes de couplage pouvant impliquer un modèle IP.

Nous avons ensuite formalisé ces deux approches, en les nommant, en les définissant de façon générale, et en les illustrant avec un exemple concret appliqué aux modèles IP. Ces deux approches sont par ailleurs combinables, comme l'illustre la Figure 5.6.

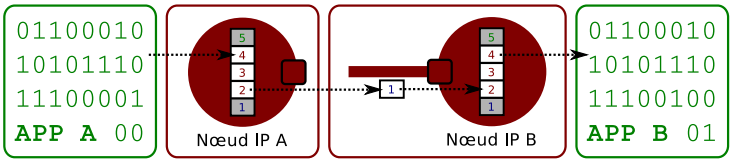



FIGURE 5.6 – Exemple de modèles couplés à la fois structurellement et spatialement.

Les définitions et les exemples donnés permettent de constater que les modèles n'interagissent pas de la même façon selon le mode de couplage utilisé. Ces différents modes d'interaction amènent à considérer différents modes d'intégration : on parlera donc d'intégration structurelle ou spatiale. La prise en compte de différentes contraintes d'intégration, parfois spécifiques à un mode de couplage en particulier, permettent de spécifier comment les topologies IP doivent être découpées et comment les modèles doivent s'interconnecter entre eux. Ces contraintes sont exposées en trois niveaux : modélisation, simulation et logiciel.

## 5.3 Contraintes de modélisation

Les contraintes de modélisation concernent les limitations dans les couplages, qui sont imposées par les propriétés des modèles. Les modèles IP doivent notamment pouvoir individuellement respecter les standards Internet, y compris lorsqu'ils sont impliqués dans un couplage avec un autre modèle.

### 5.3.1 Dépendances inter-couches

L'organisation en couches des protocoles de la suite Internet permet théoriquement de remplacer une couche $L^i$ par une couche $\mathcal{L}^j$ dès lors que $i = j$ (i.e. le protocole associé est considéré de même niveau), sans modifier les autres couches. En représentant les couches individuellement dans des modèles séparés, reliés par des couplages structurels, le multi-modèle devrait bénéficier de cette modularité. Il serait ainsi possible de remplacer un modèle de couche par un autre, pour comparer les résultats de la co-simulation avec plusieurs protocoles différents à un niveau donné, sans modifier le reste du multi-modèle.

Pour coupler structurellement un modèle IP avec un autre modèle, on pourrait considérer chaque couche de la suite Internet comme un point de vue différent sur le système, et admettre qu'on peut créer un modèle séparé pour chaque couche à représenter, comme cela a été illustré dans la Figure 5.1.

Cette vision idéaliste du modèle en couches utilisé pour la suite Internet est cependant contrainte par de nombreuses exceptions, documentées dans les standards Internet. Ces exceptions, qui créent des dépendances entre les couches, sont autant de contraintes de modélisation à prendre en considération lorsqu'on souhaite concevoir un multi-modèle IP avec des couplages structurels. Les principales dépendances entre couches IP adjacentes sont listées ci-dessous.

Dépendances entre la couche 4 et la couche 3 :

— Les entêtes IP de la couche 3 contiennent un champ qui doit indiquer quel est le protocole utilisé au niveau 4. En IPv6, c'est le champ *Next Header* (cf. RFC 2460) et en IPv4 c'est le champ *Protocol* (cf. RFC 791). La valeur du champ doit correspondre au numéro d'identification du protocole de la suite Internet, défini dans la RFC 790.
— L'entête du protocole UDP (couche 4) contient un champ nommé *Checksum* correspondant à une somme de contrôle. Celui-ci est facultatif si le protocole de couche 3 est IPv4, mais devient obligatoire si c'est IPv6 (cf. RFC 2460).
— L'entête du protocole TCP (couche 4) contient également un champ *Checksum*, similaire à celui de UDP. En TCP comme en UDP, la somme de contrôle est calculée à partir d'informations qui nécessitent de connaître la version de IP et les adresses IP qui seront utilisées dans la couche 3 (cf. *pseudo-headers* IP dans les RFC 2460 & 793).

Dépendances entre la couche 3 et la couche 2 :

— L'entête du protocole Ethernet (couche 2) contient un champ nommé *EtherType*, qui doit indiquer le protocole utilisé au niveau 3, en utilisant un numéro d'identification (cf. RFC 5342).
— L'entête du protocole Ethernet peut également contenir un champ complété avec un numéro de VLAN (numéro de réseau local virtuel, IEEE 802.1Q). Pour connaître le numéro de VLAN à utiliser (s'il y en a un), il est nécessaire connaître l'interface réseau de sortie qui sera utilisée pour envoyer la trame. Le choix de l'interface de sortie dépend de la décision prise au niveau 3, en utilisant la table de routage.
— L'entête du protocole Ethernet contient également des adresses MAC (physiques) source et destination. Pour connaître l'adresse source à utiliser, il est de nouveau nécessaire de connaître l'interface réseau de sortie qui sera utilisée. Pour connaître l'adresse de destination, il est nécessaire de connaître soit (1) l'adresse MAC calculée au niveau 3 via le protocole NDP (cf. RFC 4861) si IPv6 est utilisé à ce niveau, soit (2) l'adresse IP de destination pour pouvoir trouver l'adresse MAC via le protocole ARP (cf. RFC 826) si

IPv4 est utilisé au niveau 3.

Dépendances entre la couche 2 et la couche 1 :

— La couche de niveau 1 utilise un algorithme de codage de l'information différent, selon le protocole utilisé au niveau 2. Par exemple, le codage Manchester est utilisé lorsque Ethernet (configuré en 10 Mb/s) est utilisé.
— La méthode d'accès au média utilisé par la couche 2 diffère selon le type de lien physique rencontré au niveau 1 (e.g. détection ou évitement des collisions).

Ces dépendances sont souvent spécifiques aux protocoles qui sont largement utilisés dans la suite Internet. On peut supposer que l'introduction d'un protocole expérimental pourrait ajouter des dépendances différentes. Les dépendances listées ci-dessus sont résumées dans la Figure 5.7.

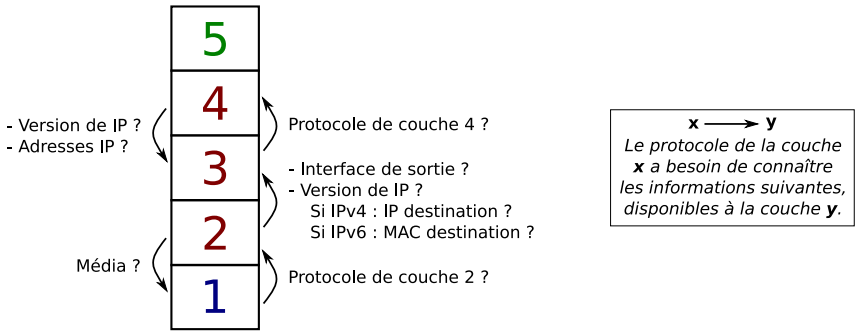



FIGURE 5.7 – Résumé des principales dépendances qui existent entre les couches de la suite Internet, d'après les standards des principaux protocoles utilisés.

Enfin, les protocoles de résolution des adresses physiques (e.g. trouver une adresse MAC à partir d'une adresse IP) sont à la bordure des couches 2 et 3. Si NDP (protocole de résolution pour IPv6 basé sur ICMPv6) serait logiquement situé au niveau 3 parce qu'il a d'autres attributions à ce niveau, ARP (protocole de résolution pour IPv4) serait plus logiquement situé au niveau 2 dans la mesure où il ne répond qu'à ce besoin et qu'il est très lié au protocole Ethernet. ARP est par ailleurs parfois décrit comme étant un protocole de couche 2 ½. Nous avons suivi cette logique en décrivant les dépendances ci-dessus, mais rien n'assure que les différentes bibliothèques de modèles IP ont toutes attribué ces fonctions aux mêmes couches, ce qui peut entraîner des incompatibilités dans le cas d'un couplage structurel.

L'ensemble des contraintes définies dans les protocoles de la suite Internet, qui créent des dépendances entre les couches, sont donc à prendre en considération dès lors qu'on souhaite structurellement coupler deux modèles IP représentant chacun une couche IP adjacente.

### 5.3.2 Découpage d'une topologie de réseau

Il existe deux solutions pour déterminer où couper une topologie de réseau IP, afin de la séparer en deux, dans le cadre d'un couplage spatial :

1. couper entre deux équipements de couches adjacentes ;
2. couper au milieu d'une couche.

**Le premier cas** a été illustré dans la Figure 5.5 (premier exemple de couplage spatial, page 68), en tant qu'exemple de couplage spatial pour des modèles IP. Dans cet exemple, les coupures interviennent entre les nœuds et les cartes réseau permettant de connecter le lien. La coupure intervient donc entre les couches 3 (couche IP de la machine) et 2 (carte réseau, e.g. Ethernet). La seule autre coupure de ce type qui aurait été possible, aurait été entre les cartes réseau (couche 2) et le lien (couche 1), par exemple pour représenter la transmission physique des bits sur le média, avec un modèle équationnel dédié (les modèles reposant sur des lois mathématiques étant généralement plus appropriés pour représenter des phénomènes de type physique).

Les contraintes de modélisation liées aux dépendances inter-couches, exposées ci-avant, peuvent s'appliquer de la même façon pour ce premier cas.

L'exemple de la Figure 5.5 aurait pu être étendu, en remplaçant le lien simple entre les deux nœuds, par un réseau de commutation (couche 2) complet. Dans ce cas, le modèle n'a théoriquement pas besoin d'être capable de comprendre les protocoles supérieurs à la couche 2, qui sont contenus dans les trames simulées qu'il manipule. Il suffirait par contre qu'un seul routeur intervienne dans le modèle, pour que celui-ci soit obligatoirement en capacité de comprendre le protocole de couche 3 qui est encapsulé dans les trames.

De façon générale, dans le cadre d'un couplage spatial, c'est l'équipement de plus haut niveau et celui de plus bas niveau, qui déterminent dans quel intervalle de couches les modèles ont besoin d'avoir des protocoles en commun. Un exemple d'intervalle minimum de compatibilité est illustré dans la Figure 5.8.

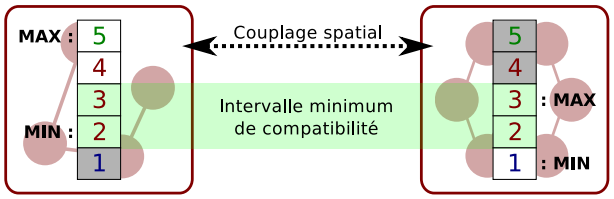



FIGURE 5.8 – L'intervalle minimum de compatibilité entre deux modèles IP spatialement couplés (i.e. les niveaux correspondant aux protocoles qu'il doivent au minimum avoir en commun), est déterminé par les niveaux minimum et maximum utilisés par les équipements simulés dans les modèles.

**Le second cas** consiste à découper une couche en plusieurs parties, pour répartir ses fonctions dans des modèles distincts. Ce cas intervient lorsqu'on souhaite couper une topologie de réseau au niveau d'un nœud de transit (e.g. commutateur ou routeur). La Figure 5.9 donne un exemple de coupure au milieu d'un routeur, avec une coupure en deux du nœud. Dans ce cas, on va répartir la représentation des deux interfaces réseau sur les deux modèles, en représentant une partie de la couche 3 (et donc du routeur) dans chacun des modèles. Les deux modèles sont donc reliés au niveau de la couche 3, et représentent chacun deux réseaux IP logiques. Ils auraient été reliés au niveau de la couche 2, si le nœud de transit avait été un commutateur (ce qui aurait permis dans ce cas, par exemple, de séparer la représentation des ports pour câbles en cuivre de ceux pour fibres optiques). On considère qu'un nœud de transit n'est pas censé émettre lui-même de données, ni en recevoir pour lui-même, mais se contenter de les transmettre d'une interface à l'autre, pour le niveau auquel il se définit.

Dans ce second cas, les contraintes de modélisation interviennent au niveau de la composition

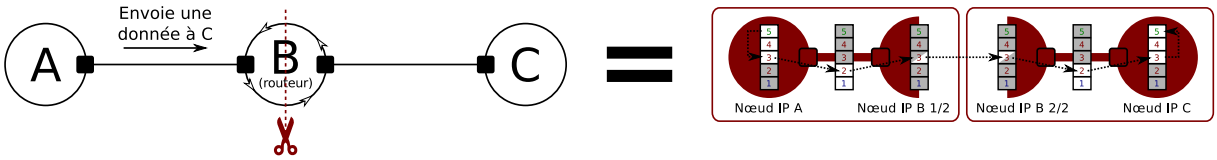



FIGURE 5.9 – Exemple simple de couplage spatial, avec un modèle différent pour représenter deux réseaux IP différents, qui sont interconnectés via un routeur.

de l'état commun que doivent partager les deux parties de la couche modélisée.

Si la coupure intervient sur une couche de niveau 3, l'état commun devrait principalement contenir :

— la file d'attente ;
— la table de routage ;
— les abonnements aux groupes multicast.

Si la coupure intervient sur une couche de niveau 2, l'état commun devrait principalement contenir :

— la table d'association entre les interfaces physiques et les adresses MAC ;
— les abonnements aux groupes multicast, si le commutateur supporte le *MLD Snooping* (IPv6) ou l'*IGMP Snooping* (IPv4) – cf. RFC 4541.

On considère que la couche sur laquelle on coupe est celle de plus haut niveau que propose le nœud de transit, et qu'on coupe ensuite l'intégralité du nœud, jusqu'à la couche physique (de façon à également séparer les interfaces).

### 5.3.3 Cartes réseau

Dans le système, les cartes réseau sont généralement associées à un certain type de lien, et inversement :

— une carte Ethernet « classique », reconnaissable avec ses ports RJ45, doit en principe accueillir un câble en cuivre équipé de paires torsadées ;
— une carte optique va, elle, accueillir un lien de type fibre optique ;
— une carte CPL (Courant Porteur en Ligne) va accueillir un câble électrique domestique ;
— une carte Wifi va émettre son propre « lien », sous forme d'ondes spécifiques ;
— etc.

Cela signifie qu'un simulateur IP qui propose un modèle pour un certain type de carte réseau a toutes les chances de proposer également un modèle pour le type de lien qui est associé, et inversement.

Un modèle de lien est également difficilement indépendant, dans la mesure où la couche 1 (physique) est à cheval entre le lien et la carte réseau : c'est la carte réseau qui va coder les bits et les transformer en impulsions électriques ou lumineuses, pour les transmettre sur le lien. Au niveau logiciel, les modèles de lien sont généralement plus ou moins confondus avec les modèles des cartes réseau, et ne peuvent pas être utilisés séparément.

Un couplage spatial entre deux modèles IP, l'un représentant une carte réseau et l'autre représentant un lien, n'est donc généralement pas utile ni réalisable.

## 5.4 Contraintes de simulation

Les contraintes de simulation concernent les conditions à respecter pour que les modèles IP puissent être correctement exécutés par les simulateurs. Elles sont à prendre à considération au moment de la conception du multi-modèle, pour définir les différents réseaux (qui pourront en réalité représenter des fragments d'un plus grand réseau, dans le cadre des couplages spatiaux) qui seront à modéliser.

### 5.4.1 Cas des demi-liens

Une façon de couper un réseau en deux, qui peut sembler intuitive de prime abord, est de couper au milieu d'un lien. Dans ce cas, en considérant un lien de longueur $\mathcal{L}$ à modéliser, un premier modèle représentera un morceau de lien de longueur $\mathcal{L}/2$, et un second modèle représentera l'autre morceau, de même longueur. Ces deux (demi-)liens seront reliés en série, via le multi-modèle.

La modélisation d'un lien consiste principalement à être capable de déterminer le temps qui serait nécessaire pour qu'un message (i.e. une certaine quantité de bits) passe d'une extrémité à l'autre. Ce temps est qualifié de latence, et sa formule est donnée dans l'Équation 5.1.

$$latence_{message} = (taille_{message}/débit_{lien}) + (longueur_{lien}/vitesse\_propagation_{lien}) \quad (5.1)$$

La première partie de l'équation correspond au temps de transmission, tandis que la seconde partie correspond au temps de propagation. La longueur du lien n'intervient que dans le calcul du temps de propagation. Or, celui-ci est souvent considéré comme négligeable, tant son impact est faible en comparaison avec le temps de transmission (exception faite de cas particuliers comme les connexions satellites). Le temps infime qu'il représente n'est souvent pas non plus représentable par les simulateurs IP. Par conséquent, les modèles de lien n'incluent généralement pas la longueur des liens dans leurs propriétés. Ainsi, par exemple, le code utilisé pour modéliser un lien pair-à-pair dans NS-3 est `bits/m_bps`, ce qui correspond à un simple calcul de temps de transmission. La notion de longueur de lien n'étant pas utilisée dans le modèle, il n'est donc pas possible de représenter un demi-lien.

Il n'est pas non plus possible de considérer qu'un lien de débit $\mathcal{D}$ pourrait être équivalent à deux (demi-)liens de débit $\mathcal{D}/2$ mis bout à bout.

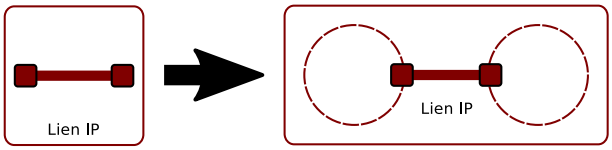
Un lien doit donc être considéré comme un élément atomique et ne peut être représenté que dans son intégralité, dans un seul modèle.

### 5.4.2 Réseaux complets

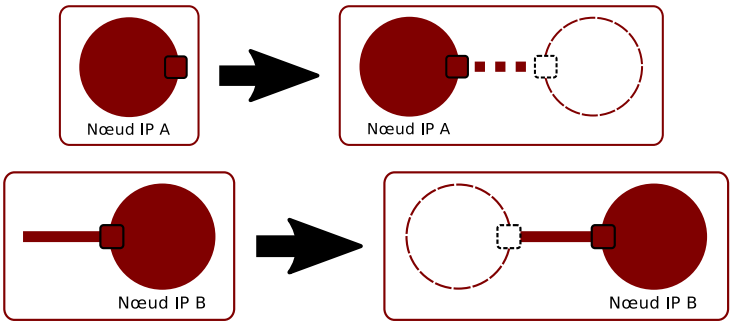
Les simulateurs IP ne permettent généralement pas de représenter des morceaux isolés de réseau, comme c'est nécessaire dans le cas de la conception de modèles dans le cadre d'un couplage spatial. Ainsi, par exemple, représenter un lien qui n'est relié à aucun nœud, comme dans le cas de la Figure 5.5 (premier exemple de couplage spatial, page 68), n'est généralement pas possible. L'exemple de la Figure 5.6 (premier exemple de modèle à la fois spatialement et structurellement couplé, page 68), avec d'un côté une carte réseau qui n'est connectée à

aucun lien et de l'autre côté un lien qui n'est attaché qu'à une seule carte réseau, n'est pas non plus possible. De la même façon, le concept de demi-couche, ou de demi-nœud, comme proposé dans la Figure 5.9 (second cas de découpe dans les contraintes de modélisation, page 72), n'est pas simulable en l'état. Des contraintes de simulation s'ajoutent alors aux contraintes de modélisation.

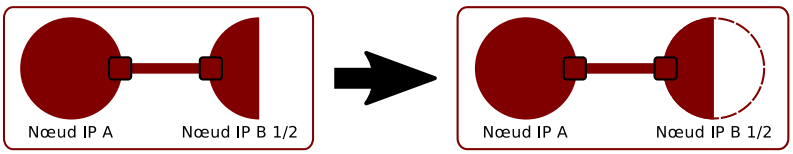
Nous partons du postulat que chaque modèle doit obligatoirement représenter un segment de réseau complet et cohérent, pour que le simulateur puisse l'exécuter. Pour pouvoir adapter les modèles théoriques des Figures 5.5, 5.6 et 5.9 à cette nouvelle contrainte, il est nécessaire d'adapter les modèles, de façon à « compléter » les fragments de réseau. Les ajouts minimum à effectuer sont visibles dans la Figure 5.10. On constate qu'il a été nécessaire d'ajouter deux types d'éléments : des nœuds supplémentaires (dans les trois exemples), et des liens supplémentaires (Figure 5.10b). La modification de la Figure 5.10c consiste à remplacer un concept qui n'est pas réalisable (un demi-nœud), par le concept le plus proche qui est susceptible d'être disponible (un nœud complet).




(a) Ajout de nœuds supplémentaires, dans l'un des modèles utilisés par l'exemple de la Figure 5.5 page 68.




(b) Ajout de nœuds et d'un lien, dans les modèles IP utilisés par l'exemple de la Figure 5.6 page 68.




(c) Utilisation d'un nœud complet, dans le premier modèle utilisé par l'exemple de la Figure 5.9 page 72 (le second modèle devant être complété de la même façon).

FIGURE 5.10 – Adaptation de modèles représentant des morceaux de réseau IP, de façon à ce qu'ils représentent chacun un réseau IP complet et cohérent, et donc simulable. Les éléments ajoutés sont dessinés en pointillés.

Les éléments en pointillés de la Figure 5.10 sont autant d'éléments, qui n'existent pas dans le système, et qui n'ont été ajoutés que pour respecter les contraintes de simulation. La conséquence est que, dans le multi-modèle, les nœuds du système par exemple, sont représentés deux fois. Dans la Figure 5.10a, les deux nœuds qu'on ajoute correspondent aux nœuds A et B de l'exemple

de la Figure 5.5 (page 68), qui continuent pourtant d'être représentés individuellement dans les modèles séparés. Il convient donc de modéliser ces appendices aux réseaux de façon particulière, de façon à ce qu'ils soient « transparents » vis-à-vis des résultats de simulation.

Ainsi, les appendices ne doivent :

1. avoir aucun impact sur les trames ou paquets qui arriveraient jusqu'à eux (aucune modification des données, quelle que soit la couche) ;
2. être à l'origine d'aucun événement de simulation, qui ferait avancer l'horloge du simulateur (consommation de temps de simulation) ;
3. avoir aucun rôle particulier dans le réseau simulé (routage, proxy, etc), puisqu'ils ne sont pas censés exister.

Par commodité, on qualifiera ces appendices de « faux ». Ainsi, on ajoutera de faux nœuds et de faux liens, de façon à compléter les réseaux.

### 5.4.3 Lookahead nul

La notion de lookahead est été abordée dans la Section 5.4.3. Nous avons également vu dans la Section 2.5.5 que certains algorithmes de synchronisation du temps, comme celui utilisé par MECSYCO, n'étaient pas capables d'accepter des modèles configurés avec un lookahead nul (i.e. à zéro).

Il est possible d'éviter les modèles à configurer avec un lookahead nul, en anticipant cet aspect dès leur conception. Le premier modèle, dans sa version corrigée, de l'exemple de la Figure 5.10b (en haut à droite), pourrait par exemple poser un problème. En effet, sur les trois principaux éléments composants le modèle (le nœud A, le faux lien et le faux nœud), deux n'auront aucun impact sur le temps de simulation (les appendices), par définition. Le calcul du lookahead ne dépendra donc que de l'application simulée qui est installée sur le nœud A. S'il s'agit, par exemple, d'un serveur ICMP qui renvoie immédiatement un paquet lorsqu'il en reçoit un, l'action est censée être tellement rapide qu'il est fort probable que le modèle de l'application ne consomme aucun temps de simulation. Le lookahead du modèle complet sera donc nul.

La seule solution pour éviter ce type de cas de figure, c'est de prendre en compte cette contrainte dès la conception des modèles : lorsqu'on sépare une topologie de réseau en plusieurs modèles dans le but de faire des couplages spatiaux entre eux, il faut prévoir d'avoir au moins un élément dans chaque modèle qui consomme à coup sûr du temps de simulation (traitement applicatif ou traversée d'un lien), entre l'arrivée d'une trame ou d'un paquet externe et la transmission de la réponse.

Dans le cadre des couplages structurels, cette contrainte peut être encore plus forte. Dans l'exemple de la Figure 5.1, avec des couplages structurels entre chaque couche, certains modèles se contenteront d'encapsuler ou de désencapsuler les données reçues, avant de les transmettre au modèle de la couche suivante. Ces opérations sont tellement rapides dans le système, qu'elles ne peuvent pas représenter de temps de simulation dans les modèles. Ainsi, un modèle de couche de la suite Internet qui se contente d'encapsuler ou de désencapsuler, doit être configuré avec un lookahead nul. De la même façon, c'est au modélisateur de faire en sorte que ses modèles ne soient pas dans cette situation, quitte à devoir en rassembler quelques-uns, pour s'assurer qu'il y aura toujours au moins une action qui consommera du temps de simulation.

## 5.5 Contraintes logicielles

Une fois les contraintes de modélisation et de simulation prises en considération pour préparer les futurs modèles, la dernière étape est de prendre en compte les contraintes purement logicielles. Celles-ci sont difficilement généralisables, parce qu'elles dépendent spécifiquement du simulateur IP utilisé pour implémenter et exécuter les modèles IP, et peuvent donc être extrêmement variées. Les contraintes logicielles sont directement liées à la notion de ports, telle que présentée dans la Section 2.4.1, qui permettent aux modèles de communiquer ensemble au sein du multi-modèle.

### 5.5.1 Emplacement des ports dans le modèle IP

L'emplacement des ports, dans le modèle, dépend de la couche concernée. Ces ports ont pour charge de recevoir des données (des bits aux données applicatives, en passant notamment par des paquets) envoyées par d'autres modèles, et de les transmettre à ladite couche. Ils sont également en charge d'intercepter les mêmes types de données, générées par leur modèle, pour les transmettre à d'autres modèles du multi-modèle. Lorsque le port sert de connecteur pour un couplage spatial, il est identifié comme un « port spatial », et lorsqu'il sert de connecteur pour un couplage structurel, il est identifié comme un « port structurel ».

En se basant sur des concepts purement réseau et en fonction des modèles, on propose les emplacement suivants pour les ports :

— Transmission de données applicatives (port de couche 5) : depuis une application simulée.
— Transmission de datagrammes (port de couche 4) : depuis une application simulée, et en utilisant des *sockets raw* simulées (réception de datagrammes non-désencapsulés, au niveau applicatif).
— Transmission de paquets ou trames (port de couche 3 ou 2) : depuis une carte réseau simulée (en tant qu'interface réseau logique).
— Transmission de bits/impulsions (port de couche 1) : depuis une carte réseau simulée (en tant que carte physique).

Les cartes réseau [6] sont les emplacements les plus à même de permettre un accès facile aux couches 3, 2 et 1 du nœud. Dans l'exemple de la Figure 5.5 (premier exemple de couplage spatial, page 68), les modèles représentant les nœuds de façon individuelle ne comportent pas de carte réseau. Il est donc nécessaire d'en ajouter une pour chaque nœud, afin de pouvoir y placer un port spatial. En tenant compte à la fois des contraintes de simulation et des contraintes logicielles, l'exemple de la Figure 5.5 évolue donc de la façon proposée dans la Figure 5.11. On peut constater que les nœuds des modèles latéraux ont désormais de fausses interfaces réseau. En raison des contraintes de simulation, ces interfaces réseau sont reliées à de faux liens, et ces faux liens sont eux-mêmes reliés à de faux nœuds.

6. Le terme de « carte réseau » est censé faire référence à la version physique de la carte, tandis que le terme « interface réseau » est plutôt censé faire référence à sa version logique (i.e. l'interface du type *eth0* qui sera visible dans le système, au moment de lister les accès réseau, grâce au pilote logiciel de la carte). Dans ce manuscrit, nous utiliserons les deux termes de façon indistincte. D'un point de vue modélisation, il n'y a généralement qu'un seul et même modèle pour représenter les interfaces réseau à la fois d'un point de vue physique et logique.

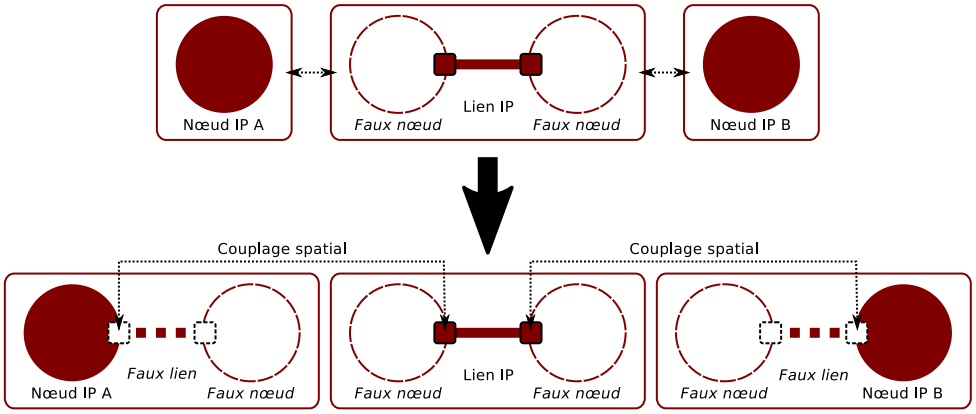



FIGURE 5.11 – Exemple de la Figure 5.5 (en haut – respectant les contraintes de simulation) adapté pour respecter en plus les contraintes logicielles (en bas).

### 5.5.2 Contraintes de l'API

L'exemple de la Figure 5.11 (version corrigée en bas) indique que des ports spatiaux sont situés au niveau des cartes réseau des nœuds A et B. Il est donc nécessaire que les ports soient capables d'intercepter les trames ou paquets que A souhaite transmettre à B via sa carte réseau et inversement. Il est aussi nécessaire que ces mêmes ports soient également capables d'injecter dans la carte réseau (et donc les couches 2 ou 3 de A ou B) des trames ou paquets venus de l'extérieur via le multi-modèle. Bien que les détails concernant la création de ports dans des modèles IP existants seront traités dans la Section 6.2, un certain nombre de contraintes logicielles courantes et liées aux API, peuvent être identifiées ici.

#### Dépendances inter-couches

Les contraintes logicielles liées aux API ajoutent notamment des dépendances inter-couches, en plus de celles identifiées au niveau des contraintes de modélisation. Les simulateurs dont le code fonctionne avec la même logique que le modèle en couches de la suite Internet (un objet par couche), proposent généralement des API qui comprennent des fonctions semblables à celles proposées dans le Tableau 5.12.

On constate que pour chacune des couches en dessous de la couche applicative, les fonctions des objets correspondants manipulent à la fois le type de donnée qui leur propre, et celui de la couche supérieure. Le modèle d'une couche $\mathcal{X}$ doit donc être compatible avec le modèle de la couche $\mathcal{X}+1$ (avec $1 \geq \mathcal{X} < 5$), dans le cas d'un couplage structurel entre couches IP, ou dans le cas d'un couplage spatial entre deux couches adjacentes.

Par exemple, la couche 2 représentée par L2 doit connaître la notion de trame (de type `FRAME`), mais aussi de paquet (de type `PACKET`). Lorsque le modèle de la couche L3 envoie un paquet IPv6 au modèle de L2 pour que celui-ci l'encapsule, le paquet reçu doit être proposé au format `PACKET` (de L2) à la fonction `encapsulate` (de L2). Si le simulateur utilisé pour L2 possède un modèle pour IPv4 mais pas pour IPv6, le seul type `PACKET` qu'il connaît correspond au format IPv4. Le paquet reçu ne peut donc pas être converti en objet de type `PACKET` dans L2, et il est impossible d'utiliser la fonction `encapsulate` de L2.

| Objet | Fonctions | |
|---|---|---|
| L5 | `DATA_APP getData();` | `VOID setData(DATA_APP);` |
| L4 | `DATAGRAM encapsulate(DATA_APP);` | `DATA_APP decapsulate(DATAGRAM);` |
| L3 | `PACKET encapsulate(DATAGRAM);` | `DATAGRAM decapsulate(PACKET);` |
| L2 | `FRAME encapsulate(PACKET);` | `PACKET decapsulate(FRAME);` |
| L1 | `BITS encode(FRAME);` | `FRAME decode(BITS);` |

TABLEAU 5.12 – Exemple d'extrait d'API (simplifiée) de simulateur IP.

#### Interception des trames et paquets

Une autre contrainte purement logicielle, concerne les possibilités limitées en terme d'interception des trames ou des paquets. Il est généralement possible d'ajouter une fonction (un *handler*) au modèle des cartes réseau, qui permet d'intercepter les trames ou paquets qui sont reçus d'un autre nœud. Par contre, certains simulateurs comme NS-3 ou OMNeT++/INET, ne fournissent aucune solution pour intercepter les trames ou paquets qui sont émis par la carte réseau elle-même. Les ports placés sur les cartes réseau sont donc capables d'intercepter les paquets en réception, mais pas en émission.

La solution pour contourner ce problème, consiste à utiliser les faux nœuds qui ont été ajoutés au niveau des contraintes de simulation. Ainsi, dans le modèle de la Figure 5.11, les couplages spatiaux (bidirectionnels) se feront désormais entre les nœuds du modèle du milieu, et les faux nœuds des modèles latéraux (plutôt que les nœuds IP A et B). Les trames et paquets émis par les nœuds A et B seront donc interceptés à leur réception par les faux nœuds, plutôt qu'à leur émission. Le transport supplémentaire de ces trames ou paquets sur le faux lien ne changera pas les résultats de simulation, puisque les appendices sont obligatoirement sans effet à ce niveau, d'après la définition fournie dans la Section 5.4.2. On qualifiera ces liens de « liens parfaits ». Le déplacement des ports est illustré dans la Figure 5.13.

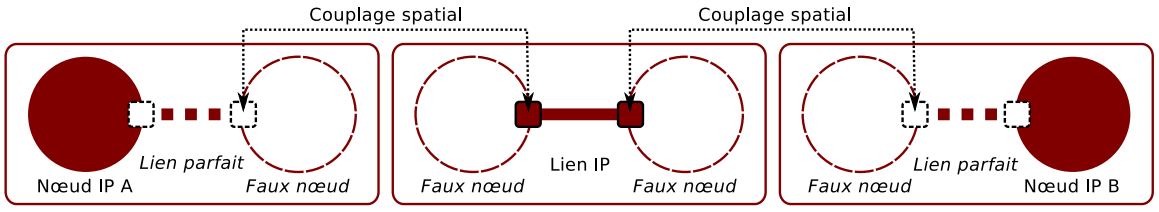



FIGURE 5.13 – Les ports spatiaux sont situés du côté des faux nœuds, pour contourner une contrainte logicielle courante au niveau des API, et les faux liens sont qualifiés de liens parfaits.

Se contenter d'intercepter les trames et paquets en réception plutôt qu'en émission et en réception permet, de façon générale, de réduire l'effort demandé pour réussir à intégrer un nouveau simulateur.

### 5.5.3 Partage d'état des nœuds de transit

Séparer une topologie IP en plusieurs morceaux, en décidant de couper à proximité d'un nœud de transit, peut amener à choisir différentes solutions pour réaliser les couplages.

Le second cas de couplage spatial, décrit dans les contraintes de modélisation à la Section 5.3.2, propose de couper la couche 3 ou la couche 2 en plusieurs morceaux, afin de répartir la représentation des interfaces réseau d'un même nœud de transit sur plusieurs modèles. Au niveau logiciel, ce mode de couplage consiste à créer un même nœud dans plusieurs modèles différents et à faire communiquer leur deux couches 3 (ou leur deux couches 2) ensemble, afin qu'ils s'échangent des informations sur leur état commun, via la plateforme de co-simulation.

Couper à proximité d'un nœud de transit ne nécessite pas toujours de couper au milieu d'une couche, selon le type de coupure souhaité. Nous allons voir quatre situations différentes, qui nécessitent quatre solutions différentes.

### Coupure 1 et 1

La coupure de type « 1 et 1 » suppose qu'on souhaite couper au milieu d'un nœud de transit qui ne possède que deux interfaces. Ce serait par exemple le cas, avec un routeur faisant office de passerelle entre un réseau filaire et un réseau Wifi. La Figure 5.14 illustre ce premier cas. Dans ce cas, la solution est de relier les interfaces réseau des deux représentations du nœud, avec un simple couplage spatial. Il est inutile d'utiliser de faux nœuds, ni des liens parfaits, parce que le nœud de transit (tel que nous l'avons défini dans la Section 5.3.2) n'émettra jamais lui-même de trame ou de paquet à intercepter : tout ce qui doit être transmis d'un côté a obligatoirement été intercepté en réception de l'autre côté du nœud.

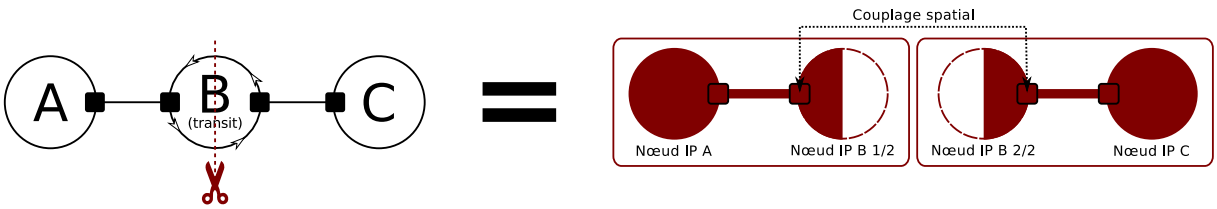



FIGURE 5.14 – Exemple de couplage spatial de type « 1 et 1 ».

### Coupure 1 et $x$

Ce type de coupure suppose qu'on souhaite simplement représenter toutes les interfaces ($> 1$) d'un nœud de transit sur un modèle, sauf une qui sera représentée sur un second modèle. Dans ce cas particulier, il ne faut pas couper au milieu d'une couche, mais simplement couper entre la carte réseau du lien à séparer, et le nœud de transit. Il suffit alors d'utiliser un faux nœud et un lien parfait, comme préconisé par les contraintes de simulation, pour obtenir un couplage similaire à celui illustré dans la Figure 5.15.

### Coupure $x$ et $y$

Ce type de coupure suppose qu'on souhaite couper en deux un nœud de transit, pour répartir $x$ ($> 1$) de ses interfaces sur un modèle, et ses $y$ ($> 1$) autres interfaces sur un second modèle. Ce cas de figure impose que le nœud de transit soit effectivement représenté dans les deux modèles, avec ses $x$ ou $y$ interfaces. Pour faire en sorte que les deux représentations du nœud partagent leur état, il est nécessaire de fictivement représenter (à l'aide d'appendices) les interfaces manquantes

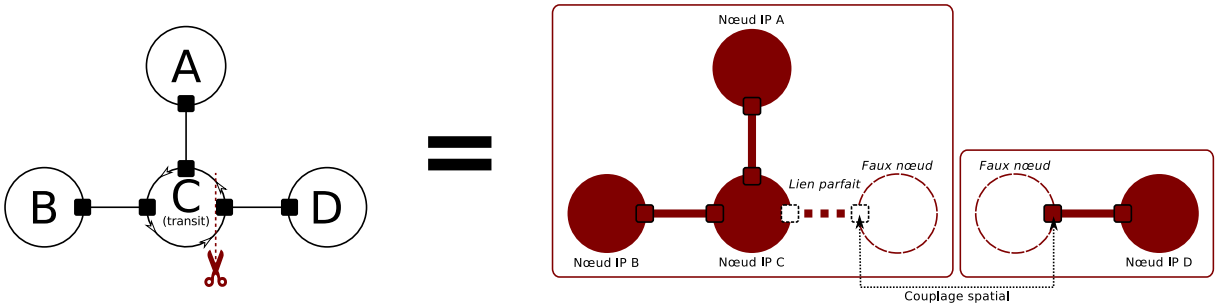



FIGURE 5.15 – Exemple de couplage spatial de type « 1 et $x$ ».

sur les deux modèles, et de relier chaque fausse interface d'un modèle à sa vraie version, située dans l'autre modèle. Cette opération nécessite de réaliser $x + y$ couplages (soit le nombre initial d'interfaces sur le nœud de transit), et est illustrée par un exemple dans la Figure 5.16.

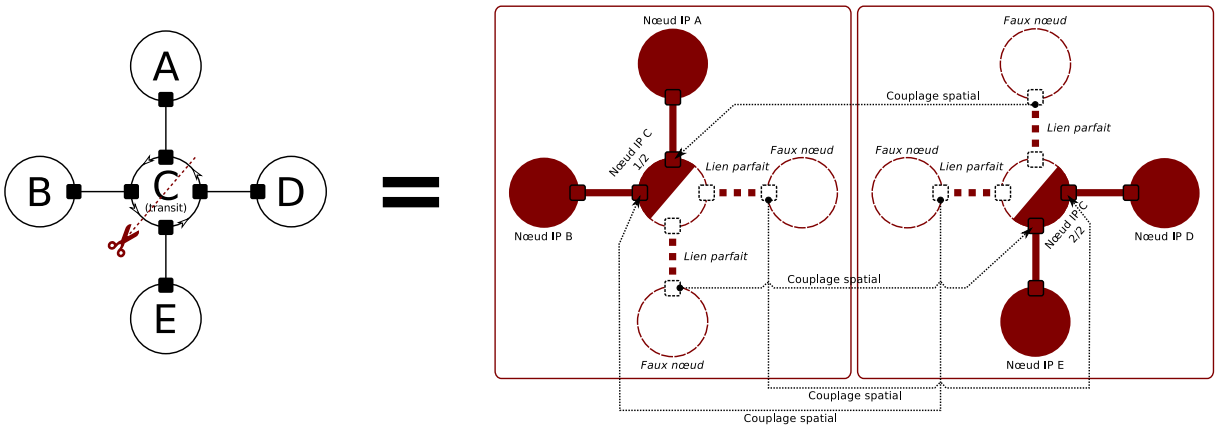



FIGURE 5.16 – Exemple de couplage spatial de type « $x$ et $y$ », sans optimisation.

Une solution pour éviter cette situation qui peut demander beaucoup d'efforts au modélisateur, est d'optimiser le nombre de couplages à réaliser, en acceptant de modifier la topologie IP. On considérera ainsi qu'il n'y a pas un seul nœud de transit à cet endroit du réseau, mais deux nœuds de transit reliés entre eux. Ainsi, la coupure désirée devient de type 1 et $x$, et donc beaucoup plus simple à réaliser. Cette adaptation est illustrée dans la Figure 5.17 (on remarque l'utilisation d'un lien parfait des deux côtés, parce que le lien entre les deux commutateurs n'est jamais supposé exister).

### Coupure en $n$

Ce type de coupure regroupe tous les cas dans lesquels on souhaite couper un nœud de transit en $n$ parties, avec $n > 2$. Ce cas de figure nécessite $n$ représentations différentes du nœud de transit, et donc un partage d'état entre toutes ces représentations. Au niveau des couplages entre interfaces, en considérant qu'il y a en tout $\alpha$ interfaces sur le nœud de transit à représenter, cette situation représente $\alpha * (n - 1)$ couplages spatiaux à mettre en place, de façon similaire au cas précédent (avant optimisation).

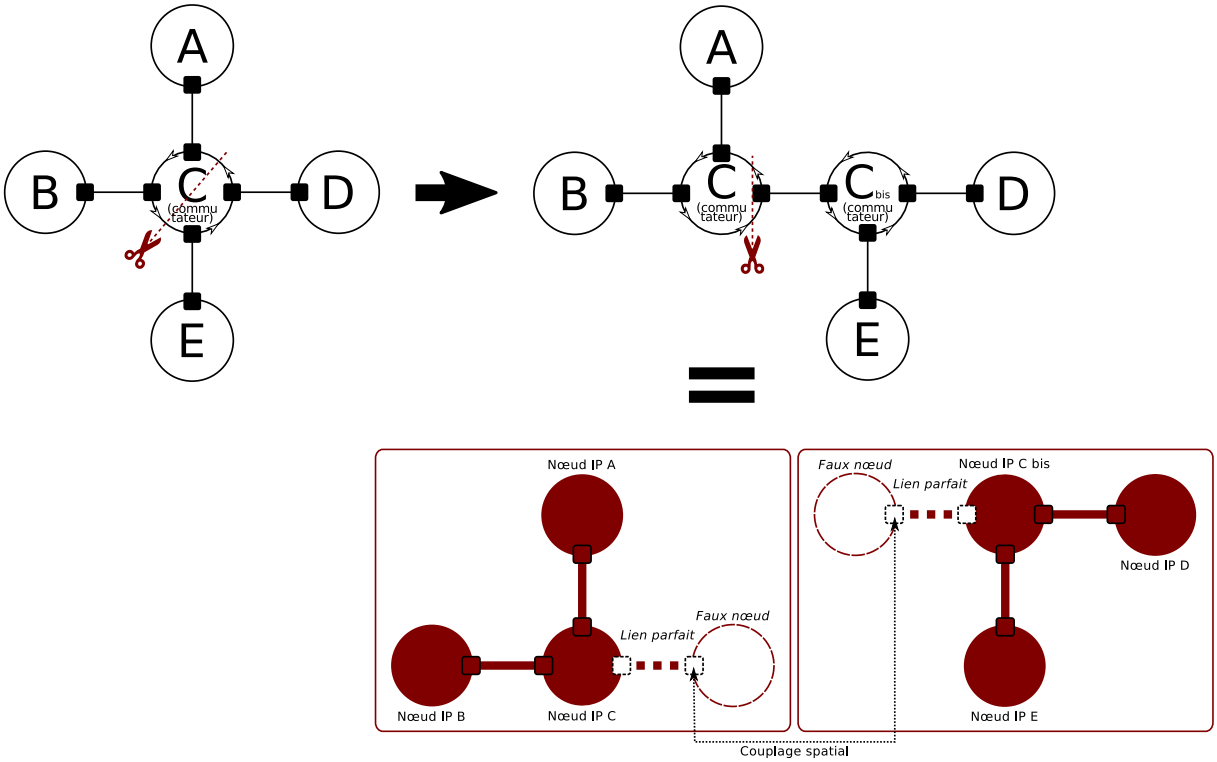



FIGURE 5.17 – Exemple de couplage spatial de type « $x$ et $y$ », pour un nœud de transit de niveau 2 et avec adaptation de la topologie réseau.

De la même façon que dans le cas précédent, il est possible d'améliorer cette situation, en acceptant de modifier la topologie IP. En considérant que chaque groupe d'interfaces, représentées sur un même modèle, est en réalité positionné sur un nœud de transit différent, et que ces nœuds sont reliés par un nœud de transit central (cf. exemple illustré par la Figure 5.18), il suffira de couper au niveau de ce dernier pour réduire le nombre de couplages à réaliser. Il sera ainsi égal à $n * (n-1)$, soit $n^2 - n$. La complexité des couplages reste donc élevée, mais sachant que $\alpha$ est potentiellement très grand dans la mesure où il peut y avoir une centaine de machines reliées à un seul nœud de transit, et que $n$ devrait être très petit dans la mesure où il est peu probable qu'on souhaite couper un nœud de transit en plus de trois ou quatre parties, cette optimisation reste potentiellement très intéressante.

Le cas où on souhaite réaliser une coupure en $n$ pourrait jouir d'une exception dans la méthode de réalisation, si certains modèles n'étaient chargés que de représenter une seule interface. Dans ce cas, on pourrait effectivement utiliser la même solution que pour la coupure de type 1 et $x$, en décidant qu'on sépare simplement cette interface et le lien qui va avec, de l'une des représentations du nœud de transit, dans un des autres modèles. On réduirait ainsi le nombre de représentation du nœud de transit, il y aurait moins de modèles impliqués dans le partage d'état, et il y aurait moins de couplages spatiaux à mettre en œuvre.

Cette exception ne sera valide, que si on considère que la contrainte de simulation concernant le lookahead nul n'existe pas (i.e. l'algorithme de synchronisation du temps de la plateforme de co-simulation supporte les lookahead à zéro – cf. Section 5.4.3). En effet, dans le cas où le modèle qui représente une seule interface transmet une trame qui n'est pas à destination de l'une des

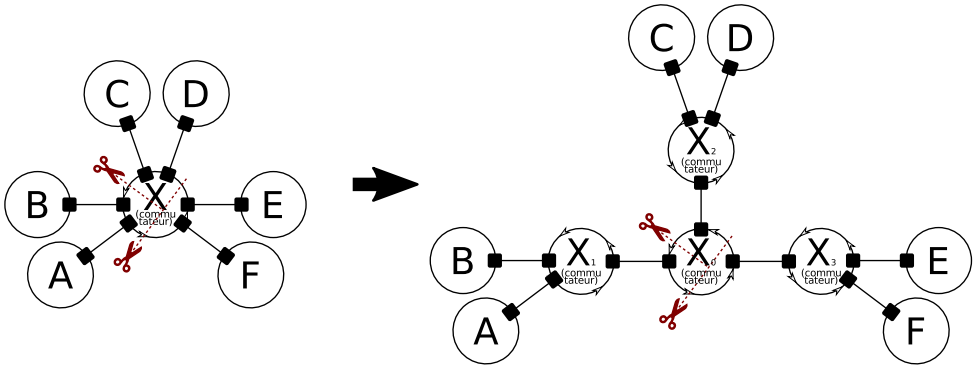



FIGURE 5.18 – Exemple de couplage spatial de type « en $n$ », pour un nœud de transit de niveau 2 et avec adaptation de la topologie réseau.

interfaces représentées dans le modèle avec lequel il est couplé, ce dernier va se contenter de passer la trame d'un port d'entrée à un port de sortie, sans consommer de temps de simulation. Ce qui équivaut à un lookahead nul.

### Optimisations et résultats de simulation

Les adaptations de la topologie IP à modéliser, dans le cadre des optimisations proposées pour les deux derniers types de coupure, impactent les résultats de simulation.

Dans le cas d'un nœud de transit de niveau 2, l'ajout de commutateur entraînera l'émission de quelques trames multicast (e.g. pour le *Spanning Tree Protocol*, IEEE 802.1Q) supplémentaires, et quelques paquets supplémentaires seront transmis en broadcast le temps que toutes les tables d'association entre les interfaces physiques et les adresses MAC soient complétées. Si ces échanges supplémentaires apparaîtront bien dans le total des données échangées par l'ensemble des nœuds au cours de la simulation, le volume infime qu'ils représenteront en comparaison du reste du trafic, devrait assurer leur transparence vis-à-vis des temps de simulation enregistrés.

Dans le cas d'un nœud de transit de niveau 3, ces adaptations sont plus critiques. Si un commutateur peut effectivement se comporter comme un appendice, dans la mesure où il ne modifie pas les données qu'il transporte et qu'il est capable de ne pas consommer de temps de simulation, ça n'est pas le cas pour un routeur. En effet, router un paquet implique d'altérer son entête (e.g. décrémentation du champ *Hop Limit* en IPv6 et du champ *TTL* en IPv4) et de recalculer sa somme de contrôle (en IPv4). Au-delà d'entraîner un impact sur les temps de simulation qui peut devenir significatif, les routeurs supplémentaires peuvent également modifier le comportement du réseau, en étant perçus comme des sauts supplémentaires à traverser dans les routes à calculer. On considérera donc que ces optimisations ne sont pas possibles dans le cas d'un nœud de transit de niveau 3.

Toutefois, il est intéressant de remarquer que les routeurs de type industriel ne possèdent jamais plusieurs interfaces. Il sont reliés à un commutateur, et se contentent de router les paquets entre les réseaux locaux virtuels (VLAN) qui lui sont acheminés. Si le simulateur représente bien les routeurs de cette façon, la séparation des interfaces sur un routeur consiste donc à séparer les interfaces du commutateur associé. Les optimisations proposées sont donc utilisables.

Cependant, un routeur peut également être une simple machine dotée de plusieurs cartes

réseau, qui est capable d'exécuter des protocoles de couche 3. Dans ce cas, les optimisations ne peuvent pas s'appliquer, et il n'y a pas d'autre solution que de réaliser l'intégralité des couplages spatiaux entre les interfaces. On peut cependant remarquer qu'il est rare qu'un même routeur serve à relier directement plus de trois ou quatre réseaux différents ensemble.

La problématique liée au partage de la table de routage entre les différentes représentations d'un même routeur, sera abordée dans la Section 6.4.

## 5.6 Restriction des modes de couplage des modèles IP

### 5.6.1 Motivations

Nous avons vu au début de ce chapitre, un exemple de multi-modèle représentant un réseau IP minimaliste (cf. Figure 5.1), pour lequel un modèle IP différent représentait chaque couche de la suite Internet. Cet exemple offrait un panorama relativement complet des différentes possibilités qui pourraient exister en termes de couplage de modèles IP.

L'étude des contraintes de modélisation, de simulation et logicielles, a permis de mettre en avant des contraintes particulièrement fortes pour certaines de ces possibilités. L'intérêt de ces possibilités étant très limité par rapport à la difficulté de les prendre en considération, et afin de simplifier les solutions qui seront proposées dans la suite du manuscrit, nous choisissons ici de restreindre la définition des couplages structurels et spatiaux, qui sont applicables aux modèles IP.

### 5.6.2 Couplages structurels

Nous avons vu que, contrairement à ce qui serait attendu, le modèle en couches de la suite Internet ne permet pas de séparer aisément chacune des couches dans des modèles isolés. Que ça soit au niveau modélisation ou au niveau logiciel, les couches sont fortement dépendantes entre elles. Dès lors, coupler structurellement ensemble des modèles IP représentant des couches adjacentes, nécessite que les modèles soient compatibles entre eux et qu'ils échangent de nombreuses informations. Nous avons également vu que la contrainte de simulation liée au lookahead nul posait problème, dans le cas où un modèle se contentait de représenter une couche, en pouvant parfois avoir pour seul rôle d'encapsuler et de désencapsuler une donnée, sans consommer de temps de simulation.

De plus, toutes les couches n'ont pas forcément d'intérêt à être séparées. Par exemple, séparer la couche 4 de la couche 3 ne servira pas à grand chose : un simulateur qui sait modéliser de l'IP à la couche 3, saura probablement aussi simuler les protocoles UDP et TCP à la couche 4. Puisque UDP et TCP sont quasiment les seuls protocoles de transport utilisés sur les réseaux IP actuels, ils suffiront quasiment dans tous les cas.

Par contre, séparer la couche 3 de la couche 2 peut avoir un intérêt : un simulateur IP qui sait modéliser la couche 2 du Bluetooth ne sait pas obligatoirement faire de l'IPv6 au niveau 3. Cependant, nous avons vu avec l'exemple de la Figure 5.5 (premier exemple de couplage spatial, page 68), que la séparation entre la couche 2 et la couche 3 pouvait aussi se faire en spatial.

Puisque, d'après les contraintes logicielles, les couches 1 et 2 sont généralement confondues, on peut finalement constater que le couplage structurel n'a réellement de sens qu'entre la couche 5

(applicative) et les autres couches (couches IP). C'est aussi la seule paire de couches qui n'a pas d'interdépendances (d'après la Figure 5.7).

Pour toutes ces raisons, et parce que nos travaux ont pour objectif d'être utilisés dans un cadre industriel, et non de recherche expérimentale (e.g. pas de conception de nouveaux protocoles de couche 4), nous choisissons de **restreindre les couplages structurels à la séparation entre le point de vue « réseau », et le point de vue « applicatif »** des équipements.

### 5.6.3 Couplages spatiaux

Nous avons vu que les couplages spatiaux pouvaient principalement intervenir à deux endroits différents, dans la topologie d'un réseau :

1. au niveau des cartes réseau ;
2. au niveau des nœuds de transit.

On peut dès lors distinguer deux types de couplage spatial :

— **De niveau 3 :** entre la couche 3 et la couche 2 (c'est-à-dire entre un nœud et sa carte réseau), ou au milieu d'un nœud de transit de niveau 3 (i.e. un routeur). Le type de donnée qui sera échangé entre les deux modèles sera donc le paquet IP.
— **De niveau 2 :** entre la couche 2 et la couche 1 (c'est-à-dire entre une carte réseau et son lien), ou au milieu d'un nœud de transit de niveau 2 (i.e. un commutateur). Le type de donnée qui sera échangé entre les deux modèles sera donc la trame.

Puisqu'on sait que les implémentations des couches 2 et 1 sont souvent confondues, le couplage de niveau 2 entre une carte réseau et son lien, sera principalement utile lorsque le modèle de lien devra être simulé de façon très précise, par exemple par un modèle équationnel plutôt que par un modèle IP.

La mise en place des différents types de couplage dépend principalement de la capacité du modélisateur à mettre en œuvre la notion de port. Sur la base de ces trois types de couplage, et à partir des différents concepts introduits au niveau des contraintes de simulation, les notions et outils qui seront nécessaires pour les concevoir et les implémenter, sont synthétisés dans la section suivante.

## 5.7 Synthèse et représentation

### 5.7.1 Introduction

Cette section présente une synthèse des solutions et des concepts permettant de concevoir un multi-modèle intégrant des modèles IP, qui ont été proposées dans ce chapitre. Elles propose également une représentation graphique pour ces concepts, à intégrer à la description d'un multi-modèle.

La représentation graphique des éléments de base d'un réseau est proposée dans la Figure 5.19 (les liens sans fil sans distingués des liens filaires uniquement pour agrémenter la lecture des schémas).

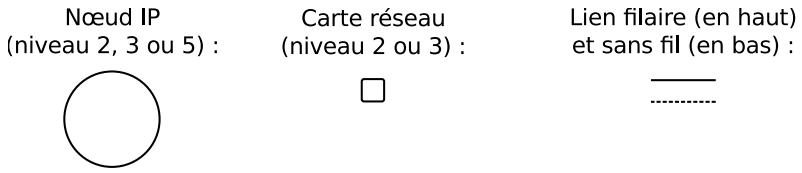



FIGURE 5.19 – Représentation graphique des éléments de base d'un réseau IP.

### 5.7.2 Trois modes de couplage

On distingue trois modes de couplage, définis dans les Sections 5.2.2 et 5.6 :

— structurel (transmission de données applicatives au niveau 5) ;
— spatial de niveau 2 (transmission de trames simulées) ;
— spatial de niveau 3 (transmission de paquets simulés).

Ces couplages interviennent entre un modèle IP et un modèle issu d'un autre domaine d'expertise, dans le cas des couplages structurels, et entre modèles IP (qui n'utilisent pas nécessairement le même formalisme) dans le cas des couplages spatiaux. Les couplages transmettent des données de façon bidirectionnelle ou directionnelle, selon le rôle (d'entrée, de sortie ou unidirectionnel) de la paire de ports auxquels ils sont reliés. Une représentation graphique des modes de couplage entre modèles est proposée dans la Figure 5.20 (les pointes indiquent les directions possibles pour les échanges, à l'instar de flèches).

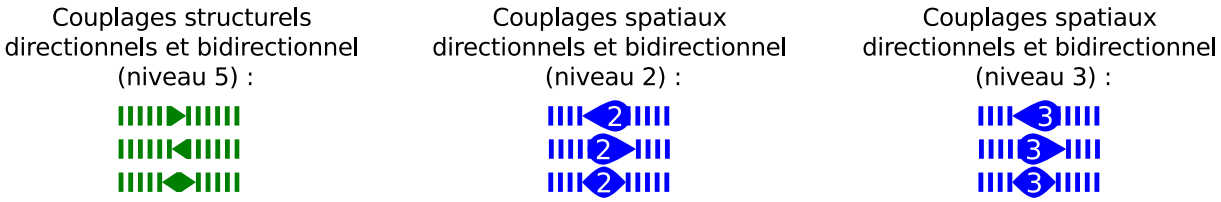



FIGURE 5.20 – Représentation graphique des différents types de couplage impliquant des modèles IP.

Les ports structurels sont situés directement au niveau des nœuds en tant qu'applications, et les ports spatiaux sont situés au niveau des cartes réseau. Une représentation graphique de ceux-ci est proposée dans la Figure 5.21. Ils peuvent être d'entrée, de sortie, ou unidirectionnels, selon le type d'échanges qu'ils sont amenés à effectuer avec les autres modèles.

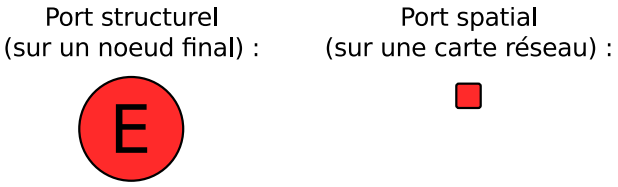



FIGURE 5.21 – Représentation graphique des différents types de port pour les modèles IP.

### 5.7.3 Solutions pour découper une topologie IP

L'étude des contraintes nous a permis d'identifier les situations les plus courantes, auxquelles se confronte un modélisateur lorsqu'il souhaite spatialement découper son réseau IP. Nous avons caractérisé les différents types de coupure qui sont envisageables, et nous avons indiqué quelle était la bonne solution pour concevoir les modèles résultants (i.e. quelle topologie IP représenter dans chacun de ces modèles), de façon à respecter l'ensemble des contraintes.

Le premier critère à prendre en considération pour identifier le type de coupure, et de vérifier si on souhaite couper :

1. autour d'une carte réseau (la séparant de son nœud ou de son lien) ;
2. ou au milieu d'un nœud de transit, de façon à séparer des interfaces.

Un nœud de transit peut être de niveau 2 (e.g. commutateur) ou de niveau 3 (e.g. routeur), est équipé de plusieurs interfaces, et se contente de faire transiter des trames ou des paquets de l'une à l'autre. Un nœud final, à l'inverse, est un nœud qui a un comportement applicatif (niveau 5) et qui peut produire ses propres données. Une représentation graphique de ces différents types de nœud est proposée dans la Figure 5.22.

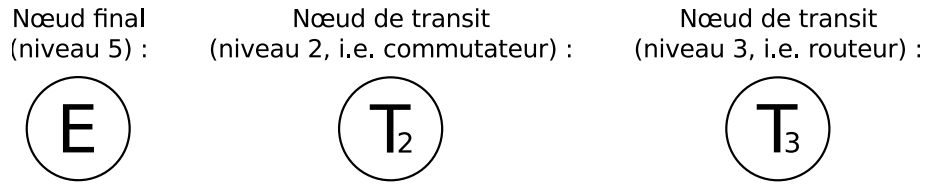



FIGURE 5.22 – Représentation graphique des nœuds finaux et de transit.

Si la coupure intervient au milieu d'un nœud de transit, celui-ci nécessite un partage d'état entre ses représentations. Nous avons identifié quatre situations différentes, qui mènent à quatre solutions différentes :

1. coupure 1 et 1 (le nœud n'a que deux interfaces, qui sont à séparer l'une de l'autre) ;
2. coupure 1 et $x$ (une seule interface est à isoler des autres) ;
3. coupure $x$ et $y$ ($x$ interfaces sont à séparer des $y$ autres) ;
4. coupure en $n$ (le nœud est coupé plusieurs fois de façon à créer plus de deux groupes d'interfaces).

Les solutions à appliquer pour ces différentes situations sont présentées dans la Section 5.5.3.

### 5.7.4 Collection d'appendices

Les différentes solutions proposées lorsqu'on découpe une topologie IP, s'appuient sur l'utilisation d'appendices définis dans les Sections 5.4.2 et 5.5.2, pour respecter les contraintes de simulation et parfois pour placer les ports spatiaux.

Les appendices sont des éléments réseau qu'on modélise, alors qu'ils n'existent pas dans le système (ils sont ainsi qualifiés de « faux »). Ils permettent simplement de lever les contraintes de

simulation, en complétant les réseau, et parfois d'accueillir les ports spatiaux. Selon la définition donnée dans la Section 5.4.2, ces éléments ne doivent avoir aucun impact sur le réseau qui se traduise par une modification des résultats de simulation finaux.

Les deux principaux appendices qui sont utilisés dans les solutions proposées, sont les faux nœuds et les liens parfaits. Les cartes réseau peuvent également être fausses, et accueillir un port spatial. Ces différents appendices sont représentés graphiquement dans la Figure 5.23.

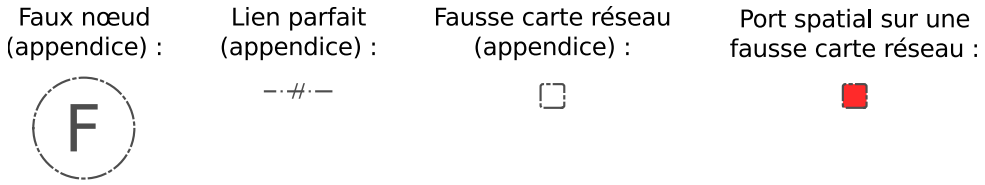



FIGURE 5.23 – Représentation graphique des différents appendices utilisables.

### 5.7.5 Représentation des nœuds de transit avec état partagé

Dans le cas des coupures de type « $x$ et $y$ » et « en $n$ », nous avons proposé dans la Section 5.5.3 des solutions optimisées, afin de limiter le nombre de couplages à mettre en place entre les modèles. Ces couplages permettent de mettre en place un partage d'état entre plusieurs représentations d'un nœud de transit, qui accueillent chacun une partie des interfaces du nœud correspondant dans le système.

Lorsque le nombre de couplages reste important, il n'est pas possible de les représenter graphiquement, en conservant un schéma humainement compréhensible. Pour cette raison, on ajoute les deux représentations de port avec état partagé, visibles sur la Figure 5.24. Tous les nœuds qui possèdent ce port, et qui portent le même nom sur le schéma, ont leurs interfaces couplées au niveau indiqué et de la façon décrite à la Section 5.5.3, sans que les couplages ne soient représentés.

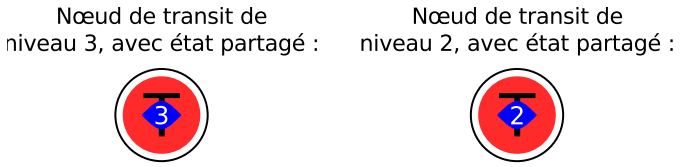



FIGURE 5.24 – Représentation graphique possible des nœuds de transit avec état partagé.

### 5.7.6 Exemple de multi-modèle

Un exemple complet de multi-modèle, illustrant un certain nombre des concepts abordés ici, est disponible dans la Figure 5.25. L'utilisation des différents concepts dans l'exemple, est justifiée par les types de coupure de la topologie, tels qu'ils sont illustrés dans la Figure 5.26.

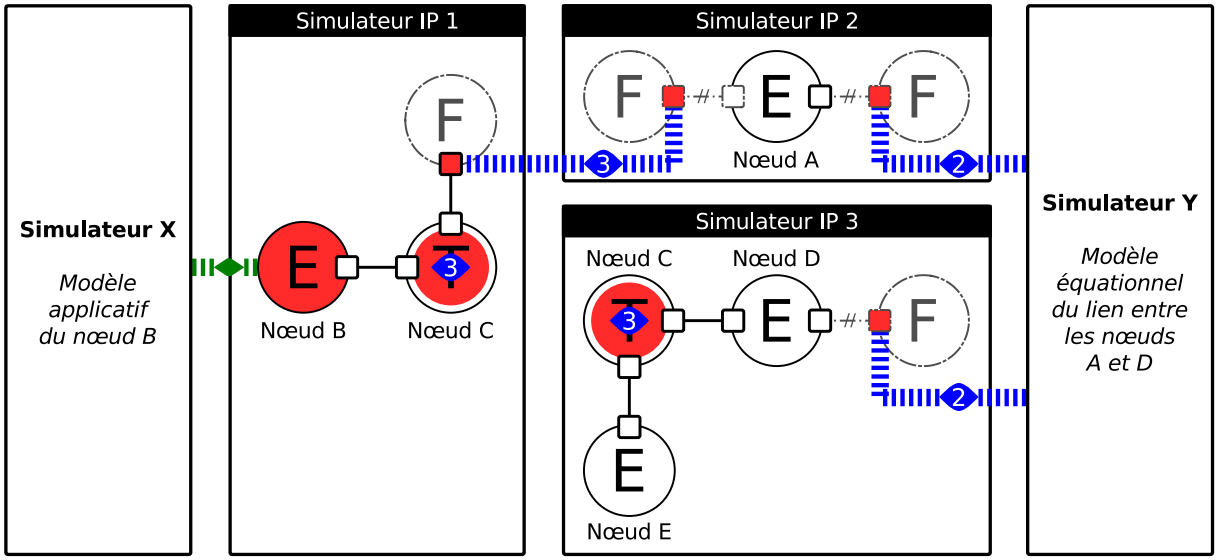



FIGURE 5.25 – Multi-modèle correspondant à l'exemple de la Figure 5.26, illustrant différentes représentations graphiques qui seront utilisées dans la suite de ce manuscrit.

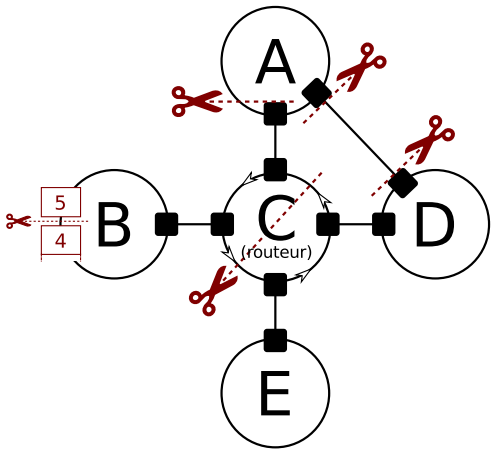



FIGURE 5.26 – Exemple de topologie de réseau IP à représenter à l'aide de modèles interconnectés, selon un certain nombre de découpes précises. Les nœuds qui ne sont pas entourés de flèches sont implicitement des nœuds finaux.

## 5.8 Conclusion

Ce chapitre a permis de comprendre que les modèles IP d'un multi-modèle étaient créés en fonction de :

1. un schéma réseau de la topologie IP à représenter ;
2. du besoin ou non de répartir la représentation de la topologie IP sur plusieurs modèles ;
3. d'un certain nombre de contraintes, du niveau modélisation au niveau logiciel, en passant par la simulation.

L'étude de ces contraintes, nous a permis de proposer des solutions permettant de :

1. spécifier la sémantique des couplages des modèles IP, en proposant trois modes de couplage clairement définis ;
2. produire un multi-modèle IP, avec la topologie IP à représenter dans les modèles IP individuels ainsi que leurs couplages, à partir du schéma d'une topologie IP à modéliser en plusieurs parties ;
3. définir et positionner des ports au niveau IP dans les modèles, permettant de faire communiquer les réseaux simulés avec d'autres éléments simulés dans des modèles externes.

Un certain nombre de concepts (modes de couplage, types de port et appendices) ont été définis, pour permettre de mettre en œuvre ces solutions. Des représentations graphiques pour ces concepts permettent de décrire l'intégration des modèles IP dans le multi-modèle, en mélangeant la représentation IP du système modélisé, avec les couplages du multi-modèle.

Le chapitre suivant donne le détail du fonctionnement attendu pour les ports structurels et spatiaux et les solutions pour concevoir les appendices, permettant de passer du niveau conceptuel défini dans ce chapitre, au niveau du code. Le chapitre suivant propose également une relation entre les ports structurel et spatiaux, et la notion de port DEVS, permettant ainsi d'intégrer les modèles à des multi-modèles DEVS. Enfin, il décrit comment la dynamique des simulateurs IP doit être modifiée, pour pouvoir être wrappés en DEVS et ainsi rejoindre une co-simulation DEVS.

# Chapitre 6

# Intégration de modèles IP existants à un multi-modèle hybride

**Sommaire**



## 6.1 Introduction

Le Chapitre 5 a permis de distinguer deux principaux modes de couplage, entre modèles IP (spatial) et entre modèles IP et modèles issus d'autres domaines d'expertise (structurel). Il a également permis d'introduire un certain nombre d'outils, qui doivent être à la disposition du modélisateur, pour concevoir les modèles IP à intégrer au multi-modèle.

Dans ce chapitre, nous verrons concrètement à quoi peuvent correspondre ces outils dans le modèle, en nous limitant à des notions propres au domaine IP, afin de pouvoir les implémenter en utilisant directement la bibliothèque IP du simulateur. Nous verrons notamment quel comportement est attendu pour les ports structurels et spatiaux, et de quelle façon ils peuvent faire le lien entre le réseau IP simulé et la plateforme de co-simulation. Puisque nous avons choisi d'utiliser une plateforme de co-simulation basée sur DEVS, nous verrons dans quelle mesure la notion de port DEVS peut être intégrée dans celle de port spatial ou structurel. L'ajout de ports DEVS aux modèles IP permettra de les ouvrir, en leur offrant la possibilité de recevoir et d'envoyer des données depuis et vers le monde extérieur.

Dès lors que nous saurons comment intégrer des ports DEVS aux modèles IP, nous nous intéressons au simulateur IP. Ce dernier doit en effet désormais intégrer la possibilité de recevoir des événements de l'extérieur et d'avoir à en transmettre, pour l'ensemble des couplages structurels et spatiaux du modèle qu'il exécute. Nous verrons ainsi comment modifier la dynamique du simulateur IP, afin qu'il avance son temps de simulation en fonction des autres simulateurs de la co-simulation, et qu'il dialogue avec ses ports pour effectuer des échanges de données. Nous nous appuyerons pour cela sur la notion de wrappeur DEVS, en utilisant l'implémentation proposée par la plateforme MECSYCO.

## 6.2 Ouverture des modèles : ajout de ports

### 6.2.1 Introduction

La section précédente a permis d'identifier les différents outils nécessaires pour concevoir des multi-modèles qui intègrent des modèles IP.

L'objectif de cette section est de comprendre comment ces outils peuvent être intégrés dans des modèles déjà existants, en utilisant les possibilités qui sont offertes par les simulateurs IP, et sans directement modifier le code de ces derniers (en accord avec les contraintes fixées dans la Section 4.2). L'ajout de ports structurels et spatiaux permettra « d'ouvrir » les modèles IP, de façon à ce qu'ils soient capables de communiquer avec l'extérieur, via une plateforme de co-simulation. En accord avec notre proposition (cf. Section 4.4), nous utiliserons une plateforme de co-simulation DEVS, ce qui implique que les ports structurels et spatiaux soient reliés à la notion de port DEVS.

Pour rappel, les ports doivent avoir un comportement différent, selon s'ils sont structurels ou spatiaux (cf. Section 5) :

**Dans le cadre d'un couplage structurel** entre deux modèles d'une même co-simulation, chacun des modèles représente un même système, au même instant, mais selon des points de vue différents. Ainsi, un même appareil pourra être représenté en sa qualité d'équipement de réseau IP dans un modèle IP, alors qu'il sera représenté d'un point de vue applicatif (i.e. en tant que générateur ou consommateur de données applicatives) par un modèle issu d'un autre domaine d'expertise (e.g. modèle physique à base d'équations différentielles).

**Dans le cadre d'un couplage spatial** entre deux modèles d'une même co-simulation, chacun des modèles représente une partie distincte du système. Ainsi, une partie de la topologie de réseau IP sera représentée dans un premier modèle IP (e.g. partie sans-fil d'un réseau IP d'opérateur mobile), tandis que l'autre partie sera représentée dans un

second modèle IP (e.g. partie filaire du réseau IP de ce même opérateur mobile). Les deux modèles IP, éventuellement exécutés par deux simulateurs IP différents, s'échangent des données de simulation de la même façon que les deux parties du réseau IP s'échangeraient des informations dans le système.

Les concepts et le formalisme graphique de la plateforme MECSYCO (cf. Section 2.5.5) sont utilisés pour expliquer les liaisons avec le reste du multi-modèle.

### 6.2.2 Approche proposée

L'approche que nous proposons est de matérialiser les ports DEVS et les appendices dans les modèles IP, en les définissant en termes de fonctionnalités IP.

Cette solution nous permet notamment d'utiliser directement les sous-modèles qui sont proposés dans les bibliothèques des simulateurs IP, pour créer les objets correspondant aux ports DEVS à définir. Puisque, dans le cas des couplages structurels, les données de simulation à échanger entre les modèles sont de type applicatif, nous choisissons de définir les ports au niveau des applications des nœuds (i.e. les ordinateurs) du réseau IP simulé. Ajouter la notion de port DEVS structurel à un simulateur IP consiste donc à utiliser ses fonctionnalités pour créer un nouveau type de sous-modèle d'application, et l'ajouter à sa bibliothèque de sous-modèles IP. Par conséquent, positionner un port DEVS structurel d'entrée ou de sortie dans un modèle IP existant consiste à « installer » une application modélisée spécifique, sur un nœud simulé en particulier.

Puisque dans le cas des couplages spatiaux, les données de simulation à échanger entre les modèles correspondent à des trames ou des paquets simulés, nous choisissons de définir les ports au niveau des cartes réseau simulées. Ainsi, ajouter la notion de port DEVS spatial à un simulateur IP consiste à utiliser ses fonctionnalités pour créer un nouveau type de sous-modèle de carte réseau, et l'ajouter à sa bibliothèque de sous-modèles IP. Par conséquent, positionner un port DEVS spatial d'entrée ou de sortie dans un modèle IP existant consiste à « équiper » un nœud simulé en particulier, d'une carte réseau modélisée spécifique. De la même façon, les appendices introduits dans le Chapitre 5 pourront être représentés uniquement en utilisant des sous-modèles IP déjà existants dans les bibliothèques IP, en les paramétrant pour qu'ils adoptent le comportement qui est attendu.

Les objets correspondants aux ports DEVS dans les modèles doivent être capables de proposer quelques primitives, permettant d'échanger des informations avec le wrappeur DEVS. Ainsi, les objets doivent notamment être capables de :

1. notifier au wrappeur DEVS lorsqu'ils sont prêts à être utilisés,
2. recevoir une donnée et la transmettre de la façon appropriée au réseau IP simulé, et
3. indiquer s'ils ont reçu des données du réseau simulé à leur transmettre, et les leur transmettre le cas échéant.

Les deux sections suivantes détaillent cette approche, en spécifiant le comportement opérationnel attendu pour les ports structurels, puis spatiaux.

### 6.2.3 Fonctionnement des ports structurels

#### Exemple de couplage structurel

Afin d'illustrer cette section, un exemple de couplage structurel simple est proposé en Figure 6.1.

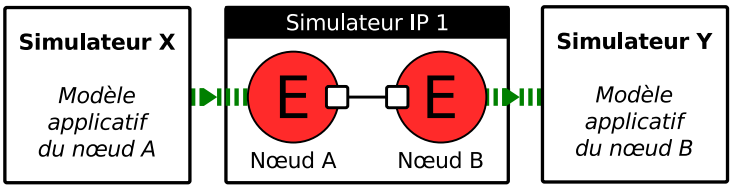



FIGURE 6.1 – Exemple simple de couplage structurel, avec un nœud A qui envoie des données (générées par un modèle applicatif externe) à un nœud B (qui traite ces données via un autre modèle applicatif externe).

Dans le système lié à cet exemple, un équipement A génère des données (e.g. des températures), qu'il doit régulièrement transmettre à un équipement B, qui est distant. Les deux équipements communiquent via un réseau IP. Les comportements de A et de B sont modélisés grâce aux simulateurs X et Y, mais on souhaite également simuler le réseau qui leur permet de communiquer (e.g. pour ajouter des délais, pour tester différents scénarios de données altérées ou qui n'arrivent pas, etc). Un simulateur IP est donc utilisé pour modéliser le réseau IP qui sépare A et B (qui est réduit ici à un simple câble entre deux machines). Dans ce réseau, les équipements A et B sont représentés, en tant qu'équipements réseau. Les couplages structurels permettent de relier leur représentation d'un point de vue réseau, à leur représentation d'un point de vue applicatif ou fonctionnel, en assurant le pont entre les modèles.

#### Représentation des ports structurels dans le modèle IP

Pour ajouter la notion de port structurel aux modèles IP, il est nécessaire de comprendre le comportement attendu de ces ports, vis-à-vis du réseau IP. Nous avons vu dans la Section 5.5.1 qu'une application simulée sur un nœud était logiquement l'endroit le plus approprié pour échanger des données applicatives.

Admettons que les nœuds de l'exemple de la Figure 6.1 échangent des données en TCP (niveau 4). Pour que les deux nœuds puissent communiquer ensemble, il faut donc qu'un nœud agisse en serveur TCP, et l'autre en client TCP. Les fonctions de client et de serveur TCP se déterminent au niveau applicatif, selon si l'application sur le nœud utilise une socket TCP de type client ou serveur. Puisque, dans l'exemple, c'est uniquement A qui envoie des données à B, le nœud client est logiquement A, et le nœud serveur est B.

Ce sont les applications simulées qui créent ces sockets, qui sont les plus à même de correspondre aux ports DEVS des couplages structurels. Un port d'entrée est donc une application cliente, et un port de sortie une application serveur. Un port structurel d'entrée peut être constitué de plusieurs ports DEVS d'entrées, un port structurel de sortie de plusieurs ports DEVS de sortie, et un port structurel bidirectionnel de plusieurs ports DEVS des deux catégories.

Le protocole TCP a été utilisé dans cet exemple, mais la communication aurait aussi pu

avoir lieu en UDP dans le système. Ce cas de figure aurait par conséquent nécessité l'utilisation de sockets UDP au niveau des ports. Puisque TCP et UDP sont incontestablement les deux protocoles de transport les plus incontournables (et quasiment les seuls utilisés), nous supposerons que ce sont les seuls intéressants à considérer, dans le cadre des couplages structurels.

### Interactions entre les ports structurels et la plateforme de co-simulation

Les ports DEVS permettent de faire la passerelle entre le modèle et le reste du multi-modèle. Les applications correspondant à ces ports doivent donc être capables de gérer les échanges de données applicatives, entre le réseau simulé et la plateforme de co-simulation.

Un exemple du comportement attendu pour les ports DEVS des ports structurels est donné avec le diagramme de séquence de la Figure 6.2 (en supposant un envoi en TCP d'un nœud A à un nœud B, comme dans l'exemple de la Figure 6.1). On constate que le port a besoin de deux temps pour fonctionner : d'abord il participe à l'établissement d'une connexion dans une phase d'initialisation (ouverture des sockets), et ensuite il procède à la transmission des données lorsqu'on le lui demande (ou bien traite leur réception, selon son rôle).

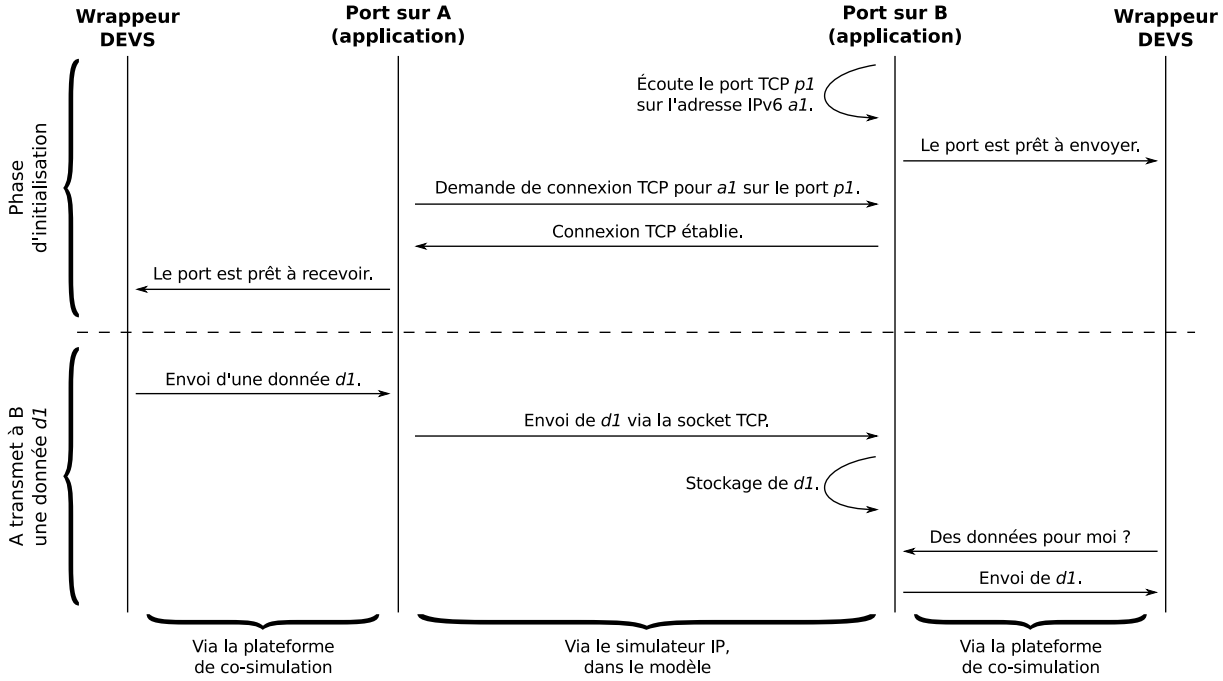



FIGURE 6.2 – Diagramme de séquence correspondant au fonctionnement des ports structurels, avec l'exemple d'une donnée applicative *d1* envoyée de A (port de sortie) à B (port d'entrée) en TCP. L'un des deux ports aurait pu être une simple application. Correspond à l'exemple de la Figure 6.1.

La phase d'initialisation a logiquement lieu au début de la simulation, lorsque le simulateur initialise le modèle IP avec ses différents composants. La nécessité d'avoir à attendre que la socket s'initialise, pour que le port DEVS soit opérationnel, peut engendrer deux effets de bord :

1. si tous les ports s'initialisent en même temps au début de la simulation, tous les paquets de connexion échangés peuvent entraîner une saturation des liens simulés et donc des

retards dans l'établissement des connexions, voire des sockets dont l'ouverture échoue à cause des paquets perdus (ce premier point concerne plus particulièrement les sockets TCP) ;

2. si les sockets associées aux ports ne sont pas immédiatement prêtes dans le modèle, il y a un risque que la plateforme de co-simulation demande la transmission de données applicatives sur le réseau simulé, avant que les ports ne soient opérationnels.

Pour résoudre ces problèmes, l'implémentation des ports structurels doit intégrer la notion de *jitter*, en permettant de définir un laps de temps durant lequel les connexions pourront se répartir de façon aléatoire, dans la phase d'initialisation. Pour que ce laps de temps soit à l'abri des envois de données trop précoces de la part des autres simulateurs, une solution consiste à décaler le temps du simulateur IP de quelques secondes, par rapport à ceux avec lesquels il est structurellement couplé. Avec la plateforme MECSYCO, ce type de décalage peut facilement être mis en place, en utilisant une opération de transformation du temps.

Dans le diagramme de la Figure 6.2, les deux nœuds sont équipés d'un port structurel, mais cette condition n'est pas obligatoire : selon les cas, le rôle d'application cliente ou serveur aurait aussi pu être tenu par une « vraie » application simulée, proposée dans la bibliothèque de modèles IP du simulateur (e.g. un serveur web).

Avec la plateforme de co-simulation MECSYCO, chaque port DEVS d'un modèle entraîne la création d'un artéfact de couplage au niveau du multi-modèle (cf. Section 2.5.5 pour les notions relatives à MECSYCO). Les couplages structurels correspondent donc au transfert de données applicatives via ces artéfacts. Grâce à notre connaissance du fonctionnement des ports structurels, il est donc désormais possible de faire le lien entre la représentation du réseau IP simulé, et les artéfacts de couplage de la plateforme de co-simulation. C'est l'objet de la Figure 6.3, qui schématise le cas d'un port structurel composé de trois ports DEVS d'entrée et de trois ports DEVS sortie.

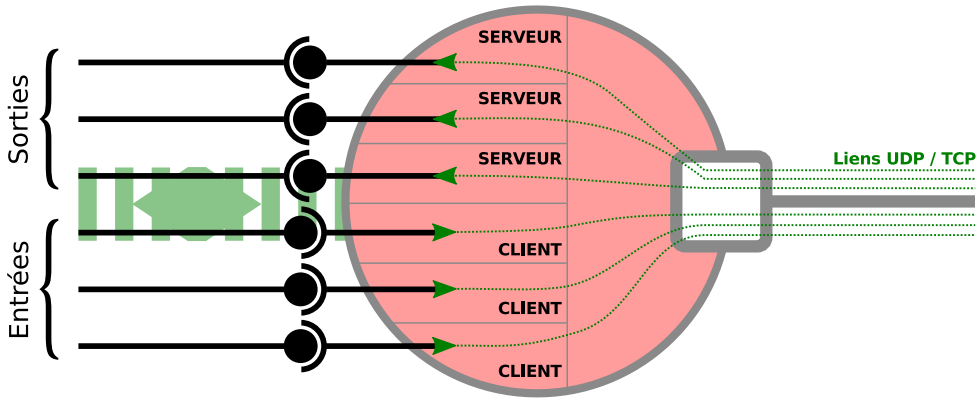



FIGURE 6.3 – Représentation schématique des connexions entre un port structurel composé de plusieurs ports DEVS, et la plateforme de co-simulation.

#### Limitations

Un port DEVS ne pouvant pas être bidirectionnel, on constate que dans notre proposition, une même connexion UDP/TCP [7] n'est jamais utilisée pour permettre à deux nœuds de communiquer dans les deux sens. Cette limitation peut poser problème, dans la mesure où les communications bidirectionnelles via une seule et unique connexion UDP/TCP ne sont pas rares dans la réalité. Ainsi, si un modèle externe devait représenter le comportement d'un humain qui discute en ligne, le port structurel bidirectionnel qui serait la représentation en nœud IP de son ordinateur, devrait disposer de deux ports DEVS (un d'entrée et un de sortie), donnant lieu à deux connexions TCP avec le nœud représentant le serveur de discussions. Cette représentation des communications entre les deux machines peut ne pas être conforme au système, qui n'utiliserait peut-être qu'une seule liaison TCP bidirectionnelle.

Lever cette limitation nécessiterait d'ajouter une couche d'abstraction entre les notions de client/serveur et de port DEVS. Chaque client ou serveur pourrait alors proposer distinctement une fonction d'écriture et de lecture de la socket, qui correspondraient respectivement à des ports DEVS d'entrée et de sortie. Il n'y aurait dès lors plus aucune relation entre le sens de la communication et le rôle de client ou de serveur, sur le nœud. Par conséquent, le modélisateur devrait préciser lui-même quel nœud est supposé être le serveur, et quel nœud le client, pour chacune des communications, afin de respecter le fonctionnement observé sur le système.

Cette contrainte permet d'illustrer un ensemble plus général de limitations. La programmation réseau (dont fait partie la manipulation de sockets) est à la frontière entre l'application et le réseau. Dans notre proposition, elle a été située du côté du réseau, précisément pour lui permettre de faire le lien avec l'application. Pourtant, la programmation réseau est parfois directement utilisée dans les fonctions de l'application : par exemple, pour interdire l'accès au serveur à certains utilisateurs (identifiés par leurs adresses IP) ou pour imposer un nombre maximum de connexions. De la même façon, dans le système, ces connexions TCP peuvent s'ouvrir et se fermer au cours du temps, selon des critères qui sont à la discrétion des applications. On pourrait encore citer le choix d'organisation des sessions des éventuels serveurs UDP multi-clients, qui varie d'une application à l'autre ; et tant d'autres cas particuliers.

Répondre à toutes ces contraintes nécessiterait de permettre de décrire le comportement de chaque application vis-à-vis du réseau, pour chaque port DEVS installé, pour chaque port structurel du modèle. Afin d'être capable de garantir une conception aisée de multi-modèles intégrant des modèles IP, nous choisissons d'ignorer ces contraintes. Nous estimons par ailleurs que, dans la majorité des cas, le niveau de précision atteint par la simulation, y compris lorsque les limitations n'ont pas permis de représenter parfaitement le système, sera suffisant pour répondre aux questions qui sont posées.

### 6.2.4 Fonctionnement des ports spatiaux

#### Exemple de couplage spatial

Afin d'illustrer cette section, un exemple de couplage spatial simple est proposé en Figure 6.1.

Dans le système lié à cet exemple, deux nœuds A et B sont reliés par un simple lien, en pair-à-pair. Les sous-modèles permettant de modéliser A et le lien sont disponibles uniquement dans

7. Bien que UDP ne soit pas un protocole orienté connexion, on appelle une connexion UDP un quartet du type *IP-SRC, PORT-SRC, IP-DST, PORT-DST*

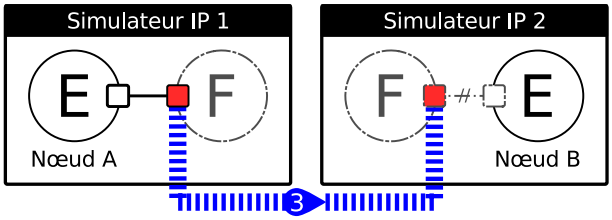



FIGURE 6.4 – Exemple simple de couplage spatial, avec un nœud A et un lien représentés dans un premier modèle, et un nœud B représenté seul dans un second modèle.

la bibliothèque de modèles du Simulateur IP 1, et les sous-modèles permettant de représenter B ne sont disponibles que dans la bibliothèque de modèles du Simulateur IP 2. On représente donc chaque partie du système en utilisant le simulateur IP le plus approprié, et on relie les deux modèles avec un couplage spatial de niveau 3, pour faire la liaison entre le lien et le nœud B. On respecte les contraintes de simulation et logicielles, en utilisant des appencides pour fermer les réseaux et placer les ports.

### Représentation des appendices dans le modèle IP

L'exemple de la Figure 6.4 illustre l'utilisation de deux appendices : le faux nœud et le lien parfait (avec ses cartes réseau).

D'après la définition de l'appendice donnée dans la Section 5.4.2, celui-ci n'existe pas dans le système qui est à modéliser, et doit donc être transparent vis-à-vis des résultats de simulation. Il ne doit ni modifier les trames et paquets qui voyagent dans le réseau simulé (quel que soit le niveau) ni modifier leurs temps (simulés) d'émission et d'arrivée sur les différentes représentations des composants du système.

Ainsi, introduire un faux nœud dans un modèle consiste à ajouter un nœud qui n'est doté d'aucune application, et qui n'émet aucune donnée sur le réseau. Vis-à-vis du simulateur, il correspond à un nœud vierge aux capacités minimales. Cette coquille vide ne sera utilisée que pour servir de support à une carte réseau : pouvoir en accueillir une est donc la seule compétence qu'il doit pouvoir avoir. S'il dispose d'une couche 3, aucune donnée ne devrait jamais l'atteindre.

Un lien parfait est un lien qui permet de faire passer un paquet ou une trame d'une carte réseau à l'autre, de façon instantanée (i.e. sans consommer de temps de simulation). Les « fausses » cartes réseau qui y sont associées, ainsi que le protocole de couche 2 qui est utilisé, ne doivent pas non plus consommer de temps de simulation.

La solution pour arriver à cette perfection consiste à bluffer le simulateur, en lui demandant de calculer des temps de transmission qui atteignent une échelle de temps plus petite que ce qu'il est capable de représenter. Un lien qui est paramétré avec un débit maximum de $10^{10}$ Tb/s mettra par exemple moins d'une picoseconde pour transférer 1 Go de données. Un simulateur qui n'est pas capable de représenter le temps au-delà de la picoseconde (ou qui est configuré pour être moins précis que cela), estimera que le transfert est instantané au moment de la simulation. Les cartes doivent également être configurées pour avoir un délai de zéro, et le lien doit utiliser un protocole de couche 2 sans adressage physique, de type pair-à-pair. Ainsi, le lien (et ses cartes réseau) peut être qualifié de parfait.

En cas d'utilisation d'un lien parfait dans le cadre d'un couplage spatial de niveau 2, le protocole de couche 2 doit être le même que celui utilisé par le vrai lien, qui est représenté dans le modèle couplé.

### Représentation des ports dans le modèle IP

Pour ajouter la notion de port spatial aux modèles IP, il est nécessaire de comprendre à quel niveau des couches de la suite Internet ils doivent pouvoir intervenir. Cette information permettra de choisir la bonne méthode, en fonction des possibilités offertes par le simulateur IP, pour donner la possibilité au modélisateur de définir des ports spatiaux au niveau de sa topologie IP.

Nous avons vu dans la Section 5.5.1 que les cartes réseau étaient l'endroit le plus approprié pour échanger des trames ou des paquets.

Un port DEVS d'entrée sur un port spatial doit par conséquent être capable de transmettre des commandes à une carte réseau simulée, pour demander la transmission d'une trame ou d'un paquet sur le réseau simulé. Son rôle changera légèrement selon s'il est de niveau 3 ou de niveau 2 :

— **De niveau 3 :** transmission d'un paquet IP à la couche 2 avec l'appel d'une fonction du type `L2->process(pkt)`.
— **De niveau 2 :** transmission d'une trame directement au lien avec l'appel d'une fonction du type `L2->send(frame)`.

À l'inverse, un port DEVS de sortie sur un port spatial doit être capable d'intercepter une trame ou un paquet qui provient du réseau simulé. Il doit donc correspondre à une fonction personnalisée, qui est appelée lorsqu'une trame ou un paquet est reçu par la carte réseau. De plus, cette fonction doit pouvoir agir en coupure dans la pile des protocoles. C'est-à-dire qu'elle doit être capable d'empêcher le paquet ou la trame d'atteindre la couche 2 ou la couche 3 (selon le niveau du couplage) du nœud, durant sa progression vers les couches supérieures, afin qu'il ne soit pas traité par le (faux) nœud sur lequel la carte réseau est installée.

Contrairement aux ports structurels, un port spatial d'entrée ne peut être constitué que d'un seul port DEVS d'entrée, un port spatial de sortie d'un seul port DEVS de sortie, et un port spatial bidirectionnel d'un seul port DEVS des deux catégories.

### Interactions entre les ports spatiaux et la plateforme de co-simulation

Pour que les trames et paquets simulés puissent transiter d'un modèle IP à l'autre, les ports DEVS doivent pouvoir faire la passerelle entre le modèle (le réseau simulé) et le reste du multi-modèle (la plateforme de co-simulation).

Un exemple du comportement attendu pour les ports DEVS des ports spatiaux est donné dans le diagramme de séquence de la Figure 6.5 (en supposant un couplage de niveau 3 et l'envoi d'un paquet de A vers B, comme dans l'exemple de la Figure 6.4). La phrase d'initialisation des ports spatiaux est quasiment instantanée, puisque la présence des ports n'a aucun impact direct sur le fonctionnement du réseau simulé (ils correspondent à des fonctions qui attendent passivement de recevoir ou d'intercepter des données). Les « échanges intermédiaires » qui ont été éclipsés en bas à droite correspondent aux échanges pouvant être engendrés par l'exécution des protocoles associés à la couche 2 (e.g. résolution des adresses IP). L'envoi et la réception qui

suivent peuvent éventuellement ne pas avoir lieu (en bas à droite), par exemple si l'adresse IP de destination ne peut pas être résolue.

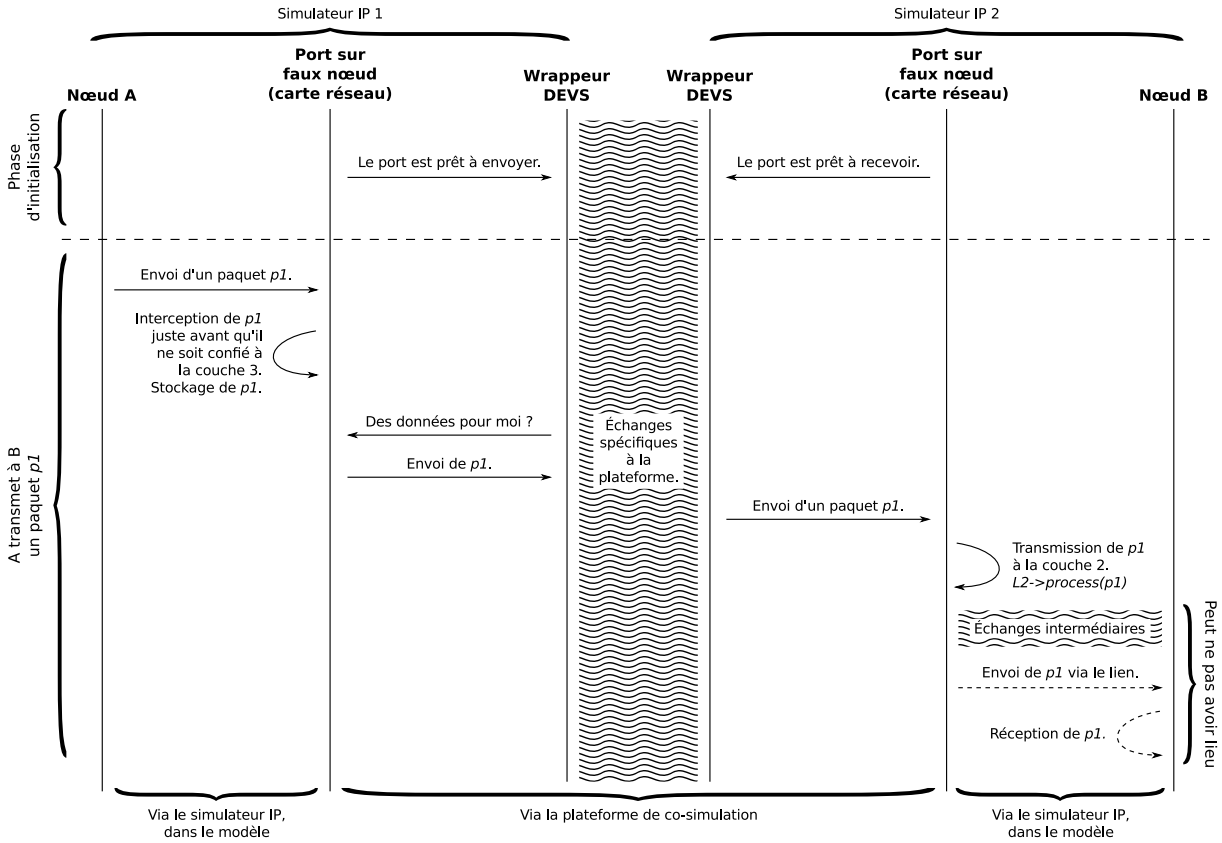



FIGURE 6.5 – Diagramme de séquence correspondant au fonctionnement des ports spatiaux (niveau 3), avec l'exemple d'un paquet IP *p1* envoyé de A (port de sortie) à B (port d'entrée), via la plateforme de co-simulation. Correspond à l'exemple de la Figure 6.4.

Pour un couplage de niveau 2, et donc la transmission d'une trame *t1*, les différences sur le diagramme auraient été les suivantes :

— interception de *t1* avant qu'elle ne soit traitée par la couche 2 (en haut à gauche) ;
— *L2->send(t1)* au lieu de *L2->process(p1)* (en bas à droite) ;
— aucun échange intermédiaire ne devrait avoir lieu, et l'envoi et la réception (mais pas forcément son acceptation) de *t1* devraient toujours avoir lieu (en bas à droite).

Les interactions avec la plateforme de co-simulation, visibles dans le diagrammes, doivent s'intercaler avec le traitement des trames et des paquets au niveau des cartes réseau. Le schéma de la Figure 6.6 permet de faire le lien entre la représentation du réseau IP simulé, et la plateforme de co-simulation. Cette dernière est représentée par ses artéfacts de couplage (cf. Section 2.5.5 pour les notions relatives à la plateforme MECSYCO), qui s'intercalent aux emplacement de la pile des fonctions, auxquels les ports DEVS d'entrée et de sortie doivent être situés.

On constate que les principales étapes qui caractérisent le fonctionnement de la couche 2 au niveau des cartes réseau sont (de haut en bas sur la figure) :

— **Transmet :** Il s'agit du passage de la couche 3 à la couche 2 (invocation d'une fonc-

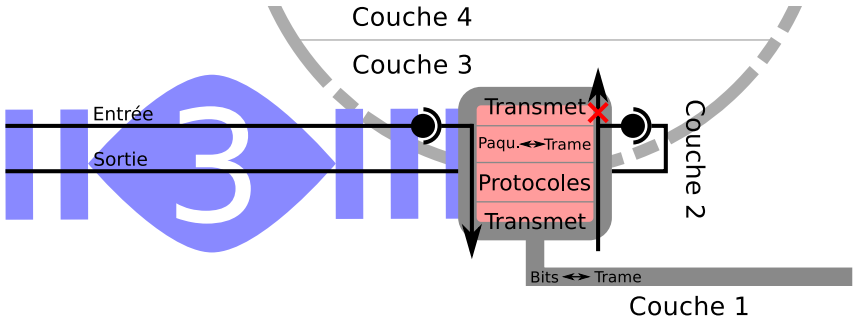



FIGURE 6.6 – Représentation schématique des connexions entre un port spatial de niveau 3 sur une carte réseau, et la plateforme de co-simulation.

tion du type `L2->process(pkt)`), et inversement (invocation d'une fonction du type `L3->process(pkt)`).

— **Paqu. ↔ Trame :** Le paquet est encapsulé dans une trame (function du type `L2->encapsulate(pk` si c'est un envoi sur le réseau, sinon la trame est désencapsulé pour obtenir un paquet (fonction du type `L2->decapsulate(frame)`).

— **Protocoles :** Les protocoles liés à la couche 2 sont exécutés, par exemple pour faire la résolution de l'adresse IP de destination.

— **Transmet :** Il s'agit du passage de la couche 2 à la couche 1 (invocation d'une fonction du type `L1->process(frame)`), et inversement (invocation d'une fonction du type `L2->process(frame)`).

— **Bits ↔ Trame :** La trame est traduite en bits (fonction du type `L1->encode(trame)`), si c'est un envoi sur le réseau, sinon les bits reçus sont traduits en une trame (fonction du type `L1->decode(bits)`).

On constate que l'arrivée d'un paquet IP via la plateforme de co-simulation déclenche une action, qui va injecter ce paquet dans un processus de descente de la pile des protocoles, en commençant par l'encapsulation en trame. C'est le rôle du port DEVS d'entrée. On constate également que l'interception d'un paquet, qui arrive sur le port spatial depuis le réseau, a lieu juste après que la trame ait été désencapsulée. C'est le rôle du port DEVS de sortie.

La Figure 6.7 décrit de la même façon la position et le rôle des ports DEVS, dans le cas où il s'agit d'un couplage de niveau 2.

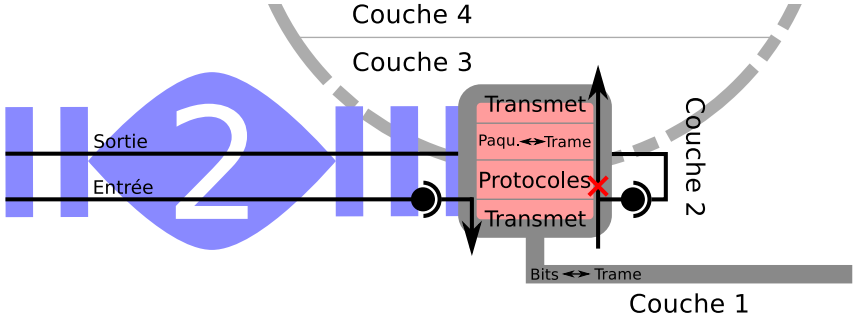



FIGURE 6.7 – Représentation schématique des connexions entre un port spatial de niveau 2 sur une carte réseau, et la plateforme de co-simulation.

### Couplages des nœuds de transit avec partage d'état

Dans certains cas, plusieurs ports DEVS de sortie peuvent être reliés à un même port DEVS d'entrée (et inversement). Cette configuration est utile pour réaliser les couplages de certains nœuds de transit avec partage d'état, et peut être réalisée en utilisant les capacités de la plate-forme de co-simulation.

Les solutions données pour permettre de couper une topologie au milieu d'un nœud de transit impliquent parfois de créer de nombreux couplages spatiaux entre les modèles (cf. Section 5.5.3). Pour cette raison, une représentation spéciale a été proposée dans la Section 5.7. Un exemple de coupure de ce type est proposé dans la Figure 6.8 (on considère que le nœud de transit est de niveau $x$, avec $x$ égal à 2 ou 3).

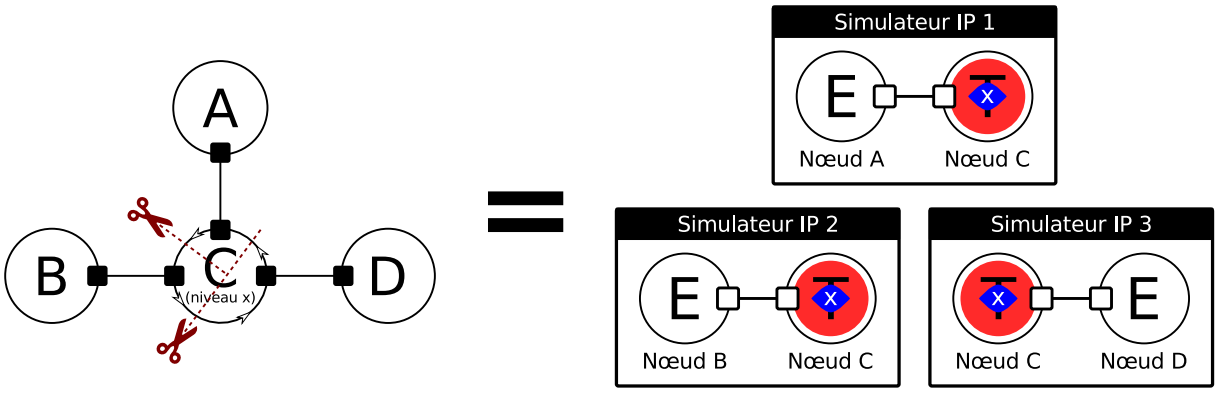



FIGURE 6.8 – Exemple de coupure en $n$ sur un nœud de transit.

La Figure 6.9 offre une version détaillée des couplages spatiaux qui sont à réaliser, pour parvenir à mettre en place le partage d'état du nœud C de la figure précédente. On trouve sur cette même figure les couplages correspondant à réaliser, du point de vue la plateforme de co-simulation. On constate que l'utilisation des multiplexeurs de MECSYCO permet de réduire la complexité du montage à réaliser.

### Limitations

Les différentes modifications de la topologie de réseau, pour placer les ports et donc marquer les frontières des couplages spatiaux, peuvent ajouter un temps de préparation des modèles qui n'est pas négligeable. C'est d'autant plus vrai lorsqu'il faut mettre en place tous les ports et couplages nécessaires pour réaliser des coupes sur des nœuds de transit.

Cette limitation pourra être masquée en offrant la possibilité au modélisateur de s'appuyer sur des assistants (*helpers*), afin de lui permettre d'automatiser un certain nombre de manipulations qui peuvent être systématisées. Par exemple, la mise en place d'un couplage spatial bidirectionnel sur l'interface d'un nœud simple, qui nécessite d'instancier un faux nœud, un lien parfait, créer un port spatial composé d'un port DEVS d'entrée et d'un port DEVS de sortie, définir un nouvel artéfact de couplage au niveau de la plateforme de co-simulation, et relier le tout, peut aisément faire l'objet d'une nouvelle fonctionnalité dans le simulateur.

La possibilité d'ajouter des assistants dépendra principalement des capacités du simulateur à

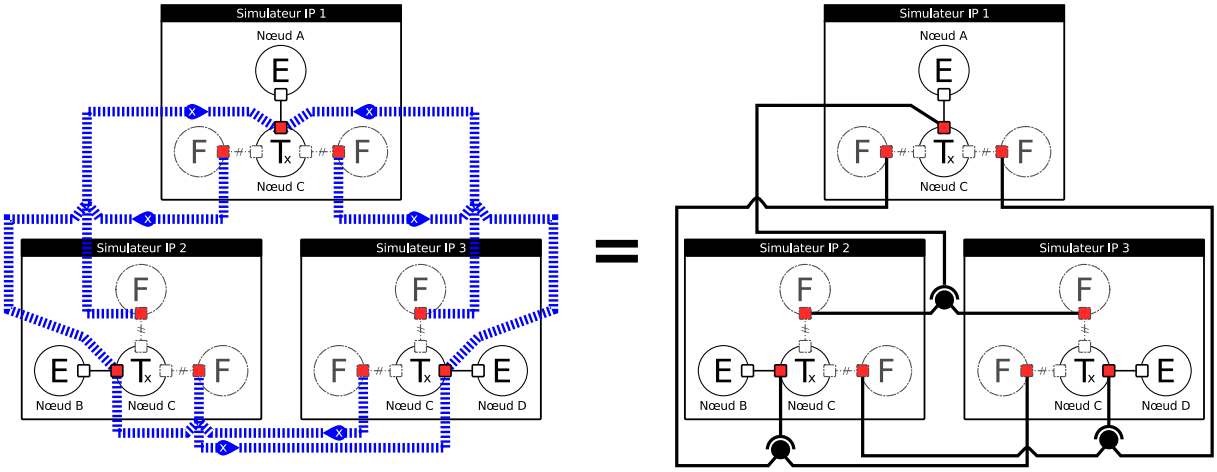



FIGURE 6.9 – Version détaillée des couplages à réaliser, pour l'exemple de la Figure 6.8 (à gauche), avec une version illustrant les artéfacts de couplage MECSYCO qui correspondraient (à droite).

intégrer des fonctionnalités d'utilisateurs. Ils peuvent être importés via un greffon, une extension, un module, une bibliothèque, etc, selon le vocabulaire et le mode de fonctionnement du projet.

### 6.2.5 Conclusion

À partir des concepts de port structurel et de port spatial, détaillés au début de ce chapitre, cette section a permis de comprendre comment il est possible d'ouvrir un modèle IP existant, en le dotant de ports DEVS.

Nous avons étudié la structure interne des ports structurels et spatiaux, indiqué de quelle façon ils étaient composés de port DEVS, et nous les avons précisément situés dans les modèles, d'un point de vue purement IP. Nous avons également détaillé comment ils pouvaient permettre de faire le pont entre la topologie IP simulée et les artéfacts de couplage de la plateforme de co-simulation DEVS, et de quelle façon ils s'y reliaient. Enfin, nous avons abordé les solutions qui permettent de créer les appendices introduits dans les contraintes de simulation, uniquement à partir des éléments qui sont déjà offerts par le simulateur IP.

La section suivante a pour objectif d'expliquer comment ces ports DEVS peuvent être utilisés dans le cadre d'une co-simulation, en ouvrant à présent le simulateur IP grâce à l'ajout de fonctions DEVS.

## 6.3 Ouverture du simulateur : ajout de fonctions

### 6.3.1 Introduction

La section précédente nous a permis de comprendre la structure interne des ports structurels et spatiaux, et de quelle façon ils sont composés de ports DEVS d'entrée et sortie.

Pour que les ports DEVS puissent être connectés à la plateforme de co-simulation, et soient en

capacité d'échanger des événements externes avec le reste du monde, un certain nombre d'étapes doivent encore être réalisées. Le simulateur IP doit notamment pouvoir être compatible avec le protocole de simulation DEVS, de façon à ce qu'il puisse être intégré à la dynamique d'une co-simulation DEVS. Nous aborderons par conséquent les solutions pour wrapper en DEVS un simulateur IP existant.

L'objectif de cette section est de comprendre comment cette intégration est possible, en prenant comme référence la plateforme de co-simulation MECSYCO pour les aspects logiciels. Nous n'hésiterons pas à nous appuyer sur des algorithmes concrets pour décrire les comportements attendus.

### 6.3.2 Rôle du wrappeur DEVS

Le wrappeur DEVS est l'interface chargée de faire le lien entre la plateforme de co-simulation, et le modèle IP et son simulateur. Il a trois principaux rôles :

1. Gérer l'avancement de la simulation du modèle en fonction des ordres reçus par le m-agent.
2. Collecter les données interceptées par les ports DEVS de sortie, et les transmettre au m-agent sous forme d'événements externes de sortie.
3. Recevoir des données sous la forme d'événements externes d'entrée de la part du m-agent, et les transmettre à un port DEVS d'entrée en particulier.

Du point de vue de MECSYCO, il s'agit de l'artéfact de modèle. D'un point de vue logiciel, il s'agit d'un thread séparé de celui du simulateur IP, qui est en capacité de communiquer avec ce dernier. À ce niveau, il n'y a plus de distinction entre les ports structurels et les ports spatiaux, ce sont juste des entrées et des sorties, qui reçoivent et transmettent des événements.

Afin de donner un aperçu global de notre objectif, le schéma de la Figure 6.10 propose un aperçu des interactions souhaitées entre les différents threads impliqués dans l'exécution d'un simulateur IP intégré à MECSYCO.

Selon le format des simulateurs IP, les threads seront instanciés de différentes façons : soit c'est l'utilisateur qui définit le contenu du point d'entrée du programme (le *main*) et donc les threads MECSYCO peuvent être lancés avant ceux du simulateur ; soit c'est le simulateur qui gère lui-même le lancement de sa simulation, et donc les threads MECSYCO doivent être lancés avant l'exécution du premier événement interne, par exemple au niveau du proxy de l'ordonnanceur. Les démonstrations d'intégration de simulateur IP du Chapitre 7 couvriront ces deux modes de fonctionnement.

La première interaction qui a lieu entre le simulateur IP et son artéfact, concerne l'enregistrement de tous les ports du modèle, auprès de l'artéfact de modèle.

### 6.3.3 Enregistrement des ports auprès de l'artéfact de modèle

Nous proposons une solution basée sur un enregistrement pro-actif des ports DEVS auprès de l'artéfact de modèle, de façon à ce que celui-ci puisse rester générique et n'ait pas à avoir connaissance a priori des caractéristiques de ports du modèle à exécuter.

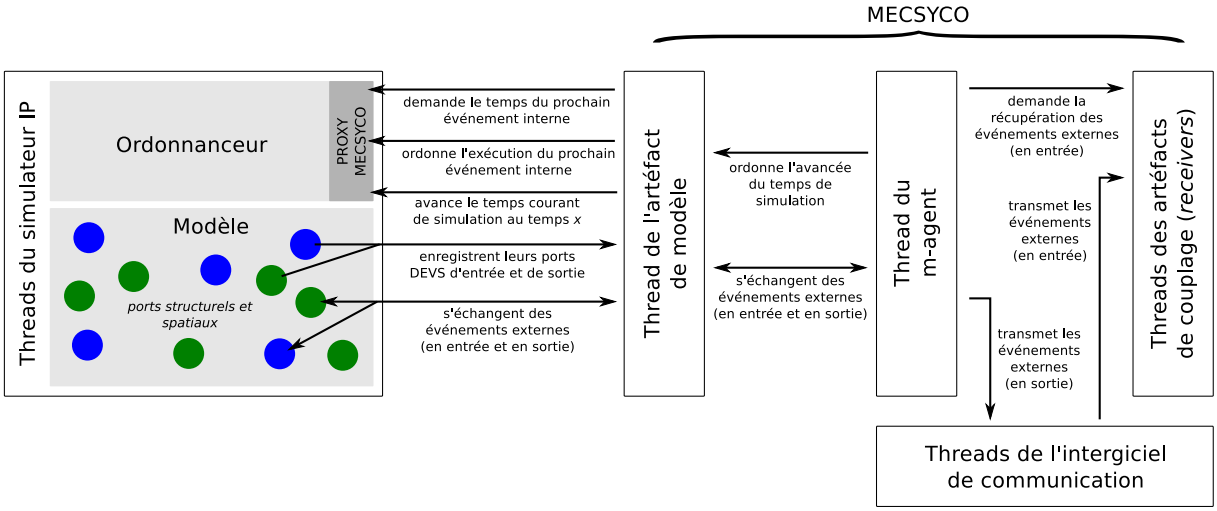



FIGURE 6.10 – Interactions entres les différents threads impliqués dans l'exécution d'un simulateur IP intégré à MECSYCO.

Ainsi, lorsque le modèle IP est développé, le modélisateur a pour charge de placer les ports structurels et spatiaux, dans la topologie de réseau IP qu'il souhaite représenter. Il utilisera pour cela l'implémentation des différents outils définis dans la Section 6.2, éventuellement par le biais d'assistants logiciels, fournis par une bibliothèque associée à MECSYCO. Le modélisateur a également pour rôle d'attribuer un identifiant unique à chacun des ports DEVS qu'il crée, au moment de leur déclaration.

Avant ou pendant que le modèle IP s'initialise, ces différents ports DEVS doivent contacter le thread de l'artéfact de modèle, afin de se faire connaître. L'artéfact de modèle enregistre chacun des ports DEVS dans une liste, et les organise selon s'ils sont de nature structurelle ou spatiale (de niveau 2 ou 3) et selon s'ils ont un rôle d'entrée ou de sortie. Il conserve également leur nom unique ainsi qu'un « pointeur » vers chacun d'entre eux, de façon à être capable de récupérer leurs événements externes de sortie, ou de leur injecter des événements externes d'entrée, selon leur rôle.

Le diagramme de séquence de la Figure 6.11 illustre ce comportement, en proposant l'exemple d'un port structurel entrant nommé « foobar42 » et positionné à l'aide d'un sous-modèle identifié par « id_objet », qui s'enregistre auprès du wrappeur DEVS.

En laissant la possibilité d'utiliser des valeurs comme « spatial2 » ou « spatial3 » à la place de « structurel », et « sortant » à la place de « entrant », le diagramme de séquence de la Figure 6.11 correspond à une proposition de détail des actions « *Le port est prêt à envoyer.* » et « *Le port est prêt à recevoir.* » des diagrammes de séquence des Figures 6.2 et 6.5 du Chapitre 6.

Les autres interactions qui doivent avoir lieu entre l'artéfact de modèle et le simulateur IP correspondent aux fonctions du protocole de simulation DEVS, dont l'implémentation est discutée dans la section suivante.

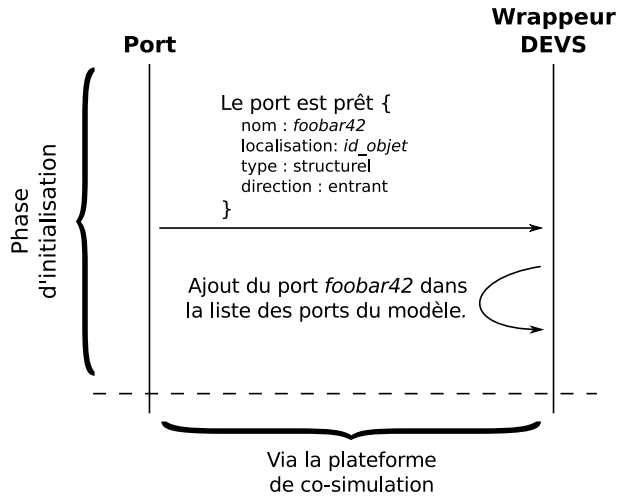



FIGURE 6.11 – Exemple de diagramme de séquence correspondant à l'enregistrement d'un port nommé « foobar42 », auprès de l'artéfact de modèle (wrappeur DEVS), durant la phase d'initialisation du modèle.

### 6.3.4 Implémentation du protocole de simulation DEVS

#### Fonctions du protocole de simulation DEVS

Wrapper le simulateur IP en DEVS implique de lui ajouter les fonctionnalités liées au protocole de simulation DEVS. Dans le cadre de MECSYCO, il suffit d'implémenter les fonctions qui ont été présentées dans la Section 2.5.5 :

- $init()$ initialise le modèle $m_i$, c'est à dire démarre le logiciel de simulation du modèle, fixe les paramètres de ce dernier et son état initial.
- $processExternalInputEvent(ein_i, t_i, x_i^k)$ exécute l'événement externe d'entrée $ein_i$ arrivant au temps de simulation $t_i$ dans $x_i^k$, le $k$-ième port d'entrée de $m_i$.
- $processInternalEvent(t_i)$ exécute l'événement interne du modèle $m_i$ planifié au temps de simulation $t_i$.
- $getExternalOutputEvent(y_k^i)$ retourne $eout_i^k$, l'événement externe de sortie présent dans le $k$-ième port de sortie de $m_i$, $y_k^i$ (ou *null* si aucun événement de sortie n'est présent dans ce port).
- $getNextInternalEventTime()$ retourne le temps du prochain événement interne de $m_i$.

Grâce aux objets correspondants aux ports DEVS que nous avons définis, le développement de ces fonctions est trivial. Elles consistent à utiliser à la fois des fonctions proposées par ces objets, et utiliser des fonctions directement fournies par l'API du simulateur IP (e.g. récupération du temps courant de simulation). Le détail des algorithmes permettant de les implémenter (avec le détail du proxy représenté dans la Figure 6.10) est à disposition dans l'Annexe B.

Certaines de ces fonctions doivent avoir la possibilité de contrôler l'exécution du simulateur IP, ce qui n'est généralement pas trivial à obtenir. Pour implémenter les fonctions du protocole de simulation DEVS, il est donc nécessaire de modifier la dynamique du simulateur IP, de façon à ce qu'il puisse être asservi par le wrappeur DEVS. Nous nous intéressons à cette difficulté, dans la section suivante.

### Asservissement du simulateur IP

Les simulateurs IP existants ne sont généralement pas conçus pour être intégrés à des co-simulations. Ils utilisent par conséquent des ordonnanceurs qui reposent sur une boucle principale dont l'objectif est d'exécuter le plus rapidement possible tous les événements internes dans la pile d'événements, jusqu'à épuisement, marquant ainsi la fin de la simulation. L'Algorithme 6.1 propose un exemple typique d'ordonnanceur indépendant, sans possibilité de contrôle depuis l'extérieur.

```
1 repeat
2 |  processNextEvent()
3 until isEventStackEmpty() || isSimulationStopRequested()
```

Algorithme 6.1 – Exemple de boucle principale utilisée par l'ordonnanceur d'un simulateur IP.

Un exemple d'ordonnanceur modifié pour nos besoins est proposé dans l'Algorithme 6.2, dont l'organisation est commentée ci-dessous.

```
1  repeat
   |  /* Contrôle de l'exécution de la boucle (1/2) */
2  |  waitForWrapperLockOpening()
3  |  closeWrapperLock()
4  |  if ¬isSimulationStopRequested() then
   |  |  /* Injection des événements externes */
5  |  |  if devsWrapper.isThereANewInputExternalEvent() then
6  |  |  |  externalEvent ⟵ devsWrapper.getNewInputExternalEvent()
7  |  |  |  devsPort ⟵ externalEvent["port"]
8  |  |  |  data ⟵ externalEvent["data"]
9  |  |  |  devsPort.send(data)
   |  |  /* Traitement des événements internes */
10 |  |  else if ¬isEventStackEmpty() then
11 |  |  |  eventTime ⟵ getNextEventTime()
12 |  |  |  repeat
13 |  |  |  |  processNextEvent()
14 |  |  |  until isEventStackEmpty() || getNextEventTime() ≠ eventTime ||
   |  |  |   isSimulationStopRequested()
   |  /* Contrôle de l'exécution de la boucle (2/2) */
15 |  openSimulatorLock()
16 until isEventStackEmpty() || isSimulationStopRequested()
```

Algorithme 6.2 – Exemple de boucle principale utilisée par un simulateur IP, modifiée pour pouvoir être contrôlée par un wrappeur DEVS.

L'ajout de verrous dans la boucle principale (lignes 2, 3 et 15) permet de pouvoir arrêter et relancer celle-ci à loisir. En manipulant ces verrous, le wrappeur DEVS a la possibilité d'ordonner

au simulateur IP de traiter son prochain événement interne, et d'attendre le prochain ordre, avant de passer au suivant. Ce système de verrou permet de créer une alternance, entre l'exécution du thread principal du simulateur IP, et celui du wrappeur DEVS (cette technique de génie logiciel est expliquée en détail dans l'Annexe B).

L'exécution d'une itération de la boucle peut donner lieu à deux actions différentes, selon le retour de la fonction *isThereANewInputExternalEvent* (ligne 5), définie par le wrappeur DEVS. On considère que cette fonction retourne vrai si le wrappeur a reçu une donnée en provenance de la plateforme de co-simulation, depuis la dernière fois que sa fonction *getNewInputExternalEvent* a été invoquée. Le cas échéant, la fonction *getNewInputExternalEvent* retourne un objet qui contient une référence vers le port d'entrée concerné, et la donnée à réceptionner.

La première des deux actions consiste à demander le traitement d'un événement externe d'entrée. Cette action est utile pour l'implémentation de la fonction *processExternalInputEvent* du wrappeur DEVS. Déporter ainsi le traitement des événements externes d'entrée dans la boucle principale plutôt que de les exécuter dans le thread du wrappeur DEVS, permet de ne jamais prendre le risque de rendre l'état du modèle incohérent, dans le cas où le simulateur ne serait pas *thread-safe*. La seconde action consiste à faire exécuter le prochain événement interne du simulateur. Cette action est utile pour l'implémentation de la fonction *processInternalEvent* du wrappeur DEVS.

Les lignes 12 à 14 indiquent qu'il peut y avoir plus d'un événement interne qui est exécuté, pour une seule itération de la boucle principale. Les simulateurs IP ont effectivement tendance à programmer énormément d'événements internes à des temps simultanés. Vis-à-vis du wrappeur DEVS, il n'y a aucun problème à exécuter d'un bloc tous les événements qui sont programmés au même temps de simulation. Cette possibilité permet d'optimiser l'exécution de la simulation, en diminuant le nombre d'interactions entre le simulateur IP et son wrappeur.

#### Impact sur le code du simulateur IP

Modifier la boucle principale de l'ordonnanceur du simulateur IP ne nécessite pas nécessairement de modifier directement le code du simulateur (ce qu'on souhaite éviter, en accord avec la seconde contrainte fixée dans la Section 4.2).

Les simulateurs IP proposent généralement dans leur API de remplacer dynamiquement l'ordonnanceur utilisé à l'exécution du modèle (durant l'initialisation de la simulation). Cette modularité permet aux modélisateurs de tester différentes stratégies de simulation, mais elle permet surtout en général d'utiliser un ordonnanceur « temps réel » à la place de celui par défaut. Un ordonnanceur « temps réel » permet de « synchroniser » le temps simulé avec le temps réel, de façon à ce que le modèle puisse éventuellement être relié à du vrai matériel, et donc échanger des paquets avec de vraies cartes réseau. Même si le simulateur IP ne supporte pas ce genre d'interaction avec du matériel, il est possible que la surcharge de l'ordonnanceur soit tout de même prévue, pour permettre d'intégrer cette possibilité à l'avenir.

Nous proposons donc d'utiliser cette possibilité pour ajouter un nouvel objet ordonnanceur à la bibliothèque du simulateur IP. L'ordonnanceur, comme l'artéfact de modèle, est spécifique à un simulateur en particulier, mais doit pouvoir prendre en charge n'importe lequel de ses modèles qui contient des ports structurels et/ou spatiaux, sans modification à apporter.

### 6.3.5 Format des données de simulation

Échanger des données entre simulateurs hétérogènes peut poser des problèmes d'hétérogénéité, comme cela avait été indiqué dans la définition du multi-modèle hybride à la Section 2.4.2.

#### Cas des couplages structurels

Dans le cas d'un couplage structurel, les données n'ont pas nécessairement besoin d'être interprétables par le simulateur ou son modèle. Dans l'exemple de couplage structurel simple donné dans la Figure 6.1 page 94, le simulateur IP 1 n'a pas besoin de comprendre la syntaxe ni la sémantique des données qu'il transporte du simulateur X au simulateur Y. Vis-à-vis de son modèle, il lui suffit de considérer que ce qu'il reçoit et transmet sont des octets, qu'il peut sans problème faire transiter via les sockets UDP/TCP reliées à ses ports DEVS.

Le cas aurait été différent si le couplage avec le simulateur Y n'existait pas, et que les données envoyées par le simulateur X étaient à destination d'une application simulée (e.g. fournie par la bibliothèque de modèles, comme un serveur web) et installée sur le nœud B, dans le modèle IP. Pour que l'application simulée puisse réagir comme convenu, elle aurait dû être capable d'interpréter les données correctement, au niveau syntaxique comme sémantique.

#### Cas des couplages spatiaux

Concernant les couplages spatiaux, le problème de compatibilité du format des données est encore plus critique, en particulier lorsqu'ils interviennent entre deux simulateurs IP.

Au niveau sémantique, certains simulateurs peuvent supporter des protocoles que d'autres ne supportent pas, et donc transmettre des paquets avec des entêtes qui ne sont pas reconnus. Ce point a été largement abordé au niveau des contraintes de modélisation (cf. Section 5.3.1) et logicielles (cf. Section 5.5.2). Toutefois, certains cas d'incompatibilité restent possibles, sans que ça ne pose obligatoirement de problème. Par exemple, si un modèle IP A transmet un paquet TCP à un modèle IP B, alors que le simulateur qui l'exécute ne sait faire que de l'UDP (niveau 4), ce n'est pas gênant si le modèle IP B représente un réseau de commutation (niveau 2).

Au niveau syntaxique, tous les simulateurs IP ne représentent pas obligatoirement leurs trames et paquets de la même façon. Bien que nous ayons supposé dans nos hypothèses (cf. Section 4.2) que tous les simulateurs IP représentaient les trames et paquets de façon atomique, ceux-ci peuvent être représentés sous différents formats, avec différents types d'objet. Le seul format commun qui peut être trouvé entre les différents simulateur est celui utilisé dans le système, standardisé par les différentes RFC des protocoles utilisés. Certains simulateurs (e.g. NS-3) utilisent directement ce format pour représenter les trames et paquets dans leur code. D'autres utilisent un format spécial, mais intègrent des fonctions permettant de convertir leur format maison vers et depuis le format standardisé (c'est généralement le cas, lorsque le simulateur est capable de connecter ses modèles à du matériel réel). Ces fonctions peuvent être utilisées directement dans l'artéfact de modèle MECSYCO, avant de transmettre les événements externes depuis et vers les ports spatiaux, ou dans des opérations de transformation sur les artéfacts de couplage.

#### Opérations de transformation

Les incompatibilités syntaxiques ou sémantiques ne peuvent pas toujours être résolues (notamment s'il manque des informations), mais elle peuvent souvent être contournées grâce à l'utilisation des opérations de transformation des artéfacts de couplage MECSYCO.

Comme expliqué dans la Section 2.5.5, ces opérations peuvent par exemple permettre de transformer un paquet IP représenté sous forme de table de hashage, en un paquet IP au format standardisé (e.g. en s'appuyant sur l'utilisation d'une bibliothèque de type *pcap*). Il sera également possible dans le cadre d'un couplage structurel, par exemple, d'encapsuler une valeur dans un morceau de XML, pour que l'application de destination puisse interpréter la donnée.

### 6.3.6 Conclusion

Cette section a permis de comprendre comment il est possible d'ouvrir un simulateur IP, en le rendant compatible avec le formalisme DEVS.

Pour ce faire, nous lui avons ajouté les fonctions essentielles du protocole de simulation DEVS, en s'appuyant sur un wrappeur DEVS, et notamment sur l'implémentation offerte par MECSYCO grâce à la notion d'artéfact de modèle. Ces fonctions sont à implémenter pour chaque artéfact de modèle, de chacun des simulateurs à intégrer (le détail des algorithmes étant disponible dans l'Annexe B). Grâce au système d'enregistrement des ports, elles sont toutefois indépendantes du modèle utilisé par le simulateur, et ne doivent donc être écrites qu'une fois pour toutes, pour un simulateur donné. Grâce à la notion de surcharge d'ordonnanceur, nous avons vu qu'il était généralement possible de ne pas avoir à modifier le code du simulateur lui-même, pour asservir sa boucle principale. Enfin, nous avons vu dans quelle mesure les problèmes d'hétérogénéité entre les simulateurs (et leurs modèles) du multi-modèle pouvaient être traités, notamment grâce aux opérations de transformation des artéfacts de couplage de MECSYCO.

L'objectif de la dernière section de ce chapitre, est de donner un aperçu des problématiques liées à l'absence de vue globale des modèles.

## 6.4 Absence de vue globale du réseau

### 6.4.1 Introduction

La simulation de fragments de la topologie IP dans des modèles différents, exécutés par des instances de simulateurs IP différentes, ne permet plus à un seul processus d'avoir une vue globale sur l'ensemble du réseau IP modélisé. Cette limitation entraîne quelques effets de bord, qui doivent être pris en compte dès l'instant où des couplages spatiaux sont utilisés pour relier deux modèles IP.

L'objectif de cette section est d'en donner un aperçu. Nous pointerons parfois uniquement le problème, de telle sorte qu'il reviendra à la charge du modélisateur de le prendre en considération lorsqu'il crée ses modèles, et nous proposerons parfois des solutions qui sont proposées dans la littérature scientifique existante. L'intégration de solutions pour répondre efficacement à ces problématiques, dans le cadre des couplages spatiaux, pourra notamment faire l'objet de travaux futurs (e.g. en ajoutant une notion de co-initialisation des modèles).

### 6.4.2 Cohérence des adressages

Les simulateurs IP permettent généralement au modélisateur de ne pas avoir à distribuer manuellement des adresses physiques (MAC) et IP à chacune des interfaces réseau de chacune des machines du réseau simulé.

Dans le cas des adresses physiques, elle peuvent par exemple être simplement générées de façon incrémentale par le simulateur, à chaque fois qu'une nouvelle carte réseau est instanciée, assurant ainsi leur unicité sur tout le réseau modélisé. Pour l'adressage IP, le modélisateur peut par exemple avoir à fournir un lot d'interfaces à un assistant logiciel, accompagné d'une adresse de réseau et de son masque, pour que le simulateur décide de lui-même les associations entre les cartes et les adresses. Une autre solution est encore d'utiliser un sous-modèle permettant de faire du DHCP ou du SLAAC (IPv6 – cf. RFC 4862), pour attribuer des adresses IP aux interfaces directement au cours de la simulation (de préférence en fonction de la méthode effectivement utilisée dans le système).

Il est indispensable de s'assurer que toutes les adresses IP assignées aux interfaces de l'ensemble des modèles sont uniques (sauf pour les appendices, ou pour les adresses de type anycast). Ainsi, pour tous les couplages spatiaux qui séparent la représentation d'un même réseau IP (i.e. domaine de diffusion) en plusieurs parties, le modélisateur doit s'assurer que les adresses IP assignées par les systèmes d'attribution automatique des différents simulateurs en jeu, ne pourront pas se recouvrir.

De la même façon, dès lors qu'un couplage spatial de niveau 2 intervient entre deux modèles, il est nécessaire de s'assurer de l'unicité de toutes les adresses physiques attribuées aux interfaces, au minimum sur l'ensemble des modèles qui partagent le même réseau de commutation.

Toujours dans le cadre d'un couplage spatial de niveau 2, les cartes réseau des faux nœuds doivent avoir la même adresse physique que celle associée au nœud distant qu'elles représentent. Dans le cadre d'un couplage spatial de niveau 3, ce sont les adresses IP qui doivent correspondre. Enfin, si le protocole SLAAC est utilisé pour assigner les adresses IP, elle seront générées à partir des adresses physiques, qui devront donc également correspondre.

Ces problématiques doivent être prise en considération directement par le modélisateur, au moment du développement des différents modèles.

### 6.4.3 Génération des tables de routage

L'éclatement de la topologie IP, dans le cadre des couplages spatiaux, peut également introduire des problèmes au niveau de la cohérence des modèles. Le fonctionnement des simulateurs IP est généralement basé sur la capacité qu'ils ont de pouvoir contrôler l'intégralité des éléments de l'entièreté du réseau IP modélisé. Ainsi, la vue globale qui est à leur disposition leur permet, entre autres, de pré-calculer automatiquement les tables de routage de l'ensemble des nœuds, ou de calculer les routes à la volée durant la simulation.

Dès lors qu'une partie de la topologie du réseau se situe ailleurs, dans un modèle distant, le simulateur ne dispose plus que d'une vue partielle du réseau, et devient donc incapable de générer certaines routes utiles. Dès lors, le modélisateur a donc lui-même à charge de compléter les tables de routages des nœuds IP, de façon à ce qu'ils soient capables d'orienter les paquets en direction des ports spatiaux qui permettront d'atteindre les nœuds distants, auxquels ils sont destinés.

L'exemple de la Figure 6.12 illustre un multi-modèle composé de deux modèles spatialement couplés au niveau 3, et permettant de représenter un nœud A et un nœud B, séparés par deux routeurs branchés en série.

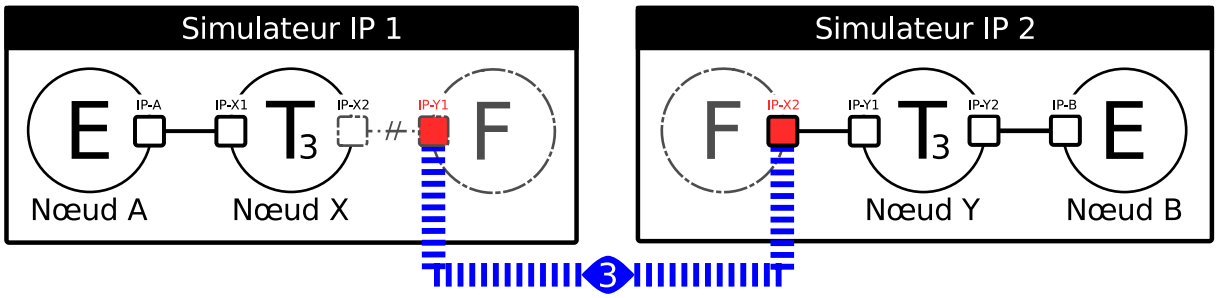



FIGURE 6.12 – A et B sont séparés par deux routeurs en série, qui sont spatialement couplés au niveau 3.

Dans cet exemple, pour que A puisse envoyer un paquet à B, le modélisateur doit définir une route sur le nœud X qui indique qu'il faut passer par *IP-Y1* pour atteindre *IP-B*. De la même façon, pour que B puisse répondre à A, il faut que Y possède une route lui indiquant que *IP-A* est joignable en passant par *IP-X2*.

[Riley et al., 2004] propose une solution qui permet d'éviter de confier cette tâche potentiellement lourde au modélisateur, grâce à la notion de *nœud fantôme* (évoquée dans la Section 3.3.1). Les nœuds fantômes sont utilisés pour représenter les parties de la topologie IP qui ne sont pas censées faire partie du modèle IP, en les représentant de façon minimaliste. De cette façon, chaque simulateur dispose de nouveau d'une vue globale du réseau, a minima suffisante pour calculer toutes les routes de la partie « utile » de la topologie IP de son modèle.

Cette solution est simple à mettre en place, en utilisant les outils que nous avons définis jusqu'ici. En représentant le reste de la topologie uniquement à base d'appendices (e.g. faux nœuds et liens parfaits), les événements qui y seront éventuellement associés n'auront aucun impact sur les résultats de simulation. Un exemple de la mise en place de cette solution, à partir du réseau de la Figure 6.12, est proposé dans la Figure 6.13.

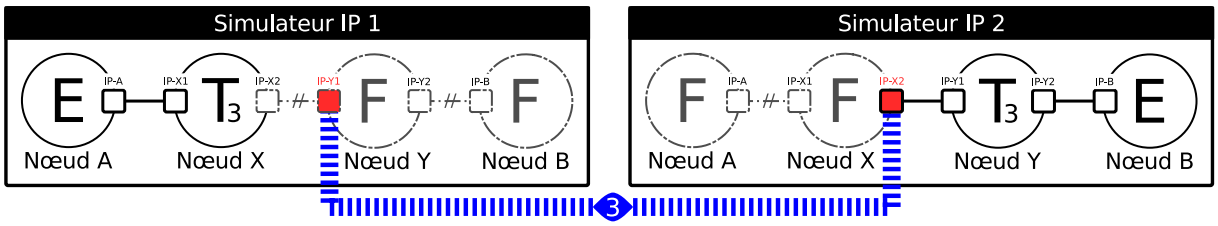



FIGURE 6.13 – Pré-calcul des tables de routage réalisable grâce à la présence de *nœuds fantômes*.

Si cette solution permet bien au modélisateur de ne pas avoir à écrire lui-même les routes, elle lui impose tout de même de recréer l'intégralité de la topologie IP, dans chacun des modèles impliqués. Pour éviter cet inconvénient, selon les simulateurs et leurs limites, le principe des nœuds fantômes pourrait être radicalement optimisé. Plutôt que de recréer l'intégralité du morceau de topologie manquant, attribuer au faux nœud l'ensemble des adresses IP concernées par cette partie, peut parfaitement suffire. Ainsi, dans l'exemple de la Figure 6.13, il suffirait que

l'adresse *IP-B* soit attribuée à l'interface du faux nœud dans le simulateur IP 1, et que *IP-A* soit attribuée à l'interface du faux nœud du simulateur IP 2, pour que les routes de X et Y puissent être calculées sans avoir à modifier la topologie IP.

Dans le cas d'un couplage de niveau 2, la solution serait similaire, mais nécessiterait d'ajouter un faux nœud avec un lien parfait, derrière celui qui accueille le port spatial. C'est ce second faux nœud qui accueillerait les adresses IP. De cette façon, avec le premier faux nœud configuré en routeur, le simulateur ajoutera obligatoirement un saut à toutes les routes, pour atteindre les IP ajoutées. Cette astuce permet de s'assurer que l'adresse physique utilisée pour forger les trames avant le port spatial, sera bien celle du nœud suivant et non celle du nœud de destination.

Si la topologie est amenée à évoluer au cours de la simulation, ces solutions ne sont toutefois pas adaptées.

### 6.4.4 Évolutions de la topologie IP

Selon la question qui est posée sur le système et qui motive sa simulation, des scénarios de défaillance matérielle peuvent avoir à être programmés. Ainsi, un lien ou un nœud de transit particulier devra être considéré comme inutilisable, à partir d'un certain temps de simulation. Sur le système, un protocole de routage dynamique peut être utilisé, pour garantir le recalcul des tables de routage de façon dynamique, dès qu'un changement dans la topologie intervient. Des sous-modèles correspondant à ces protocoles doivent donc pouvoir être utilisés dans les modèles IP.

Les deux protocoles de routage dynamique les plus utilisés dans le monde (OSPF et BGP) fonctionnent au-dessus du niveau 3. Par conséquent, les annonces entre routeurs qui sont liées à ces protocoles sont naturellement transportées d'un modèle IP à l'autre, grâce aux couplages spatiaux. Deux exceptions sont toutefois à prendre en considération dans ce cas :

1. Les modèles ne doivent pas être spatialement couplés sur des nœuds de transit de niveau 3 partageant un état, ce qui signifie que seul le type de coupure « 1 et $x$ » de la Section 5.5.3 peut être utilisé sur un routeur. La raison à cette limitation, est que nous avons supposé dans la Section 5.3.2 qu'*un nœud de transit n'est pas censé émettre lui-même de données, ni en recevoir pour lui-même, mais se contenter de les transmettre d'une interface à l'autre, pour le niveau auquel il se définit.* Les échanges auxquels participent les routeurs avec les protocole de routage dynamique, contredisent cette définition, et invalident certaines de nos propositions dès lors que ces routeurs doivent avoir plusieurs représentations d'eux-mêmes.
2. Les liens qui accueillent des ports spatiaux, et les nœuds qui sont situés sur ce type de lien, ne peuvent pas faire eux-mêmes l'objet d'une défaillance. La raison de cette limitation est que les liens parfaits (resp. les faux nœuds) doivent être synchronisés avec les liens (resp. les nœuds) qu'ils représentent dans les modèles distants, pour que les informations reçues par les protocoles de routage dynamique ne deviennent pas incohérentes d'un modèle à l'autre. Une solution simple pour contourner cette limitation pourrait être de programmer la défaillance au même moment dans les deux modèles : sur l'élément légitime, et sur son appendice.

Conformément à leur définition, il est également nécessaire de veiller à ce qu'aucun faux nœud ne participe aux décisions du protocole de routage.

De façon générale, l'utilisation d'un protocole de routage dynamique est une solution pratique pour correctement initialiser les tables de routage des modèles, y compris lorsque la topologie n'a pas à évoluer au cours du temps. Il est toutefois indispensable que les bibliothèques de l'ensemble des simulateurs impliqués aient en commun de toutes proposer un sous-modèle pour le protocole de routage dynamique souhaité.

## 6.5 Conclusion

Ce chapitre a permis de détailler le fonctionnement attendu des différents concepts décrits dans le chapitre précédent. Ce travail permet d'aller jusqu'à les implémenter dans un simulateur IP avec des algorithmes précis, faisant ainsi le lien entre le niveau conceptuel et le code.

Nous avons notamment vu comment les différents outils nécessaires pour créer les couplages d'un modèle IP, qu'ils soient structurels ou spatiaux, pouvaient être construits en se limitant uniquement à des notions IP. Cette façon de faire permet de pouvoir pleinement exploiter les possibilités offertes par la bibliothèque de sous-modèles IP fournie par le simulateur, et ainsi d'éviter d'avoir à en modifier directement le code.

Nous avons également rapproché la notion de port structurel et spatial à celle de port DEVS, et nous avons vu comment un simulateur IP pouvait être wrappé en DEVS. Ces efforts nous permettent d'intégrer des modèles IP à des multi-modèles DEVS, et des simulateurs IP à des co-simulations DEVS. Ainsi, le travail effectué n'est pas spécifique à un cas d'usage ou à une co-simulation en particulier, mais est suffisamment générique pour permettre aux modèles IP et leurs simulateurs d'être théoriquement intégrés à n'importe quelle plateforme de co-simulation qui reconnaît le formalisme DEVS. La plateforme qui a été choisie pour étudier les aspects liés à l'implémentation de nos concepts est MECSYCO, qui propose nativement la notion d'artéfact de modèle pour créer des wrappeurs DEVS.

De plus, puisque les méthodes que nous avons proposées ne reposent pas sur des solutions ad-hoc ou des cas particuliers :

1. intégrer un nouveau simulateur IP consiste à concevoir et développer une fois pour toutes son wrappeur DEVS, qui pourra être utilisé de façon générique, quel que soit le modèle exécuté et quel(s) que soi(en)t le(s) type(s) de couplage souhaité(s) ;
2. intégrer les modèles du simulateur IP consiste à concevoir et développer une fois pour toutes une collection d'outils, directement au niveau IP et en utilisant la bibliothèque de sous-modèles du simulateur IP, qui permettent de définir et positionner des ports et des appendices.

Le wrappeur DEVS, ainsi que les différents outils pour les modèles IP, pourront ensuite être regroupés sous forme de bibliothèque (ou équivalent) pour le simulateur IP. C'est le sujet du chapitre suivant, avec deux exemples d'intégration de simulateur IP.

# Chapitre 7

# Évaluation et applications

**Sommaire**

## 7.1 Introduction

Ce chapitre correspond au volet pratique de ce manuscrit de thèse.

Nous présenterons les applications de nos travaux en quatre temps :

1. Nous commencerons par expliquer quelle est la démarche à adopter, lorsqu'on souhaite co-simuler un système cyber-physique (SCP) avec des réseaux IP en utilisant nos travaux, en s'appuyant sur un exemple concret. Nous nous placerons dans la position du maître d'œuvre qui doit réaliser la co-simulation, en ayant pour charge de réaliser la modélisation de la partie IP, mais en utilisant des modèles déjà existants pour tout le reste du multi-modèle. Nous ferons donc le point sur les informations qui sont nécessaires pour relier les modèles entre eux et créer le modèle IP, et nous expliquerons les différentes étapes à suivre.
2. Une fois la démarche acquise, nous nous focaliserons sur la conception et l'utilisation des bibliothèques MECSYCO pour les simulateurs IP. Pour ce faire, nous détaillerons de quelle façon deux simulateurs IP répandus ont été entièrement intégrés à MECSYCO. Cette section permettra de démontrer que les outils et méthodes conceptuels qui ont été définis dans ce manuscrit, sont réellement implémentables et applicables à des simulateurs IP existants. Nous fournirons ainsi des solutions pratiques pour intégrer d'autres simulateurs IP.
3. Puis, nous proposerons de prouver expérimentalement que la représentation du réseau IP en éclatant la topologie IP sur deux modèles IP distincts reliés par des couplages spatiaux, n'introduit pas de biais dans les résultats de simulation. Nous utiliserons les simulateurs IP dont l'intégration aura été détaillée dans la partie précédente.
4. Enfin, la dernière partie proposera quelques exemples de co-simulation de SCP complets, qui exploitent les nouvelles possibilités offertes par nos travaux. Certains de ces exemples correspondent à des commandes industrielles, réalisées en suivant la démarche détaillée au premier point.

Sur les différents schémas proposés, les trois principaux domaines d'expertise généralement impliqués dans les SCP (cf. Section 1.1.2) seront représentés d'une couleur différente. Pour plus de clarté, les artéfacts de couplage seront également colorés différemment selon s'ils sont utilisés dans le cadre d'un couplage structurel ou spatial, lorsqu'ils sont reliés à un modèle IP, sur les schémas MECSYCO. Les couleurs utilisées sont présentées dans la Figure 7.1.

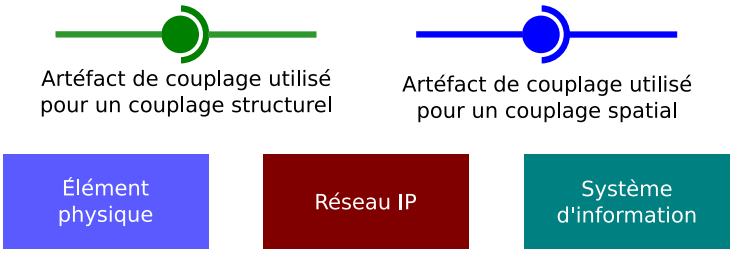



FIGURE 7.1 – Visualisation des couplages structurels et spatiaux au niveau des artéfacts de couplage (en haut), et représentation colorée des trois différents domaines d'expertise impliqués dans des SCP (en bas).

## 7.2 Démarche attendue pour co-simuler un SCP avec des réseaux IP

### 7.2.1 Introduction

L'objectif de cette section est de comprendre quelle est la démarche concrète qui est à mettre en œuvre, dès lors qu'on souhaite construire une co-simulation intégrant des modèles IP, en utilisant MECSYCO et les travaux qui ont été présentés dans ce manuscrit.

Nous nous baserons sur un exemple simple de co-simulation de SCP à réaliser. Nous considérerons que nous avons le rôle de maître d'œuvre pour la réalisation de la co-simulation (relier des modèles existants entre eux) et la réalisation des modèles IP. La fourniture des autres modèles (de type physique et système d'information) est de la responsabilité du maître d'ouvrage, et le détail de leur fonctionnement ne nous intéresse pas directement.

Afin d'illustrer le plus clairement possible la démarche et de démontrer la modularité de notre solution, nous procéderons en trois phases (qui ne sont pas obligatoires) pour construire la co-simulation finale :

1. Une première co-simulation sera réalisée, uniquement en utilisant les modèles fournis par le maître d'ouvrage, sans réseau IP.
2. Cette co-simulation sera modifiée, pour y insérer un modèle IP et donc désormais prendre en considération l'impact du réseau IP sur les résultats de simulation.
3. Elle sera de nouveau modifiée, pour finalement modéliser le réseau IP en s'appuyant sur des sous-modèles issus de deux bibliothèques IP différentes, en utilisant désormais deux modèles IP.

Le passage par la première phase, sans réseau IP, pourrait sembler hors propos. Cette étape préliminaire permet principalement de se familiariser avec la conception de multi-modèles MECSYCO, et de pouvoir ensuite illustrer de quelle façon un multi-modèle existant peut évoluer pour intégrer la prise en considération du réseau IP. Ces trois étapes sont également utiles lorsqu'on souhaite avancer progressivement, et ainsi valider progressivement le montage de la co-simulation, afin notamment de faciliter le déboguage logiciel final.

Le cas d'utilisation sur lequel se base cet exemple concerne un système de chauffage intelligent. Il ne sera décrit qu'au niveau qui est strictement nécessaire, pour réaliser la co-simulation et le modèle IP.

### 7.2.2 Phase 1 : Co-simulation sans modèle IP

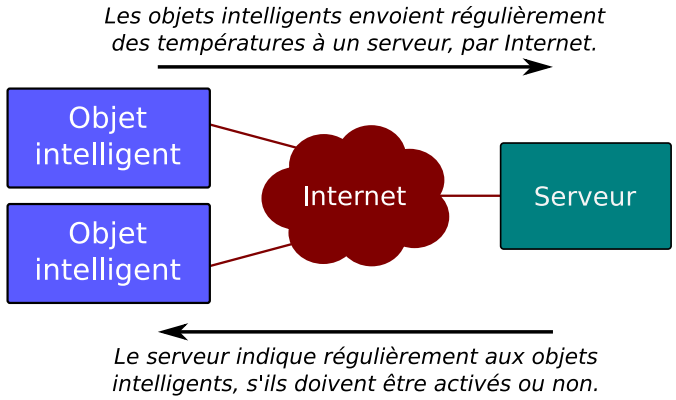
#### Étape 1 : Compréhension du système

La mise en œuvre de la co-simulation nécessite que le maître d'ouvrage soit en capacité d'expliquer ce qu'il souhaite simuler, en restant au niveau de détails nécessaire pour le maître d'œuvre :

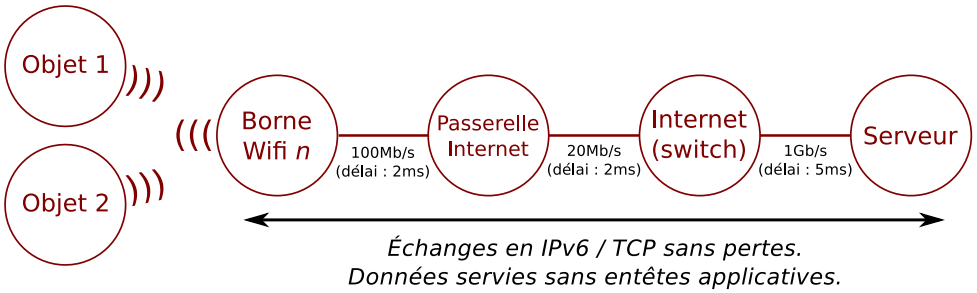
1. La dynamique du système doit être succintement expliquée. Par exemple, dans la Figure 7.2a, on constate que le système est composé de deux objets intelligents, du réseau Internet et d'un serveur. Si chacun de ces composants sera représenté avec un modèle

atomique, il est inutile de détailler leur dynamique interne. En revanche, créer la co-simulation nécessite de connaître la nature des échanges qui auront lieu entre ces composants qui seront individuellement représentés. On apprend dans cet exemple que les objets intelligents envoient régulièrement des températures au serveur distant, via Internet, et que ce dernier a la charge de régulièrement indiquer aux objets intelligents s'ils doivent être activés ou non.

2. Puisque nous sommes également chargés de concevoir le réseau IP, le maître d'ouvrage doit également fournir une description de la façon dont il souhaite qu'il soit représenté. C'est l'objet de l'exemple de schéma réseau qui est en Figure 7.2b. Ce schéma est suffisant, puisqu'il permet de réaliser l'intégralité du modèle de réseau IP, sans se poser de question : on connaît tous les matériels à modéliser, les types de lien et le débit qui les sépare, et on sait comment les données transitent (type de sockets) entre les objets et le serveur. On constate que la partie du réseau qui représente Internet à proprement parler, est réduite à un simple commutateur : ce choix a été fait par le maître d'ouvrage, en cohésion avec le type et le niveau de réponses qu'il attend des résultats de simulation. L'échelle de temps à utiliser pour modéliser le réseau doit également être précisée (e.g. nanosecondes).




(a) Exemple de schéma de description succinte de la dynamique du système à modéliser.




(b) Exemple de schéma réseau, contenant des informations suffisantes pour qu'il puisse être modélisé.

FIGURE 7.2 – Exemples de schémas devant être fournis par le maître d'ouvrage.

### Étape 2 : Compréhension du multi-modèle

Les modèles qui sont fournis directement par le maître d'ouvrage doivent être décrits selon deux aspects principaux (on considère que la technologie utilisée pour décrire les modèles a déjà

fait l'objet d'une intégration dans MECSYCO auparavant, et que les artéfacts de modèle sont donc déjà disponibles) :

1. Leurs interfaces, avec la liste des ports d'entrée et de sortie qui seront à utiliser, doivent être documentées. Ces ports doivent être décrits syntaxiquement (e.g. ils acceptent ou retournent des réels), et sémantiquement (e.g. ces réels correspondent à des températures exprimées en degrés Fahrenheit). Le nom précis qui est utilisé dans le code du modèle doit également être fourni. Un bon exemple est donné par la Figure 7.3 : le comportement des objets intelligents sera représenté par un modèle équationnel (deux fois le même modèle) et celui du serveur sera représenté par un modèle de système d'information avec un automate.
2. La nature et la forme des résultats de simulation attendus, doivent être proposés. Sur la base de ces informations, nous ajouterons un (ou plusieurs) modèle(s) d'observation au multi-modèle, pour collecter et mettre en forme les données produites par les autres modèles, qui permettront de déduire des conclusions de la simulation (dans notre exemple, nous considérerons que suivre l'évolution de la température du premier objet intelligent apporte les réponses nécessaires). Le modèle d'observation peut par exemple être un simple collecteur de données, qui a pour rôle de les stocker en les représentant en fonction du temps simulé, dans un fichier texte.
3. L'échelle de temps utilisée, et la façon dont le temps doit avancer doivent être explicitées. Ainsi par exemple, pour des modèles équationnels, il peut être nécessaire de préciser un pas de temps de résolution et un pas de temps de communication. Si des événements discrets sont associés, il faut préciser comment ceux-ci doivent être détectés et ce qu'ils impliquent. La fréquence de collecte des données à observer peut également devoir faire l'objet d'un choix.
4. Le schéma de calcul doit être fourni : c'est à dire que chacun des ports de sortie décrits, de chacune des instances d'un modèle, doit être relié à au moins un port d'entrée de l'instance d'autre modèle, et inversement. D'un point de vue logiciel, cette information permettra de créer autant de canaux de communication qu'il y a de couples ports d'entrée et de sortie à connecter, comme illustré dans la Figure 7.4 (le modèle d'observation est symbolisé par un œil). Ces canaux seront créés par l'intermédiaire des artéfacts de couplage MECSYCO, et permettront donc de décrire le multi-modèle MECSYCO.

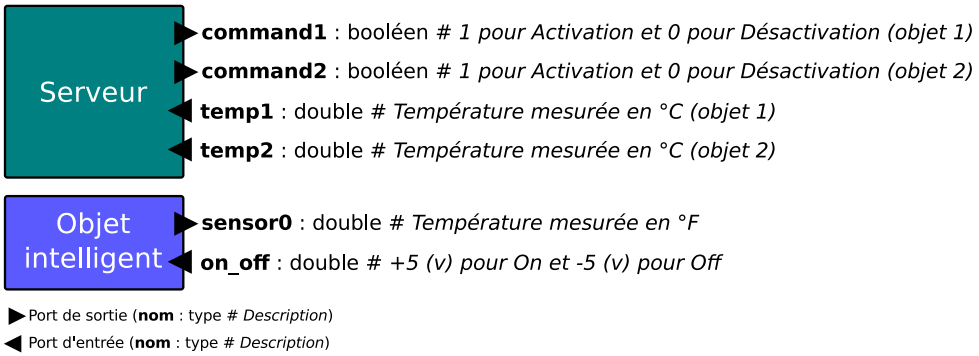



FIGURE 7.3 – Exemple de documentation des interfaces de modèles existants, avec le nom des ports, leur syntaxe et leur sémantique.

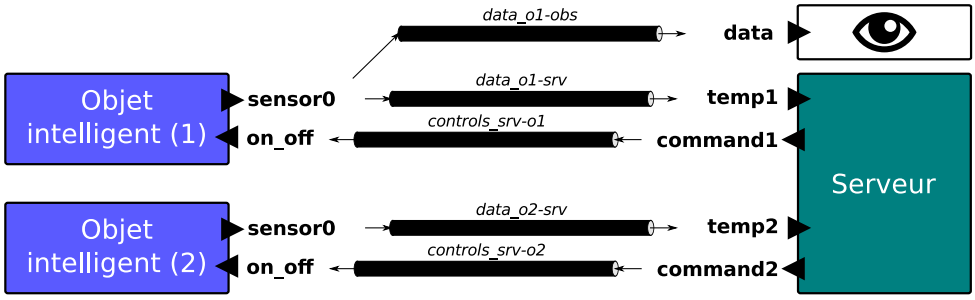



FIGURE 7.4 – Exemple de schéma de calcul d'un multi-modèle, avec le nom des canaux de communication.

### Étape 3 : Intégration des modèles

La mise en place des artéfacts de couplage doit nous interroger sur les problèmes d'interopérabilité, autant au niveau du temps que des données de simulation. Dans les exemples qui ont été donnés, on constate dans la Figure 7.4 que :

1. Les ports de sortie *sensor0* doivent être connectés aux ports d'entrée *tempX*. Si syntaxiquement la connexion des deux ports ne devrait pas poser de problème, puisque ce sont des doubles (sauf problème de représentation différente du type), elle pose un problème d'un point de vue sémantique. Le modèle de serveur accepte des températures en Fahrenheit, tandis que les objets intelligents les mesurent en Celsius.
2. Les ports de sortie *commandX* et d'entrée *on_off* sont pour leur part compatibles d'un point de vue sémantique (ils représentent les mêmes deux états binaires) mais pas syntaxique (doubles vs. booléens).

Puisque ces incompatibilités ne sont pas fortes (elles peuvent être résolues par des mécanismes de conversion), il est possible de créer des opérations de transformation des données de simulation, au niveau des artéfacts de couplage. Un multi-modèle MECSYCO, avec les opérations de transformation qui s'appliquent dans le cadre de cet exemple, est proposé dans la Figure 7.5 (les opérations de transformation sont appliquées à la réception des événements).

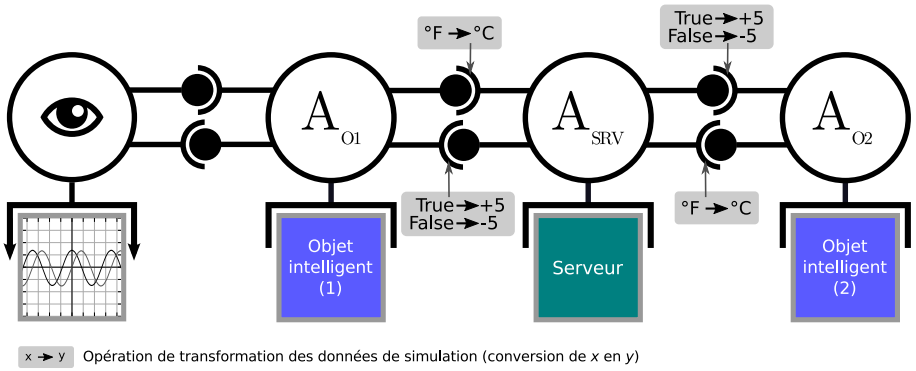


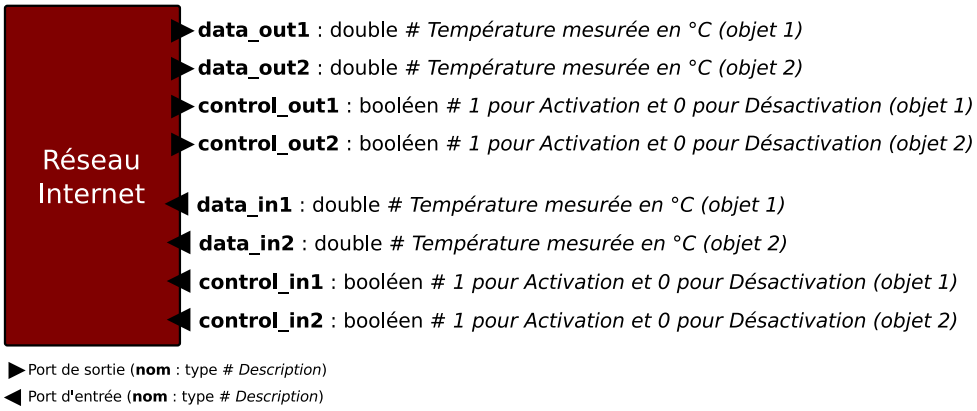

FIGURE 7.5 – Exemple de multi-modèle MECSYCO, avec le détail des opérations de transformation.

Une fois que ce premier multi-modèle est mis en place, son fonctionnement peut être vérifié en exécutant la co-simulation et en vérifiant les résultats de simulation. Nous ajoutons maintenant un modèle de réseau IP, pour pouvoir faire évoluer les résultats de simulation, en fonction de sa configuration.

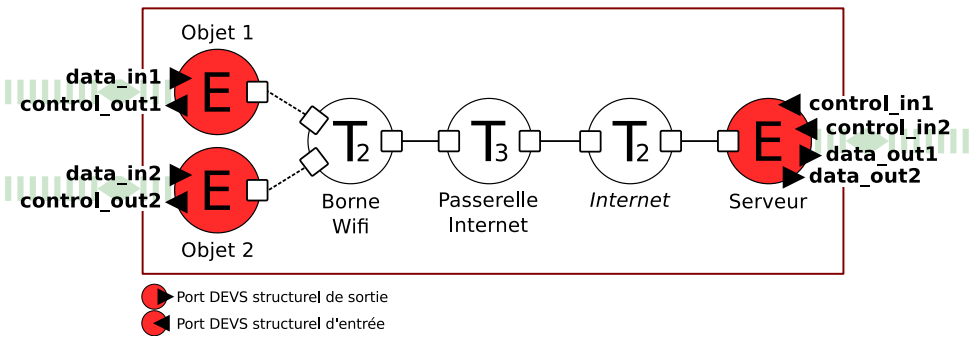
### 7.2.3 Phase 2 : Prise en considération du réseau IP

#### Étape 1 : Conception du modèle IP

Puisque nous avons la charge de réaliser le modèle IP, nous commençons par choisir un simulateur IP qui dispose d'une bibliothèque de sous-modèles IP, permettant de représenter le réseau IP du système, avec le niveau de détails qui a été illustré dans la Figure 7.2b. Nous considérons dans un premier temps que nous avons à notre disposition une bibliothèque IP qui permet à elle seule de modéliser l'intégralité du réseau IP (i.e. Wifi, IPv6, TCP), et dont le simulateur a déjà fait l'objet d'une intégration à MECSYCO. Le modèle IP que nous créons avec le simulateur correspondant, correspond à celui qui est décrit dans la Figure 7.6.




(a) Exemple de documentation de l'interface du modèle IP, avec le nom des ports, leur syntaxe et leur sémantique.




(b) Exemple de représentation du modèle IP, avec le détail des ports DEVS structurels qui sont définis.

FIGURE 7.6 – Définition de modèles IP à coupler structurellement.

Sur les nœuds correspondant aux objets intelligent et au serveur, des ports DEVS structurels ont été définis, pour définir leurs comportements applicatifs avec les modèles externes fournis

par le maître d'ouvrage. Puisque le rôle du modèle IP est de représenter le transfert des données sur le réseau, entre les objets et le serveur, ces dernières doivent être en capacité de « traverser » le modèle IP. Ainsi, par exemple, les niveaux de température produits par le premier objet intelligent peuvent « entrer » par le port d'entrée *data_in1* et « sortir » par le port de sortie *data_out1*, avec une différence en temps simulé entre les deux événements (et éventuellement une altération des données) qui correspondra à l'impact de la présence du réseau IP entre les objets et le serveur.

D'un point de vue pratique, il est nécessaire de :

1. modéliser le réseau IP du système en utilisant les fonctionnalités du simulateur IP ;
2. importer la bibliothèque MECSYCO existante pour ce simulateur ;
3. installer des « applications » sur les nœuds représentant les objets et le serveur, en utilisant les sous-modèles importés par la bibliothèque MECSYCO : ces applications permettent de définir l'emplacement des ports DEVS structurels d'entrée et de sortie, tout en leur associant un nom et un type de donnée.

La prochaine étape est de définir comment ces ports doivent être reliés au reste de la co-simulation, en modifiant le multi-modèle précédemment créé.

### Étape 2 : Modification du multi-modèle

Du point de vue du multi-modèle, le modèle IP s'interpose dorénavant entre les modèles des objets intelligents et celui du serveur. Comme illustré dans la Figure 7.7, il reçoit donc les données produites par les autres modèles, plutôt que de les laisser se les échanger directement. Il va donc pouvoir ainsi influencer les temps simulés auxquels reçoivent l'un et l'autre des modèles, et éventuellement altérer le contenu des échanges.

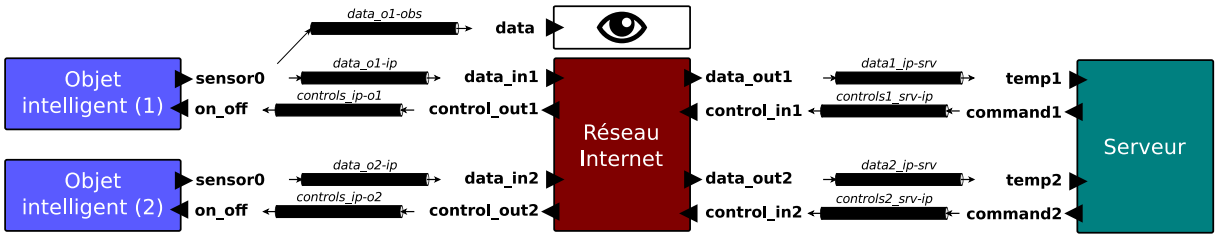



FIGURE 7.7 – Exemple de schéma de calcul du multi-modèle, avec le modèle IP.

Cette étape nécessite de définir de nouveaux canaux de communication, et de méthodiquement les associer à une paire de ports DEVS d'entrée et de sortie, en utilisant leurs noms. Ces canaux de communication seront utilisés par les artéfacts de couplage.

### Étape 3 : Intégration du modèle IP

En définissant l'interface du modèle IP (cf. Figure 7.6a), en accord avec le maître d'ouvrage, nous avons fait le choix de représenter les données de la même façon que le modèle de serveur (i.e. les températures en degrés Celsius et les commandes en valeurs booléennes). Ainsi, il n'y a pas d'incompatibilités ni sémantiques ni syntaxiques entre le modèle de réseau IP et le modèle

de serveur. Par contre, les conversions qui étaient nécessaires jusqu'à présent entre les modèles d'objets et le modèle de serveur, doivent dorénavant avoir lieu entre les modèles d'objets et le modèle IP. La modification du multi-modèle d'un point de vue MECSYCO est illustrée dans la Figure 7.8.

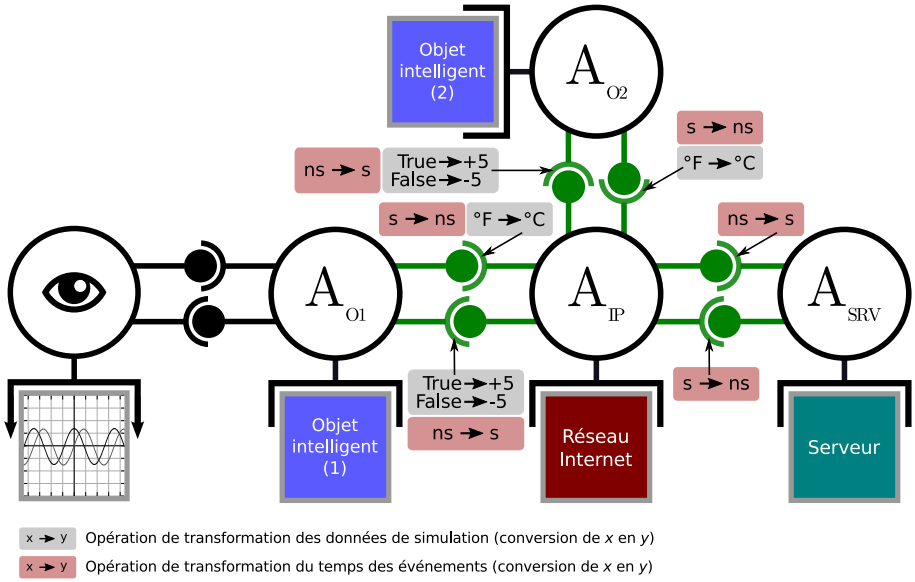



FIGURE 7.8 – Exemple de multi-modèle MECSYCO, avec le détail des opérations de transformation de données et de temps, après intégration du modèle IP.

On constate dans la Figure 7.8 que des opérations de transformation de temps ont été ajoutées entre le modèle IP et tous les autres modèles. En effet, pour les modèles équationnels des objets et pour l'automate du modèle de serveur, une représentation du temps simulé à la seconde est suffisante. Par contre, le niveau de détails souhaité pour la modélisation du réseau IP, nécessite d'utiliser une échelle de temps à la nanoseconde pour celui-ci. Pour que les temps associés aux événements échangés entre le modèle IP et les autres modèles soient compatibles, il est donc nécessaire de systématiquement les convertir de la seconde à la nanoseconde et inversement.

Une fois que ce nouveau multi-modèle est mis en place, nous exécutons la nouvelle co-simulation. Si tout est fonctionnel, les résultats de simulation capturés par le m-agent d'observation devraient a minima révéler des décalages dans les temps simulés associés au événements, par rapport aux résultats obtenus avec la version précédente de la co-simulation. Ces décalages représentent l'impact de la présence d'un réseau IP, qui n'est pas instantané, dans le système.

La dernière évolution du multi-modèle qui est proposée, consiste à représenter le réseau IP en utilisant des sous-modèles IP, qui ne sont pas disponibles pour une même bibliothèque IP.

### 7.2.4 Phase 3 : Utilisation de plusieurs bibliothèques IP

#### Étape 1 : Conception de deux modèles IP

Dans notre exemple, nous considérons désormais que nous souhaitons utiliser une bibliothèque IP A pour représenter le réseau IP domestique (i.e. le réseau Wifi) et une bibliothèque

IP B pour la partie Internet avec le serveur. Les bibliothèques IP A et B n'étant pas disponibles pour un même simulateur IP, nous sommes contraints d'utiliser deux simulateurs IP différents, et donc de définir deux modèles IP séparés.

Les deux modèles IP résultants sont définis dans la Figure 7.9. Les ports structurels qui étaient auparavant définis sur l'unique modèle IP sont désormais répartis entre les deux modèles IP, selon l'endroit où ils apparaissaient dans la topologie IP. Des ports DEVS spatiaux d'entrée et de sortie ont été ajoutés au niveau de la carte réseau de la passerelle Internet, qui permet ainsi de « relier » les deux morceaux de la topologie IP. L'éclatement de la topologie en deux réseaux IP distincts, tel qu'il a été réalisé, correspond à une coupure *1 et 1* (cf. Section 5.5.3) au niveau 3.

D'un point de vue pratique, il est nécessaire de :

1. modéliser les deux morceaux de réseau IP séparément, comme s'il s'agissait de réseaux complets indépendants (cf. méthodes présentées dans la Section 5.5.3), en utilisant les bibliothèques de sous-modèles des simulateurs IP utilisés ;
2. importer la bibliothèque MECSYCO existante, pour chacun de ces simulateurs ;
3. à l'aide des fonctions importées par la bibliothèque MECSYCO, dans chacun des deux modèles : désigner le sous-modèle de carte réseau du nœud représentant la passerelle Internet, comme étant un port spatial bidirectionnel de niveau 3 (en lui association un nom).

La prochaine étape est de modifier à nouveau le multi-modèle, pour définir comment ces nouveaux ports spatiaux doivent être reliés au reste de la co-simulation.

#### Étape 2 : Modification du multi-modèle

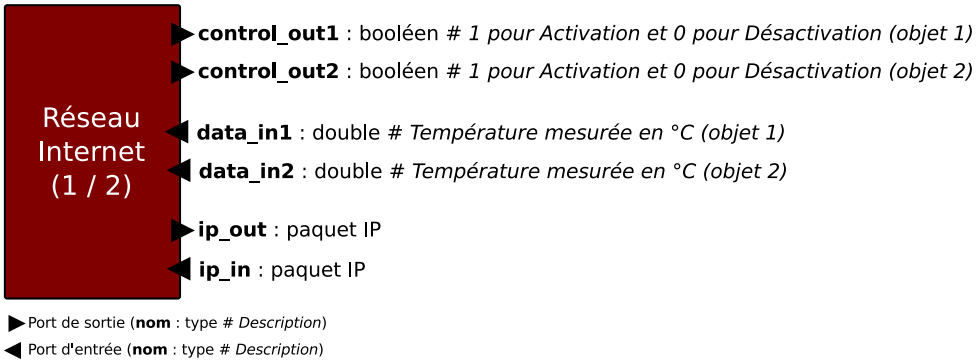
Puisque les ports structurels ont simplement été répartis sur les deux modèles IP, la façon dont ceux-ci sont connectés aux autres modèles, ne change pas. Ainsi, vis-à-vis du reste du reste du multi-modèle, le remplacement d'un seul modèle IP par $n$ modèles IP spatialement couplés est totalement transparent.

La Figure 7.10 illustre la partie du multi-modèle qui change, en introduisant de nouveaux canaux de communication, entre les modèles IP.

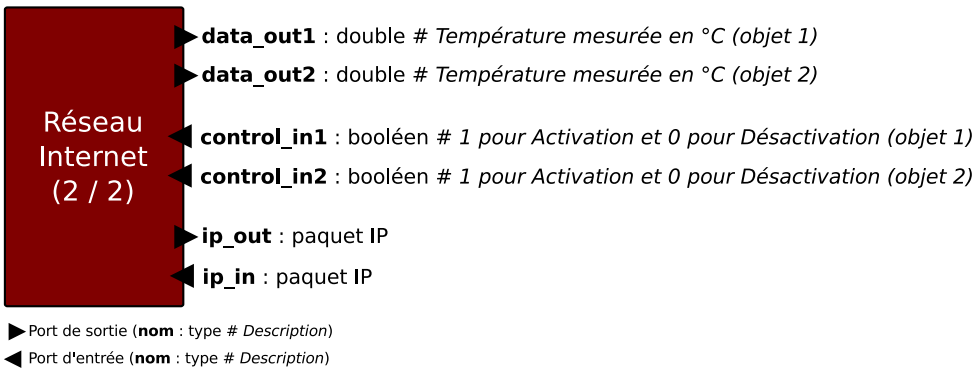
#### Étape 3 : Intégration du second modèle IP

Les opérations de transformation étant liées à des artéfacts de couplage et donc à des ports, les opérations qui existaient dans la version précédente du multi-modèle MECSYCO sont dorénavant réparties entre les modèles IP. L'intégration du second modèle IP est illustrée par la Figure 7.11. L'absence d'opération de transformation du temps entre les modèles IP indique que les deux modèles utilisent la nanoseconde comme échelle de temps, et l'absence d'opération de transformation des données indique que leur représentation des paquets IP est parfaitement compatible (e.g. format *pcap*).

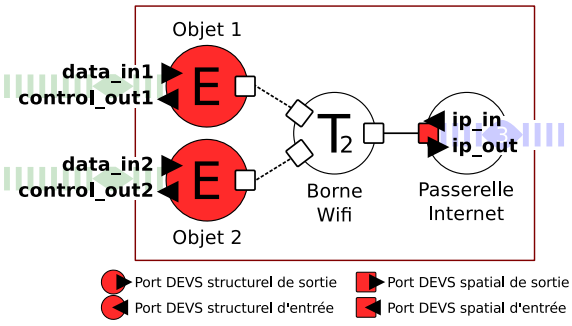
Une fois que ce dernier multi-modèle est mis en place, nous exécutons de nouveau la co-simulation. Si les sous-modèles IP des différentes bibliothèques IP représentent les technologies du système d'une façon strictement identique, alors les résultats de simulation devraient être parfaitement identiques à la version précédente.



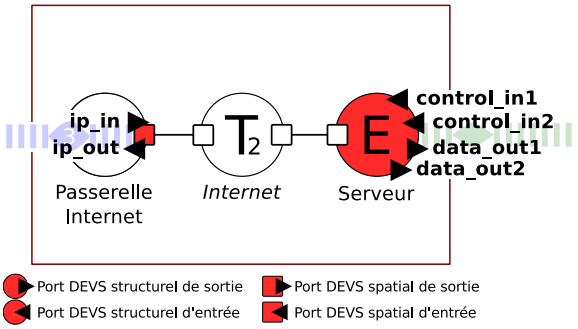

(a) Exemple de documentation de l'interface d'un premier modèle IP destiné à être spatialement couplé à un autre.




(b) Exemple de documentation de l'interface du second modèle IP.




(c) Exemple de représentation du premier modèle IP, avec le détail des ports DEVS spatiaux et structurels qui sont définis.




(d) Exemple de représentation du second modèle IP.

FIGURE 7.9 – Définition de modèles IP à coupler spatialement et structurellement.

### 7.2.5 Conclusion

Cette section a permis d'expliquer pas à pas la démarche permettant d'aller jusqu'à co-simuler un SCP complet, à partir de modèles existants et des spécifications du réseau IP du système, en utilisant des couplages structurels comme spatiaux.

Nous avons vu que le commanditaire de la co-simulation doit être capable de fournir toutes

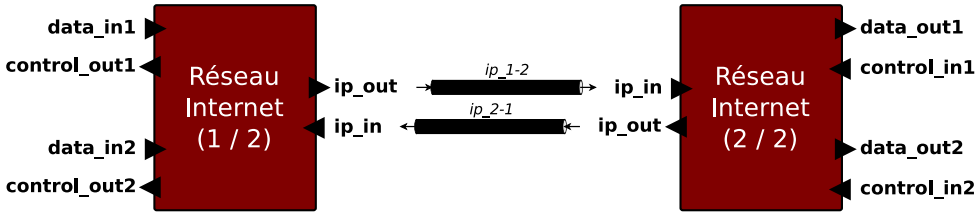



FIGURE 7.10 – Exemple d'extrait de schéma de calcul focalisé sur le couplage spatial entre deux modèles IP.

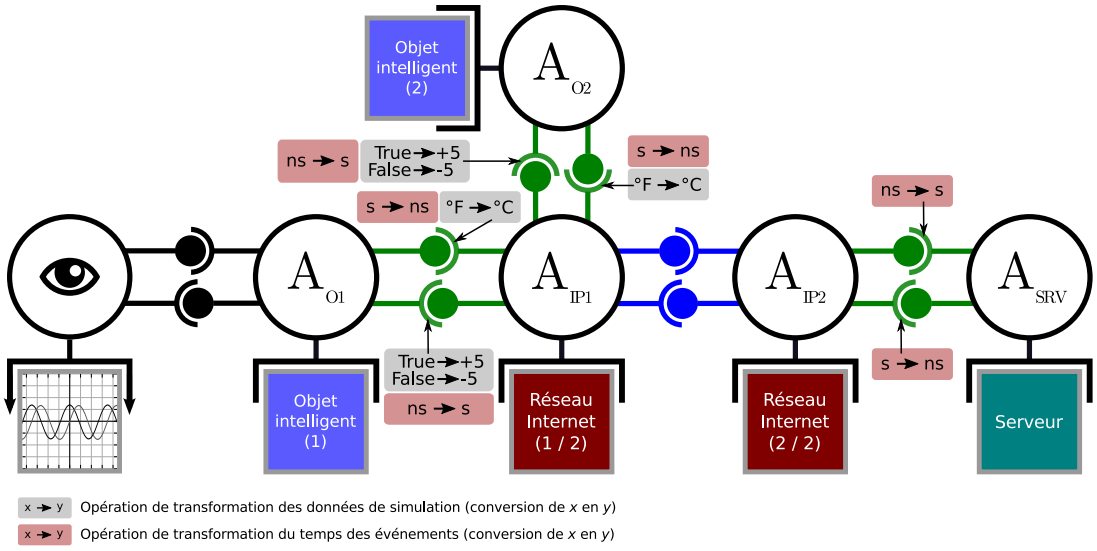



FIGURE 7.11 – Exemple de multi-modèle MECSYCO, avec le détail des opérations de transformation de données et de temps, après intégration du second modèle IP.

les informations requises sur les modèles déjà existants (e.g. documentation des ports, échelle de temps et mode d'avancement du temps), le niveau de détails attendu pour la représentation du système, et le type de résultat de simulation souhaité.

Passer par les trois phases étudiées pour arriver à la co-simulation finale n'est pas obligatoire, et n'est parfois pas possible (notamment si on utilise deux bibliothèques IP différentes parce que l'une ne peut pas représenter ce que permet l'autre). Toutefois, la modularité induite par l'utilisation de MECSYCO permet bien de changer de mode représentation du réseau IP, sans impacter le reste du multi-modèle.

La modification des modèles réseau pour les relier au multi-modèle se résume à utiliser les fonctionnalités du simulateur IP, augmentées par une bibliothèque MECSYCO. Le fonctionnement de ce type de bibliothèque, qui correspond à la mise en pratique des outils présentés dans les chapitres précédents, est détaillé dans la section suivante.

## 7.3 Exemples d'intégration logicielle de simulateurs IP à MECSYCO

### 7.3.1 Introduction

Cette section présente les bibliothèques MECSYCO ont été écrites pour les simulateurs IP NS-3 et OMNeT++/INET, à partir des concepts et solutions proposés dans ce manuscrit. Le fonctionnement des différents outils offerts par ces bibliothèques, est conforme aux solutions proposées dans le Chapitre 6.

Les deux simulateurs que nous avons choisi respectent l'ensemble de nos hypothèses (cf. Section 4.2) et sont très régulièrement cités dans la littérature :

— **NS-3** [Henderson et al., 2006] est un simulateur IP particulièrement utilisé dans le monde scientifique et disposant d'une importante communauté de développeurs. Il est le successeur de NS-2, bien qu'il ait été réécrit depuis zéro, et que la bibliothèque de modèles de ce dernier ne soit pas compatible. Il s'agit d'un logiciel libre distribué sous la licence GNU GPLv2, développé en C++ et permettant de l'utiliser au travers de Python.

— **OMNeT++** [Varga and Hornig, 2008] est un simulateur à événements discrets généraliste, qui a particulièrement été adopté dans la communauté scientifique pour sa capacité à simuler des réseaux. Combiné à la bibliothèque de modèles INET, il devient un vrai simulateur IP, particulièrement complet et utilisé dans le monde scientifique et industriel. Le projet inclus un environnement de développement très complet basé sur Eclipse, avec notamment un éditeur visuel permettant de décrire la topologie d'un modèle de réseau IP. OMNeT++ est un logiciel dont l'intégralité du code source est ouvert, mais qui n'est pas libre. Il est distribué sous licence Academic Public, qui interdit de l'utiliser commercialement [8].

Nous avons vu dans les Sections 3.3.2 et 3.3.2 que quelques travaux avaient déjà réussi à rendre l'exécution de ces deux simulateurs distribuable, mais avec certaines limitations. Ils sont nativement capables de communiquer avec des logiciels ou équipements réels, mais ils ne peuvent pas nativement rejoindre des co-simulations hybrides. Ils ne peuvent pas non plus modéliser des topologies IP en collaborant avec d'autres simulateurs IP qui disposent de sous-modèles complémentaires.

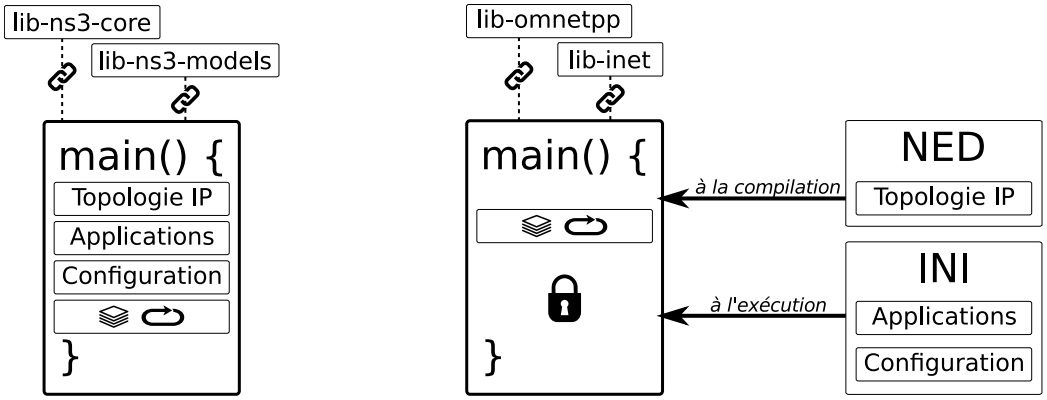
Cette section a pour objectif de démontrer que nos solutions, d'abord définies au niveau conceptuel, nous ont permis d'aller jusqu'à des implémentations concrètes et pleinement fonctionnelles. Pour que ce travail d'implémentation puisse également aider à intégrer d'autres simulateurs IP, nous n'hésiterons pas à donner des indications techniques. La connaissance de NS-3 et OMNeT++/INET est nécessaire pour précisément comprendre la justification de certains choix, mais n'est pas requise pour comprendre le principe des solutions utilisées.

### 7.3.2 Organisation logicielle des simulateurs IP

La façon dont est organisé le simulateur IP détermine directement de quelle manière les composants de la bibliothèque MECSYCO vont pouvoir être conçus. Puisque nous ne souhaitons pas modifier le code des simulateurs eux-mêmes, nous sommes contraints de nous adapter à leur organisation, et d'exploiter au mieux les outils qui sont à notre disposition.

8. Une version commerciale existe toutefois : `https://omnest.com/comparison.php`

NS-3 et OMNeT++/INET sont deux projets dont l'organisation est très différente. Les deux simulateurs IP sont schématisés dans la Figure 7.12, qui sera ensuite détaillée.




(a) Organisation de NS-3.

(b) Organisation de OMNeT++/INET.

FIGURE 7.12 – Comparaison entre l'organisation logicielle de NS-3 et OMNeT++/INET.

Pour NS-3 comme OMNeT++/INET, exécuter un modèle consiste à exécuter un programme binaire (dont le point d'entrée est la fonction *main()*) qui est lié à différentes bibliothèques C++ (symbolisées avec des pointillés et un pictogramme de lien), en lui passant éventuellement des fichiers à la compilation ou en paramètre à l'exécution (symbolisés par d'épaisses flèches entrantes).

On distingue différents blocs communs aux deux schémas :

**Topologie IP** décrit l'architecture physique du réseau, en instanciant des sous-modèles de nœuds (e.g. utilisateur final, routeur, commutateur) et de liens (e.g. filaire catégorie 6, radio Wifi), et en indiquant de quelle façon ces éléments sont reliés entre eux (e.g. le nœud A est directement relié au nœud B avec le lien C). On considère que l'adressage IP fait partie de ce bloc.

**Applications** indique quel sous-modèle d'application (e.g. serveur web, générateur de trafic TCP) est installé sur quel nœud.

**Configuration** contient les informations relatives au déroulé de la simulation : échelle de temps, temps de simulation maximum, configuration des applications, programmation d'événements (e.g. lien coupé à partir du temps $t$), etc. Ce bloc contient également le nom de l'ordonnanceur à utiliser pour exécuter la simulation.

**La pile avec la boucle** (principale) correspond au traitement des événements de la pile d'événements du simulateur, du premier jusqu'au dernier (ou jusqu'à ce que le temps de simulation maximum soit atteint). La présence du symbole de cadenas sur le schéma OMNeT++/INET, sera expliquée un peu plus loin.

Les différents éléments qu'on distingue sur les schémas peuvent être classés en trois parties :

— **Le modèle :** Les blocs *Topologie IP* et *Applications* permettent de définir le modèle à exécuter. Les bibliothèques *lib-ns3-models* et *lib-inet*[9] contiennent les sous-modèles IP officiels des simulateurs, qui permettent de composer les modèles à exécuter.

9. Le nom des bibliothèques citées sur les schémas est symbolique : elles se décomposent en réalité en plusieurs bibliothèques complémentaires, pour une plus grande granularité.

— **Le simulateur :** Les bibliothèques *lib-ns3-core* et *lib-omnet* contiennent les fonctions du simulateur, qui vont permettre l'exécution du modèle. Parmi ces fonctions, on trouve le bloc avec *La pile et la boucle*, qui détermine quand le modèle commence à faire évoluer son état et produire des résultats de simulation, et dont l'arrêt indique la fin de la simulation.
— **Le scénario :** Le bloc *Configuration* permet de paramétrer le modèle avant de l'exécuter, afin de produire des résultats de simulation comparables, dans le cadre d'un plan d'expériences.

Le dernier élément est le point d'entrée du programme, qui correspond à la fonction *main()* des schémas :

— Avec NS-3, le modélisateur a pour responsabilité de créer lui-même la fonction *main()*. Dans ce programme principal, il doit utiliser les sous-modèles de NS-3 pour décrire la topologie IP et les applications de son modèle. En utilisant les fonctions du simulateur, il a également pour responsabilité de configurer le modèle [10] et de lancer lui-même l'exécution de la boucle principale. L'utilisation des sous-modèles IP, la configuration et le lancement de la simulation, sont écrits en C++ [11].
— Avec OMNeT++/INET, le modélisateur n'a pas accès à la fonction *main()* (restriction symbolisée par le cadenas sur le schéma), qui contient la boucle principale, et qui est générée directement par OMNeT++ à la compilation du programme. La topologie IP est décrite dans un fichier texte au format NED de OMNeT++, en faisant référence à des sous-modèles IP. La description des applications à installer sur les nœuds (en faisant également référence à des sous-modèles IP) ainsi que la configuration du modèle, sont décrits dans un autre fichier texte au format INI de OMNeT++. Ces fichiers sont passés en argument au programme principal, à l'exécution. La création d'un sous-modèle personnalisé peut nécessiter l'écriture d'une classe en C++.

L'objectif des sections suivantes est d'expliquer de quelle façon nous avons exploité les espaces de liberté offerts au modélisateur, de sorte qu'à l'exécution d'un modèle, celui-ci puisse rejoindre une co-simulation MECSYCO.

### 7.3.3 Contenu des bibliothèques MECSYCO

NS-3 et OMNeT++/INET tous les deux écrits en C++, et nécessitent tous les deux de passer par une phase de compilation C++ pour générer tout ou partie des modèles [12]. Ainsi, ajouter des fonctionnalités aux modèles peut directement se faire en liant des bibliothèques C++ supplémentaires à leur compilation.

Deux principales bibliothèques supplémentaires doivent être ajoutées à la compilation :

— la bibliothèque qui contient tout le cœur de MECSYCO écrit en C++ ;
— et une bibliothèque MECSYCO spécialement développée pour le simulateur IP.

La bibliothèque spécialisée pour le simulateur contient les outils qui permettront de modifier le modèle et d'agir sur le comportement du simulateur à son exécution. Ces outils dépendent des possibilités offertes par le simulateur IP et des différentes stratégies adoptées. Pour NS-3 et

10. NS-3 permet également de faire varier (par surcharge) les paramètres du modèle en passant des paramètres à l'exécution du programme.
11. NS-3 propose également une API, un peu plus limitée, pour utiliser du Python plutôt que du C++.
12. Des environnements de compilation sont fournis par les deux projets. Celui de OMNeT++ permet notamment d'inclure une compilation préliminaire du fichier NED en fichier C++.

OMNeT++/INET, ils sont relativement similaires :

— un artéfact de modèle MECSYCO (wrappeur DEVS – obligatoire) ;
— un ordonnanceur modifié ;
— deux sous-modèles d'application (port structurel entrant et sortant) ;
— un sous-modèle de carte réseau ou équivalent (ports spatiaux).

### 7.3.4 Ouverture des modèles

Comme indiqué dans la Section 6.2, l'ouverture des modèles consiste à permettre au modélisateur de positionner des ports structurels et spatiaux dans sa topologie IP. Ce besoin intervient dans la Section 7.2, à la première étape de la phase deux pour les ports structurels, et à la première étape de la troisième phase pour les ports spatiaux.

#### Positionnement des ports structurels

D'après la description du fonctionnement des ports structurels proposée dans la Section 6.2.3, positionner un port structurel devrait être équivalent à installer un sous-modèle d'application sur un nœud simulé.

Pour NS-3 comme OMNeT++/INET, définir un nouveau sous-modèle d'application installable sur un nœud consiste à implémenter une classe C++, dans laquelle il est possible d'utiliser les fonctions du simulateur pour gérer des sockets UDP et TCP. Selon si le nœud a vocation à devenir un port structurel entrant (i.e. un client) ou sortant (i.e. un serveur), une application différente doit être installée.

Lors de l'installation d'une application de port structurel sortant, différents paramètres doivent être précisés :

— une chaîne de caractère qui permet d'identifier le port de façon unique dans le modèle ;
— la version de IP qui doit être utilisée pour créer la socket serveur (IPv6 ou IPv4) ;
— le protocole de transport à utiliser pour créer la socket (TCP ou UDP) ;
— le port de couche 4 à utiliser pour créer la socket (e.g. 2048).

Lorsqu'un port structurel entrant doit être installé, les paramètres changent légèrement :

— une chaîne de caractère qui permet d'identifier le port de façon unique dans le modèle ;
— l'adresse IP du nœud à contacter pour connecter la socket (qui détermine aussi la version de IP à utiliser) ;
— le protocole de transport à utiliser pour connecter la socket (TCP ou UDP) ;
— le port de couche 4 à utiliser pour connecter la socket (e.g. 2048).

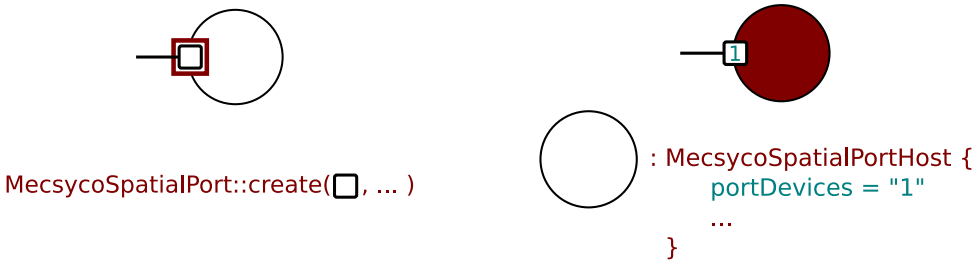
Un même nœud peut accueillir plusieurs ports structurels entrants et sortants, dès lors que le port de couche 4 utilisé est différent.

Les applications de ports structurels pour NS-3 et OMNeT++ s'installent et se configurent de la même façon que tous les autres sous-modèles d'application de ces simulateurs (situés au niveau des blocs *Applications* de la Figure 7.12).

#### Positionnement des ports spatiaux

La description du fonctionnement des ports spatiaux à la Section 6.2.4 indique que positionner un port spatial devrait être similaire à remplacer le sous-modèle de la carte réseau concernée.

En fonction des contraintes techniques observées, des solutions différentes ont été retenues pour NS-3 et OMNeT++, pour obtenir un résultat équivalent. Ainsi, positionner un port spatial dans un modèle NS-3 consiste à wrapper une carte réseau déjà installée sur un nœud simulé, pour la déclarer a posteriori dans le modèle comme un port spatial. Côté OMNeT++/INET, positionner un port spatial est plus intrusif, puisque la solution adoptée nécessite de changer le type de nœud qui doit l'accueillir, en indiquant le numéro de la carte réseau concernée. Les deux logiques sont illustrées dans la Figure 7.13.




(a) Positionnement d'un port spatial dans NS-3, en wrappant la carte réseau (code C++ imagé).

(b) Positionnement d'un port spatial dans OMNeT++, en utilisant un type de nœud spécial (syntaxe NED imagée).

FIGURE 7.13 – Comparaison entre les logiques (imagées) adoptée dans NS-3 et OMNeT++, pour positionner un port spatial.

Lorsque la carte réseau est wrappée ou que le nœud spécial est utilisé, les paramètres suivants dovient être précisés :

— une chaîne de caractère qui permet d'identifier le port de façon unique dans le modèle ;
— le niveau du couplage spatial souhaité (2 ou 3) ;
— la direction du port (entrant, sortant, ou bidirectionnel).

Un même nœud peut accueillir autant de ports spatiaux qu'il a de cartes réseaux installées.

### 7.3.5 Ouverture des simulateurs

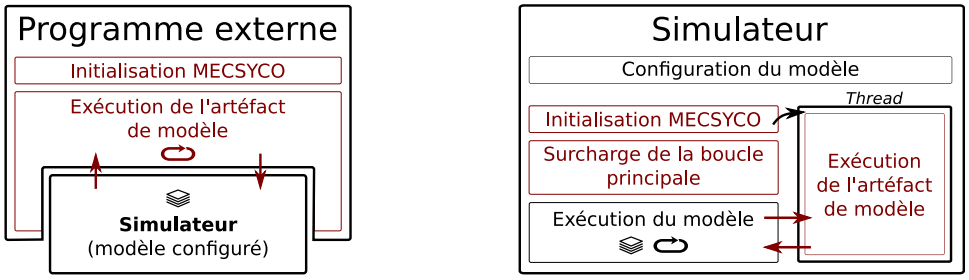
#### Méthodes d'intégration

Avant que le modèle ne commence à être exécuté, il est nécessaire que l'environnement MECSYCO soit entièrement initialisé. Ainsi, l'artéfact de modèle doit impérativement avoir pris le contrôle de la boucle principale avant qu'elle ne soit activée, et que le premier événement ne soit exécuté. Il doit aussi être capable d'injecter des événements externes dans les ports structurels ou spatiaux du modèle, dès le temps zéro.

La méthode adéquate pour intégrer l'initialisation des objets MECSYCO au processus d'initialisation du modèle, diffère très fortement en fonction des choix d'architecture logicielle des simulateurs.

L'approche la plus pratique, consiste à développer l'artéfact de modèle comme un logiciel indépendant, conçu pour communiquer avec le simulateur, depuis l'extérieur (e.g. via des sockets, des arguments, une API, etc, selon ce que propose le simulateur). Ainsi, ni le simulateur ni le modèle n'ont à être modifiés, pour être intégrés à une co-simulation MECSYCO. Cette approche est également intuitive, puisqu'elle est très proche du patron de conception « adaptateur », qui permet l'intégration d'un wrappeur logiciel. Cette façon de faire est illustrée dans la Figure 7.14a, et est utilisée pour intégrer certains simulateurs dans MECYSCO (e.g. NetLogo). Dans ce cas idéal, les objets MECSYCO peuvent être initialisés par le programme externe, avant que ce dernier ne commence à communiquer avec le simulateur pour faire exécuter le modèle.

Lorsque le simulateur ne prévoit pas nativement de méthode de contrôle de son exécution et de ses entrées/sorties depuis l'extérieur, cette solution n'est pas envisageable. Il est alors nécessaire de modifier le simulateur ou son modèle, pour y intégrer directement des instructions spécifiques à MECSYCO, permettant de faire initialiser ses objets au moment adéquat. Parmi ces objets à initialiser se trouve l'artéfact de modèle, qui doit alors être capable de s'exécuter comme un thread. Ce thread, lancé depuis l'intérieur du simulateur, pourra ensuite jouer le même rôle que le programme externe de la solution précédente, en s'exécutant en parallèle de la boucle principale. Il pourra contrôler cette dernière, grâce à la surcharge de l'ordonnanceur du simulateur, qui doit également avoir lieu avant le début d'exécution de la boucle principale. Cette façon de faire est illustrée dans la Figure 7.14b. Elle correspond à ce qui est nécessaire de faire dans le cas de NS-3, comme OMNeT++/INET.




(a) Contrôle d'un simulateur par MECSYCO depuis l'extérieur.

(b) Contrôle d'un simulateur par MECSYCO depuis l'intérieur.

FIGURE 7.14 – Comparaison entre deux stratégies de contrôle d'un simulateur par MECSYCO.

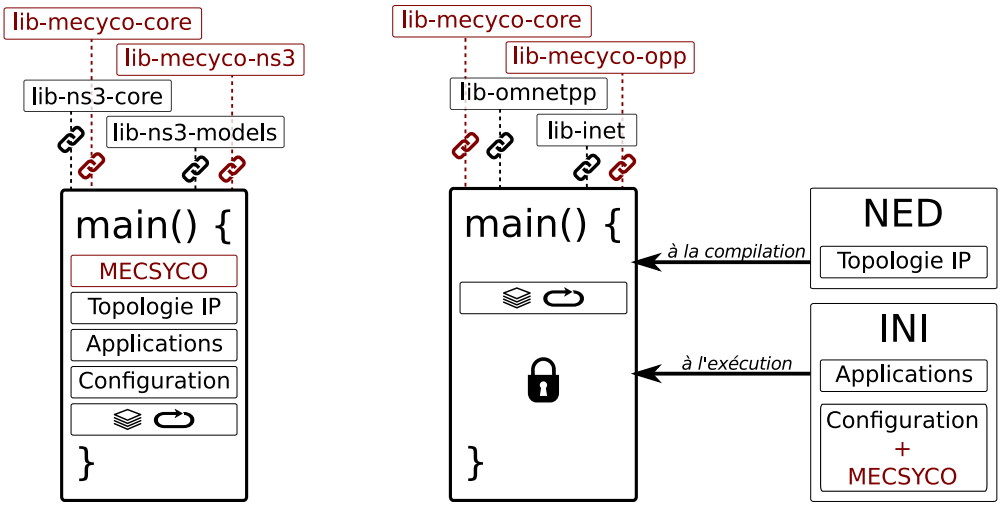
### Intégration de NS-3 et OMNeT++

Nous avons observé dans la Section 7.3.2 que les architectures logicielles de NS-3 et OMNeT++/INET étaient très différentes, et n'offraient pas les mêmes espaces de liberté. Une solution simple et radicale serait de profiter du code-source ouvert des deux projets, pour directement modifier le code des simulateurs de façon à y intégrer du code spécifique à MECSYCO, pour enfin les recompiler. Conformément aux contraintes spécifiées dans le Chapitre 4, nous souhaitons éviter cette solution autant que possible.

L'architecture de NS-3 permet de facilement ajouter du code C++ arbitraire, directement dans la fonction *main()* (point d'entrée du programme) du modèle, avant que le moindre objet NS-3 ne soit initialisé. Intégrer NS-3 consiste donc à compiler le modèle en ajoutant les bibliothèques MECSYCO lors de l'édition des liens, et à instancier les objets MECSYCO avant

d'instancier ceux de NS-3. La modification d'un modèle NS-3 pour MECSYCO est illustrée dans la Figure 7.15a.

Dans le cas de OMNeT++/INET, l'impossibilité d'accéder à la fonction *main()* complique son intégration. Une solution possible est d'ajouter des instructions dans le fichier de configuration (INI) qui est associé au modèle. Puisque ce fichier permet également de spécifier l'ordonnanceur qui est à utiliser, et que ce dernier est écrit en C++, il est possible d'intégrer directement dedans du code MECSYCO capable d'exploiter les instructions de configuration du fichier INI qui ont été ajoutées. De la même façon que NS-3, le modèle OMNeT++/INET peut être compilé en ajoutant les bibliothèques MECSYCO lors de l'édition des liens. La modification d'un modèle OMNeT++/INET pour MECSYCO est illustrée dans la Figure 7.15b.




(a) Configuration de MECSYCO dans NS-3.

(b) Configuration de MECSYCO dans OMNeT++/INET.

FIGURE 7.15 – Ajout de la configuration du m-agent et des artéfacts de couplage MECSYCO, dans NS-3 et OMNeT++/INET.

Le code ajouté pour MECSYCO contiendra des appels à des fonctions mises à disposition par la bibliothèque MECSYCO, permettant notamment de :

1. configurer et instancier un m-agent (notamment en indiquant le lookahead associé au modèle, comme décrit dans la Section 5.4.3) ;
2. configurer et instancier les artéfacts de couplage (cf. deuxièmes étapes des trois phases de la Section 7.2) ;
3. configurer et instancier les opérations des artéfacts de couplage (cf. troisièmes étapes des trois phases de la Section 7.2) ;
4. instancier l'artéfact de modèle, qui fera le lien entre le m-agent et le modèle.

Un exemple de configuration d'artéfacts de couplage, définis en XML pour être interprétés par un fichier INI, est proposé à titre d'illustration dans la Figure 7.1 (pour OMNeT++/INET).

La modification de la boucle principale du simulateur, pour permettre à l'artéfact de modèle d'en prendre le contrôle, est discutée dans la section suivante.

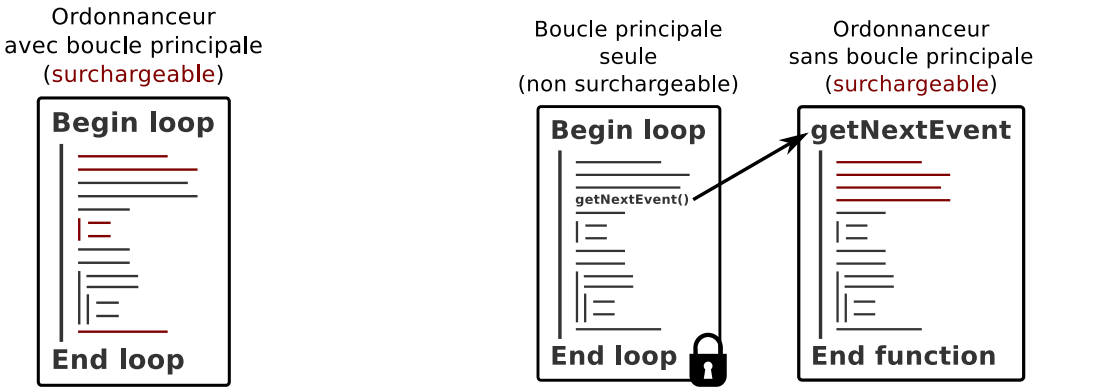
```
<coupling_artifacts>
   <receiver id="fromFoo" type="double" port="fooPortIn" />
   <receiver id="fromBar" type="MecsycoIpPacket" port="barPort" />
   <sender id="toFoo" port="fooPortOut" />
   <sender id="toBar" port="barPort" />
</coupling_artifacts>
```

Implémentation 7.1 – Exemples d'artéfacts de couplage MECSYCO à instancier, décrits en XML via le fichier INI de OMNeT++.

### Modification de la boucle principale

L'ordonnanceur utilisé par le simulateur est obligatoirement étroitement lié à sa boucle principale, puisque c'est lui qui définit de quelle façon et dans quel ordre les événements de la pile doivent être traités.

Selon les simulateurs, il existe deux principaux cas : soit l'ordonnanceur contient directement la définition de la boucle principale (i.e. traitement de tous les événements de la pile, jusqu'à ce que cette dernière soit vide, ou que le temps de simulation maximum soit atteint), soit l'ordonnanceur contient des fonctions qui sont invoquées à l'intérieur de la boucle principale. NS-3 correspond au premier cas, tandis que OMNeT++/INET correspond au second, comme illustré dans la Figure 7.16.




(a) Modification de la boucle principale de NS-3, directement dans l'ordonnanceur.

(b) Modification de la boucle principale de OMNeT++, indirectement par la fonction *getNextEvent()*.

Figure 7.16 – Modification de la boucle principale, dans NS-3 et OMNeT++.

Dans les deux cas, la surcharge de l'ordonnanceur permet néanmoins d'ajouter des actions spécifiques à MECSYCO, entre deux traitements d'événements internes. Ainsi, par exemple, le système de verrous évoqué à la Section 6.3.4 qui permet à l'artéfact de modèle de contrôler l'exécution de la boucle principale depuis un autre thread, peut être mis en place par ce biais.

La Figure 7.2 illustre les lignes à ajouter dans un modèle, qui permettent d'indiquer à NS-3 ou OMNeT++ qu'il est nécessaire d'utiliser l'ordonnanceur mis à disposition par la bibliothèque MECSYCO pour le simuler.

```
// Version NS-3 en C++
GlobalValue::Bind("SimulatorImplementationType", StringValue("ns3::MecsycoSimulatorImpl"));
```

```
// Version OMNeT++ en INI
scheduler-class = MecsycoScheduler
```

Implémentation 7.2 – Surcharge de l'ordonnanceur pour l'exécution du modèle, exprimée en une seule ligne, pour NS-3 comme OMNeT++.

### 7.3.6 Exemples complets

Des exemples détaillés et complets de modèles NS-3 et OMNeT++/INET modifiés pour être intégrés à des co-simulations MECSYCO, sont disponibles en Annexe C.

### 7.3.7 Conclusion

L'implémentation des différents concepts et solutions présentés dans ce manuscrit, avec la réalisation des bibliothèques MECSYCO pour NS-3 et OMNeT++/INET, a permis d'intégrer ces deux simulateurs et leurs modèles à des co-simulations DEVS hybrides. Ces bibliothèques permettent d'appliquer la démarche permettant de co-simuler un SCP complet, qui a été présentée dans la Section 7.2.

Le code des simulateurs n'a pas été modifié, et il suffit de lier ces bibliothèques à la compilation des nouveaux modèles créés, pour que ces derniers puissent :

1. échanger des données avec n'importe quel simulateur déjà intégré à MECSYCO (i.e. compatible DEVS, ou pouvant être wrappé en DEVS) ;
2. distribuer leur simulation sur plusieurs machines (via un modèle IP décomposé en plusieurs modèles IP interconnectés) ;
3. échanger des paquets avec un autre simulateur IP qui dispose de modèles complémentaires pour représenter la topologie IP.

La section suivante a pour objectif de vérifier la validité des résultats de simulation, y compris lorsque des couplages spatiaux (avec appendices) sont utilisés.

## 7.4 Validité des résultats de simulation avec des couplages spatiaux

### 7.4.1 Objectifs et démarche

Les expériences présentées dans cette section ont pour objectif de démontrer que les couplages spatiaux n'ont aucun impact sur les temps de simulation.

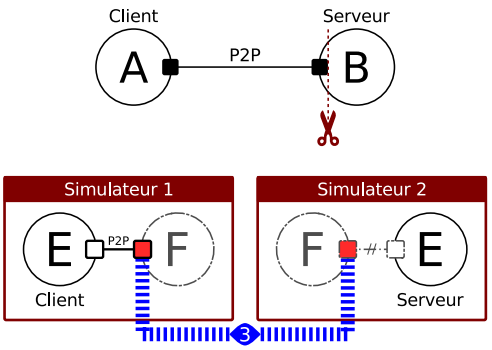
Nous utiliserons deux cas d'utilisation, pour lesquels nous souhaiterons couper la topologie soit entre deux couches, soit au milieu d'un nœud de transit. Nous modéliserons les fragments de topologie IP correspondants, en utilisant deux simulateurs IP différents. Puis, nous simulerons ces cas d'utilisation, en couplant spatialement ces modèles, et en utilisant différentes combinaisons entre les simulateurs. Nous simulerons également ces cas d'utilisation sans couplages

spatiaux, en utilisant un seul simulateur IP, pour les deux simulateurs. La comparaison entre les résultats de simulation obtenus pour les différentes expériences, nous permettra de constater que les couplages spatiaux (et donc également les concepts et outils utilisés pour les réaliser) n'ont pas d'impact sur les résultats de simulation : ils n'introduisent pas de biais, les mêmes résultats sont constatés aux mêmes temps simulés avant et après coupure, et l'introduction des événements mecsyco ne modifie donc pas l'ordonnancement des événements du simulateur.

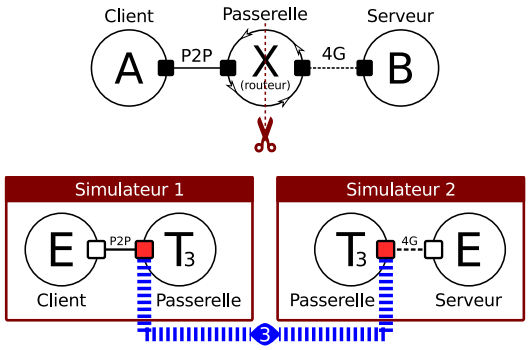
Ces expériences permettront aussi de présenter une première preuve de concept, qui exploite les nouvelles possibilités, offertes par nos travaux.

### 7.4.2 Cas d'utilisation

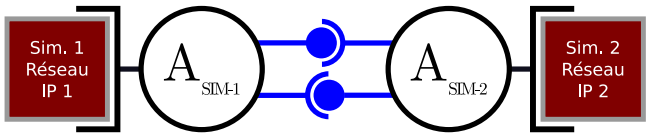
Nous nous appuierons sur les deux cas d'utilisation présentés dans la Figure 7.17. Ces cas mettent en scène un client et un serveur, séparés par un réseau IP, et qui échangent des données entre eux. Dans le premier cas, les deux nœuds sont séparées par un simple lien pair-à-pair (P2P), alors que dans le second cas, ils sont séparés à la fois par un lien P2P et un lien 4G (qui sont eux-mêmes reliés par une passerelle). Ces deux cas sont représentés en haut des Figures 7.17a et 7.17b.




(a) Découpe entre deux couches.




(b) Découpe au milieu d'une couche.




(c) Multi-modèle MECSYCO correspondant aux cas (a) et (b).

FIGURE 7.17 – Couplages spatiaux entre un client et un serveur.

Nous souhaitons modéliser ces deux cas d'utilisation, en utilisant pour chacun deux modèles séparés, exécutés par deux simulateurs différents. Dans le premier cas, on souhaite avoir un premier modèle représentant l'intégralité du système, sauf le serveur, qui est représenté dans un second modèle. Ce premier cas correspond à une coupure entre deux couches, en séparant la carte réseau du nœud serveur. Dans le second cas, on souhaite avoir un premier modèle représentant le client et le réseau P2P, et un second modèle représentant le serveur et le réseau 4G. Ce second cas correspond à une coupure au milieu d'une couche, en coupant en deux le nœud de passerelle (niveau 3), de façon à séparer ses deux interfaces.

Les couplages spatiaux correspondant à ces coupures sont visibles au bas des Figures 7.17a

et 7.17b. On constate l'utilisation de couplages spatiaux bidirectionnels de niveau 3, de ports spatiaux, de faux nœuds et d'un lien parfait. Le multi-modèle MECSYCO qu'on utilise est identique pour les deux expériences et est visible dans la Figure 7.17c. On y constate l'utilisation de deux artéfacts de couplage, permettant l'échange de paquets IP simulés, dans les deux sens.

### 7.4.3 Modèles utilisés

Pour modéliser et simuler ces deux cas d'utilisation, nous utiliserons les intégrations de NS-3 et OMNeT++/INET qui ont été présentées dans la section précédente. Le comportement applicatif du client et du serveur sera défini par des sous-modèles issus des bibliothèques IP proposées par ces deux simulateurs. Nous choisissons de considérer que les interactions entre le client et le serveur correspondent à des pings sur UDP (les deux bibliothèques IP disposent de sous-modèles permettant de modéliser ce type d'application).

Un ping correspond à l'envoi d'une requête (*echo message*) du client à destination du serveur, qui lui renverra systématiquement une réponse (*echo reply*). Le nœud client sera donc équipé d'une application simulée de ping client (envoyant un *echo message* toutes les secondes au serveur), tandis que le nœud serveur sera équipé d'une application simulée de ping serveur (répondant par un *echo reply* à chaque *echo message* reçu).

On considère que les résultats de simulation correspondent aux temps simulés auxquels les réponses du serveur sont reçues par le client. L'échelle de temps utilisée par les simulateurs IP, pour ces expériences, est la nanoseconde.

### 7.4.4 Expériences réalisées

Toutes les expériences décrites ci-dessous sont réalisées pour les deux cas d'utilisation.

On commence par réaliser les expériences qu'on pourra qualifier de témoins, en considérant qu'il n'y a pas de coupures de la topologie et donc pas de couplages spatiaux :

1. Simulation avec un seul modèle, en utilisant NS-3 ;
2. La même chose, mais en utilisant OMNeT++/INET.

On applique ensuite les coupures de la topologie et on met en place les couplages spatiaux.

On commence par coupler les instances d'un même simulateur entre elles (ce qui correspond à une distribution) :

1. Simulation avec deux modèles, en utilisant deux instances de NS-3 ;
2. La même chose, mais en utilisant OMNeT++/INET.

Puis on mélange les simulateurs IP :

1. Simulation avec NS-3 en tant que *Simulateur 1* et OMNeT++/INET en tant que *Simulateur 2* (cf. Figure 7.17) ;
2. La même chose, mais en inversant les rôles de NS-3 et OMNeT++/INET.

### 7.4.5 Groupes d'expériences

Grâce aux expériences sans couplages, on constate que les modèles des applications et du lien P2P, provenant de NS-3 et de OMNeT++/INET, donnent des résultats de simulation strictement identiques. Par contre, on constate que les modèles de 4G sont suffisamment différents pour ne jamais donner exactement les mêmes résultats de simulation, selon si on utilise NS-3 et OMNeT++ (ils sont beaucoup plus complexes donc plus enclins à être implémentés différemment).

Afin de pouvoir comparer les résultats de simulation, on crée trois groupes avec toutes les expériences présentées ci-dessus, réalisées pour les deux cas d'utilisation :

1. toutes les expériences pour lesquelles il n'y a pas de 4G ;
2. toutes celles pour lesquelles la 4G est modélisée avec NS-3 ;
3. la même chose, mais avec OMNeT++/INET.

### 7.4.6 Comparaison des résultats de simulation

Pour chacune des expériences, nous enregistrons les temps de simulation associés à la réception de messages *echo reply* par le client. Nous exécutons ces expériences 100 fois chacune, et nous constatons que nous obtenons exactement les mêmes temps de simulation pour chaque expérience. Nous pouvons donc considérer que les résultats de simulation sont de nature déterministe, y compris lorsque les couplages spatiaux sont utilisés.

Puis, nous comparons les résultats obtenus pour toutes les expériences qui sont réunies au sein d'un même groupe. Nous constatons qu'ils sont systématiquement strictement identiques. Ces comparaisons incluent les expériences témoins, sans couplages spatiaux.

### 7.4.7 Conclusion

Les comparaisons des résultats de simulation nous permettent ainsi de démontrer que les couplages spatiaux sont sans effet sur les résultats de simulation. Les outils utilisés pour les mettre en place, comme les ports et les appendices, ainsi que la modification des ordonnanceurs et l'utilisation de la plateforme MECSYCO, peuvent ainsi être utilisés sans risquer d'influencer les conclusions à tirer des simulations.

Ces expériences ont aussi permis de prouver que nous avions réussi :

1. à distribuer des simulations purement NS-3 et OMNeT++/INET, en utilisant une plateforme basée sur DEVS ;
2. à simuler un réseau IP à l'aide de sous-modèles issus de bibliothèques IP différentes, en faisant communiquer des instances de NS-3 et OMNeT++/INET.

Les sections suivantes sont des exemples d'utilisation de nos travaux, en tant qu'applications industrielles ou en tant que preuves de concept, qui utilisent des couplages spatiaux et/ou structurels avec des modèles IP, dans des multi-modèles hybrides permettant de simuler des SCP complets.

## 7.5 Démonstration de simulation de SCP complets

### 7.5.1 Introduction

Cette section a pour objectif de présenter quelques démonstrations de co-simulation de SCP, qui modélisent les trois domaines d'expertise à la fois, grâce à nos travaux.

Nous verrons notamment trois exemples concrets :

1. Un système de chauffage intelligent de collectivité, qui gère la température de deux bâtiments de onze bureaux chacun et qui permet de faire de l'effacement de consommation électrique. Nous aborderons notamment grâce à cet exemple, l'importance que peut avoir la modélisation des réseaux IP dans les résultats de simulation obtenus pour les SCP.
2. Un réseau électrique intelligent avec cinq maisons équipées de pompes à chaleur, qui permet de tester des scénarios d'effacement cascado-cycliques. Nous verrons notamment grâce à cet exemple, comment nos travaux ont été utilisés d'un point de vue industriel par EDF R&D, en partenariat avec Inria, en intégrant des boîtes noires.
3. Un système de chauffage intelligent individuel, avec jusqu'à vingt maisons équipées de contrôleurs internes connectés à Internet en filaire et en 4G. Nous verrons notamment grâce à cet exemple, comment nos travaux peuvent être utilisés pour comparer des sous-modèles issus de différentes bibliothèques IP.

Une seconde application industrielle a été réalisée par EDF R&D et Inria en utilisant nos travaux, mais elle ne sera pas présentée ici en raison des restrictions en matière de propriété intellectuelle que EDF R&D a imposé.

Pour les trois expériences présentées :

— Les modèles équationnels FMU (cf. Section 2.5.4) ont été intégrés dans les multi-modèles MECSYCO à l'aide du wrappeur DEV&DESS présenté dans [Camus et al., 2016]. Quand ce n'est pas précisé, on considère que les FMU sont de type co-simulation.
— Les adresses IP et les tables de routage ont été définies statiquement dans les modèles IP.
— Lorsque plusieurs instances d'un même modèle sont connectées au même modèle IP, et que des données sont transmises à intervalles fixes, un *jitter* de quelques secondes a été utilisé pour désynchroniser leurs envois. Les envois sont donc effectués dans un temps de simulation compris entre l'instant $t$ auquel le modèle transmet la donnée, et l'instant $t + x$, avec $x$ choisi aléatoirement entre zéro et la valeur du jitter. Cette fonctionnalité est nécessaire pour reproduire une partie du comportement applicatif (destinée à ne pas saturer le réseau) qui n'est pas présente dans les modèles équationnels utilisés. Elle peut être implémentée dans des opérations de transformation du temps avec MECSYCO, ou être directement prévue dans l'implémentation des ports structurels DEVS.

### 7.5.2 Importance de la simulation des réseaux de communication

#### Introduction

Ce premier exemple correspond à la modélisation et simulation d'un système de chauffage intelligent pour une collectivité, doté d'une fonction d'effacement de la consommation électrique.

Cet exemple correspond à la modélisation d'un SCP complet, avec les trois domaines d'expertise réunis.

L'objectif de ce cas d'utilisation est d'illustrer, en utilisant nos travaux, l'intérêt de prendre en considération l'impact des réseaux de communication (IP) dans les résultats de simulation de SCP. Nous verrons notamment dans cet exemple comment le réseau IP peut changer les résultats de simulation, selon la configuration choisie.

### Présentation du système

Le système à modéliser est schématisé dans la Figure 7.18. Il s'agit d'un système de chauffage intelligent de collectivité, qui contrôle les radiateurs de deux bâtiments, chacun composés de dix bureaux et d'un couloir. Chaque pièce est équipée à la fois d'un radiateur et d'un thermomètre, capables d'être reliés à un réseau IP. Tous les radiateurs et les thermomètres sont reliés au contrôleur en filaire. Chaque bâtiment possède son propre réseau, avec un routeur de sortie. Un routeur principal permet de relier les deux bâtiments, ainsi qu'un contrôleur qui est situé à l'extérieur.

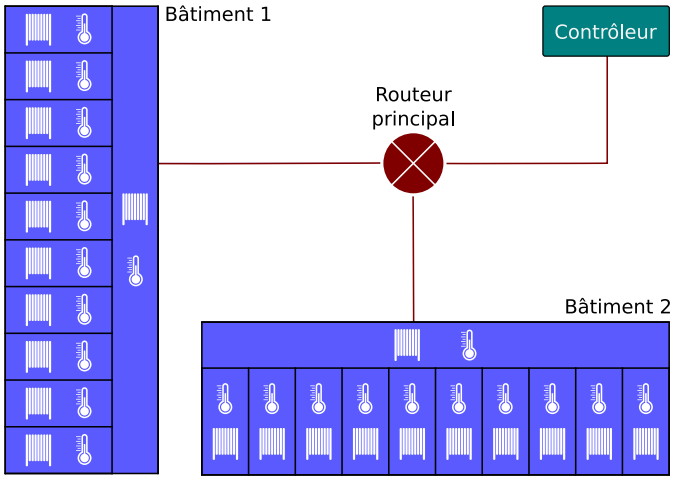



FIGURE 7.18 – Système de chauffage intelligent de collectivité, avec deux bâtiments de dix chambres et un couloir.

Le scénario qu'on souhaite modéliser consiste à faire de l'effacement de consommation électrique, c'est à dire lisser la courbe de charge par le pilotage de la demande. On mesure la charge d'un bâtiment à partir de la puissance instantanée totale de l'ensemble de ses radiateurs. On souhaite que cette puissance totale reste inférieure à un seuil fixé à l'avance (en watts). Si la puissance dépasse le seuil, des radiateurs seront éteints, jusqu'à ce que la contrainte soit de nouveau respectée.

Les décisions sont prises par le contrôleur, auquel sont connectés en IP tous les radiateurs du bâtiment. Tous les radiateurs envoient régulièrement leur puissance instantanée par le réseau IP, au contrôleur. En fonction du calcul de la charge du bâtiment complet et de sa politique, le contrôleur peut lui-même envoyer des ordres d'effacement (extinction ou allumage) par le réseau IP, à chacun des radiateurs, de façon individuelle. Les équipements des deux bâtiments sont tous connectés au même contrôleur, mais ce dernier contrôle les deux bâtiments de façon

indépendante (i.e. chaque bâtiment a son propre seuil à respecter).

Chaque radiateur est configuré avec une consigne de température et une bande de tolérance associée (en K). Grâce à un thermostat interne, il doit essayer de faire en sorte que la température de la pièce soit toujours la plus proche possible de sa consigne, en ayant l'autorisation de la faire osciller de plus ou moins la moitié de la valeur de la bande de tolérance. Pour y parvenir, il peut décider de s'allumer et de s'éteindre, de façon complétement autonome. Lorsqu'il a reçu au préalable un ordre d'extinction de la part du contrôleur, le radiateur ne s'allume pas, même si son thermostat interne indique une température inférieure à la tolérance admise. Par contre, le contrôleur dispose lui-même d'une consigne de température minimum « extrême ». Si la température de la pièce est inférieure à la consigne du contrôleur, ce dernier doit décider d'ordonner au radiateur de se rallumer, et en choisir un autre pour faire de l'effacement. Pour cette raison, les thermomètres situés dans les pièces sont également tous connectés en IP au contrôleur, et l'informent régulièrement de la température actuelle des pièces.

La question naïve qui est posée est la suivante : la configuration du réseau IP peut-elle directement influencer la consommation énergétique des bâtiments ?

### Multi-modèle utilisé

Le multi-modèle correspondant est composé de six modèles distincts :

— 1 modèle thermique représentant la température extérieure au cours de la journée (FMU pour Windows produite avec OpenModelica) ;
— 2 modèles thermiques représentant chacun les radiateurs et les thermomètres des onze pièces d'un bâtiment (FMU pour Windows produites avec OpenModelica) ;
— 2 modèles de système décisionnel représentant chacun une partie du contrôleur dédiée au contrôle d'un bâtiment (programmes java ad-hoc) ;
— 1 modèle de réseaux IP (avec NS-3 pour GNU/Linux).

Les interactions souhaitées entre les différents modèles sont illustrées dans la Figure 7.19. On y retrouve les interactions décrites au niveau du système, dans la partie précédente.

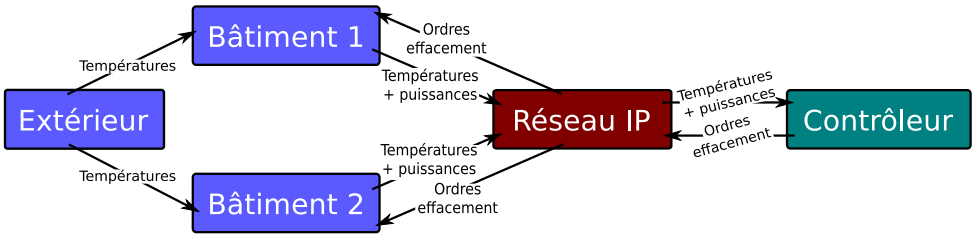



FIGURE 7.19 – Vue intuitive du multi-modèle pour la simulation du système de chauffage intelligent de collectivité.

Bien que le détail du fonctionnement des modèles thermiques soit ici hors propos, le modèle thermique *Extérieur* est utile pour leur permettre de calculer la température des pièces. Ces derniers calculent en effet la température de chaque pièce en prenant en compte les transferts de flux de chaleur qui ont lieu au travers des murs. Afin de pouvoir évaluer les transferts qui ont lieu par l'intermédiaire des murs extérieurs, les modèles des bâtiments sont reliés au modèle permettant de représenter l'évolution de la température extérieure au cours de la journée.

Le multi-modèle MECSYCO utilisé pour la simulation de cet exemple est présenté dans la Figure 7.20.

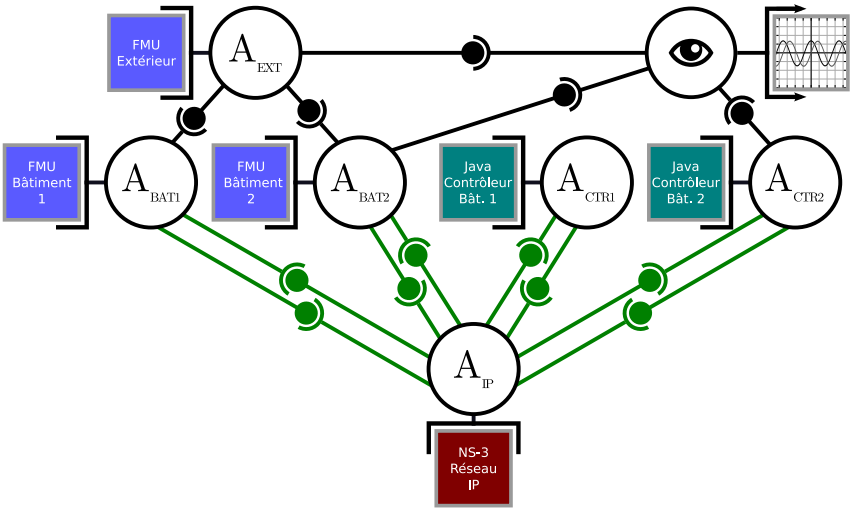



FIGURE 7.20 – Multi-modèle MECSYCO pour la simulation du système de chauffage intelligent de collectivité.

On constate que le contrôleur est en réalité représenté par deux instance d'un même modèle, chacune dédiée à un bâtiment. Cette représentation fragmentée du contrôleur est possible parce que la politique énergétique de chaque bâtiment étant indépendante, les données de l'un ne peuvent pas influencer l'autre, et les décisions prises pour l'un ne concernent pas l'autre. L'intégralité du réseau IP est représenté avec un seul modèle IP.

Les couplages entre le modèle de réseau IP et les différents modèles sont tous de nature structurelle, parce que les radiateurs, comme les thermomètres et le contrôleur, sont des équipements (capteurs et serveur) qui produisent et échangent des données de type applicatif. Un m-agent d'observation a été ajouté au multi-modèle, afin d'observer l'évolution des températures des pièces au cours du temps. Il reçoit également régulièrement les puissances instantanées des radiateurs, ainsi que les ordres d'effacement (allumage et extinction) transmis par le contrôleur.

### Modèle IP utilisé

La topologie IP correspondant au système qui a été décrit est visible dans le schéma de la Figure 7.21. Les points de suspension sont des ellipses, qui indiquent que tous les équipements compris entre 1 et 11 sont également présents, mais non représentés (pour des raisons de lisibilité).

Chaque radiateur et chaque thermomètre de chaque pièce est représenté individuellement, puisqu'ils correspondent chacun à un équipement réseau distinct. On sait que dans le système, les liens entre les routeurs de sortie des bâtiments et le routeur principal sont vieux et potentiellement soumis à des parasites, pouvant entraîner du bruit dans les transmissions. La représentation de ces liens dans le modèle utilise donc un modèle d'erreurs de type *RateErrorModel* (fourni par la bibliothèque NS-3), permettant d'inverser régulièrement un bit dans les communications transmises, correspondant à l'effet du bruit. Le nombre de bits inversés dépend

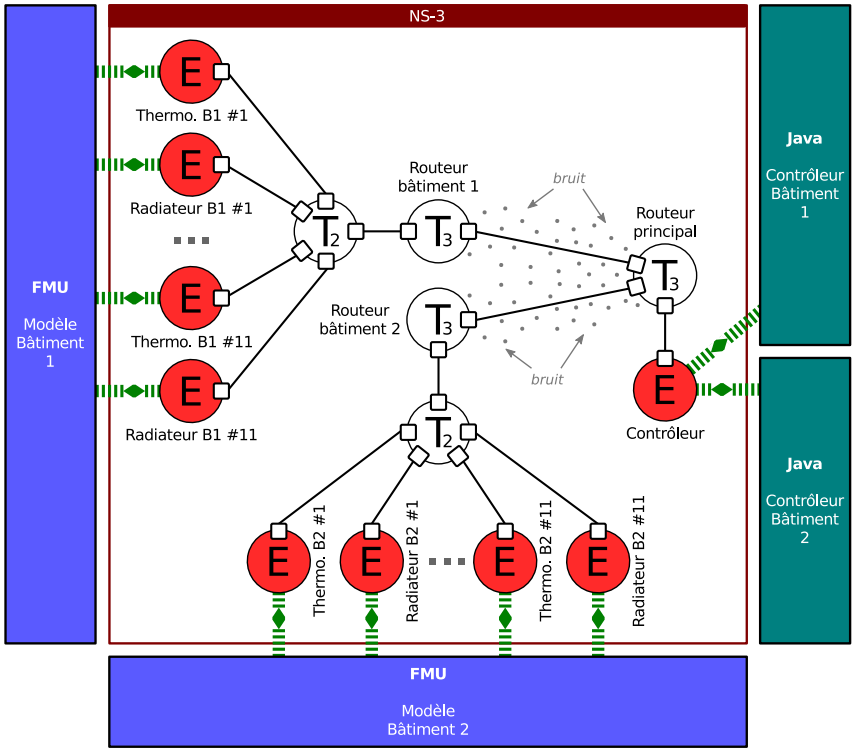



FIGURE 7.21 – Couplages structurels du modèle IP du système de chauffage intelligent de collectivité (avec deux pièces par bâtiment – #1 et #11 – au lieu de onze).

d'un taux configuré dans le modèle (e.g. un bit sur mille inversés).

Concernant les ports structurels DEVS, ils sont au nombre de 132 dans le modèle IP ($x$ représentant le numéro du bâtiment et $y$ le numéro de la pièce dans le bâtiment) :

— Chaque nœud représentant un thermomètre est équipé d'un port d'entrée `BxRyTempIn`.
— Chaque nœud représentant un radiateur est équipé à la fois d'un port d'entrée `BxRyPowIn` et d'un port de sortie `BxRyBlackoutOut`.
— Le nœud représentant le contrôleur est équipé de tous les ports de sortie `BxRyTempOut` et `BxRyPowOut` et de tous les ports d'entrée `BxRyBlackoutIn`, pour $x$ valant 1 et 2 et $y$ variant de 1 à 11.

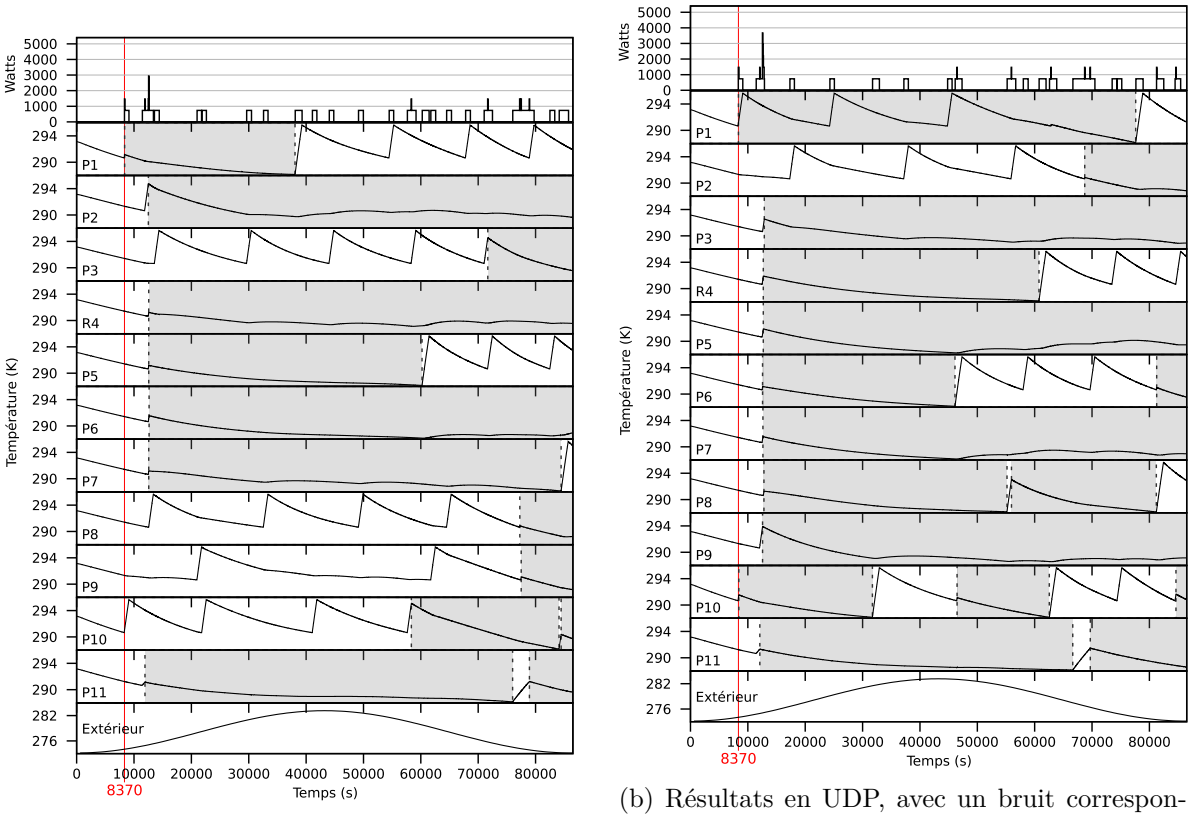
Les ports *Pow* correspondent à des échanges de puissances, les ports *Temp* à des échanges de températures, et les champs *Blackout* à des échanges d'ordre d'effacement. Les températures et les puissances sont transmises sous forme de double, tandis que les ordres sont des booléens (valant « vrai » lorsque le radiateur doit obligatoirement s'éteindre et « faux » lorsqu'il a de nouveau le droit de se rallumer). Les ports d'entrée *In* établissent tous une connexion TCP ou UDP (selon l'expérimentation ci-dessous) avec le port de sortie *Out* correspondant.

### Résultats de simulation

On souhaite faire deux expérimentations : une première avec un réseau sans bruit et en utilisant TCP (protocole fiable) et une seconde avec un bruit important sur les liens et en

utilisant UDP (protocole non fiable). La différence entre les deux configuration devrait illustrer l'impact potentiel de la configuration du réseau IP, sur les résultats de simulation (consommation énergétique des bâtiments).

Pour ces deux expérimentations, on considère que la consigne des radiateurs est fixée à 293,15 K (avec une bande de tolérance de ± 5 K), et que la consigne de température minimum du contrôleur est à 288,15 K. Pour ce dernier, la puissance maximum autorisée pour un bâtiment (le seuil) est réglée à 735 W. Les puissances et les températures sont transmises au contrôleur toutes les 30 secondes. Les paramètres sont tous identiques pour les deux bâtiments et les simulations sont réalisées sur une journée de temps simulé. Les résultat de simulation sont présentés pour un seul bâtiment (cf. couplages du m-agent d'observation), dans la Figure 7.22.




(a) Résultats sans bruit sur le réseau et avec des connexions en TCP (réseau fiable).

(b) Résultats en UDP, avec un bruit correspondant à 1 bit inversé tous les 1000 (réseau non-fiable).

FIGURE 7.22 – Résultats de simulation, pour l'exemple du système de chauffage intelligent de collectivité.

Les résultats sont présentés avec le détail des températures relevées par les thermomètres, pour chaque pièce (P1 à P11), tout au long de la journée. Les surfaces grisées sur ces graphiques indiquent les périodes durant lesquelles le contrôleur a souhaité que le radiateur la pièce soit éteint, quoiqu'il arrive (ce qui ne correspond pas forcément à la vision du radiateur, si celui-ci n'a pas correctement reçu tous les ordres du contrôleur). On remarque l'évolution des températures en dents de scie, caractéristique des radiateurs avec thermostat qui n'ont pas beaucoup d'inertie, et qui font systématiquement évoluer la température autour de leur consigne, en s'allumant et s'éteignant régulièrement. Les graphiques qui sont tout en haut correspondent à la puissance

instantanée consommée, pour tout le bâtiment.

La Figure 7.22a présente les résultats, avec un réseau IP parfaitement fiable, qui ne peut pas altérer les données et qui assure leur transmission. La Figure 7.22b présente les résultats de la même co-simulation, mais avec un réseau IP particulièrement non fiable, puisque le bruit sur les liens entre les routeurs de sortie et le routeur principal inversent un bit sur mille, et que l'utilisation de UDP (sans sommes de contrôle) ne permet pas d'assurer la bonne transmission ni l'intégrité des données. Dans ce second cas, des données ont donc été altérées ou perdues. Par exemple, en comparant les deux résultats, on constate que l'ordre d'effacement pour le radiateur de la pièce P1 au temps 8370 s n'est jamais arrivé dans la version avec réseau non-fiable. En effet, l'évolution de la température dans la pièce après la supposée réception de l'ordre d'extinction, continue de varier en dents de scie sur la Figure 7.22b, indiquant que le radiateur est en fait resté allumé.

Par conséquent, on constate que les températures des pièces divergent rapidement au cours de la journée, entre la version avec réseau fiable et la version avec réseau non fiable. La consommation énergétique du bâtiment est alors modifiée, comme l'indiquent les graphiques de puissance. On constate par exemple que le pic de puissance maximum atteinte dépasse les 3 kW lorsque le réseau n'est pas fiable [13].

Ces résultats suffisent donc à montrer que la configuration du réseau IP peut directement avoir un impact sur la consommation énergétique des bâtiments.

### Conclusion

Cet exemple a permis d'illustrer l'utilisation d'une co-simulation pour évaluer l'impact du réseau IP sur un cas de SCP concret. Les modèles qui sont utilisés pour le représenter sont particulièrement hétérogènes, comme le montre le Tableau 7.23.

<table>
<tr><th>Modèle</th><th>Domaine</th><th>Simulateur</th><th>Plateforme</th><th>Lang.</th><th>Formalisme</th><th>Éch. de temps</th></tr>
<tr><td>Bâtiment 1</td><td rowspan="3">Physique</td><td rowspan="2">FMU (échange de modèles)</td><td rowspan="3">Windows</td><td rowspan="4">Java</td><td rowspan="3">Équationnel</td><td rowspan="4">Seconde</td></tr>
<tr><td>Bâtiment 2</td></tr>
<tr><td>Extérieur</td><td>FMU (co-simulation)</td></tr>
<tr><td>Contrôleur Bât. 1</td><td rowspan="2">Système d'info.</td><td rowspan="2">Ad-hoc</td><td rowspan="3">GNU/Linux</td><td rowspan="2">Automate</td></tr>
<tr><td>Contrôleur Bât. 2</td></tr>
<tr><td>Réseau IP</td><td>Communication</td><td>NS-3</td><td>C++</td><td>Événementiel</td><td>Nanoseconde</td></tr>
</table>

Tableau 7.23 – Hétérogénéité du multi-modèle pour la simulation du système de chauffage intelligent de collectivité.

D'une façon plus générale, cet exemple a permis de prouver expérimentalement que la représentation des réseaux de communication (IP) est importante, et peut entraîner des variations dans les résultats de simulation des SCP.

13. Attention, cette valeur dépend également de la graine utilisée par NS-3 pour calculer les valeurs stochastiques qu'il utilise dans le modèle. Il serait donc nécessaire de faire une moyenne des puissances obtenues, en faisant évoluer la graine sur plusieurs exécutions, si les résultats étaient réellement importants.

### 7.5.3 Intégration de modèles industriels

#### Introduction

Ce second exemple correspond à une application industrielle, réalisée par EDF R&D, avec l'aide de Inria. Le système modélisé est un réseau électrique intelligent, qui intègre des modèles des trois principaux domaines d'expertise des SCP.

Le modèle de réseau électrique a été réalisé par un ingénieur EDF R&D, qui a maintenu confidentiel le détail de son implémentation. La personne qui a réalisé le multi-modèle MECSYCO pour relier le modèle électrique au modèle IP ne connaissait ni les détails du modèle de réseau électrique, ni le détail du modèle IP. Cet exemple permet donc d'illustrer la capacité de notre solution à structurellement coupler des boîtes noires industrielles à des modèles IP.

De plus, le système modélisé par cet exemple a également fait l'objet d'une expérience sur le terrain, afin de pouvoir comparer les résultats de simulation directement avec les observations faites sur le système lui-même. Cette expérience en grandeur réelle a eu lieu sur le site de EDF Lab Les Renardières, à Écuelles (aperçu du site en Figure 7.24).

FIGURE 7.24 – Aperçu de l'expérience grandeur nature réalisée à EDF Lab Les Renardières.

Cette application a été en partie présentée dans [Vaubourg et al., 2015].

#### Présentation du système

Le système à modéliser est décrit par la Figure 7.25.

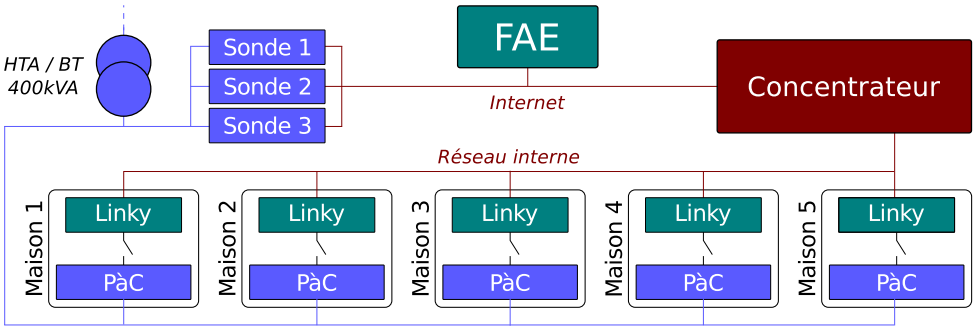



FIGURE 7.25 – Système de réseau électrique intelligent à modéliser.

Cinq maisons utilisent un même réseau électrique, et sont chacunes équipées d'un compteur électrique intelligent (Linky), lui-même relié à une pompe à chaleur (PàC). Les compteurs Linky sont chargés de gérer l'allumage et l'extinction des pompes à chaleur, en fonction de l'heure de la journée. Pour ce faire, ils suivent un planning qui leur a été envoyé par un centre décisionnel (FAE – Fonction d'Effacement Avancée) distant et centralisé. Ce dernier a la capacité d'envoyer de nouveaux plannings à suivre aux compteurs, à n'importe quel moment de la journée. Trois sondes de mesure placées sur le réseau électrique, ont pour fonction de lui envoyer régulièrement des informations (tensions électriques) sur la charge de ce dernier.

Tous les compteurs sont reliés à un même réseau IP interne, qui est lui-même relié à un concentrateur. Le concentrateur est également relié à Internet, tout comme la FAE et les sondes de mesure. La FAE utilise cette connexion Internet pour envoyer les plannings de tous les compteurs, rassemblés dans un même fichier XML, au concentrateur. Ce dernier a ensuite la responsabilité d'extraire les plannings individuels et de les transmettre aux compteurs concernés, via le réseau IP interne. Lorsque les compteurs reçoivent leur programme, ils doivent envoyer un message d'acquittement (au niveau applicatif) au concentrateur, qui doit ensuite les transmettre à la FAE. Les sondes de mesure transmettent leurs informations à la FAE également via Internet.

Au niveau du détail de la topologie réseau, on sait que le réseau interne est un réseau Ethernet commuté classique, et que Internet peut être représenté par un simple commutateur [14]. Par contre, on sait que les communications sur le réseau interne doivent se faire en TCP sur de l'IPv6, tandis que les communications sur Internet doivent se faire en IPv4 sur de l'UDP.

Ce cahier des charges, conçu avec EDF R&D, permet de mettre en place un scénario d'effacement cascado-cyclique, et donc de tester différents algorithmes de délestage électrique, en testant différents plannings à transmettre aux compteurs.

#### Multi-modèle utilisé

Le multi-modèle à réaliser est composé de huit modèles distincts :

— 1 modèle de réseau électrique (FMU pour Windows produite avec Dymola, par EDF R&D) ;
— 5 modèles représentant la couche applicative de chaque Linky (FMU pour Windows produites avec Dymola, par EDF R&D) ;
— 1 modèle de système décisionnel représentant la FAE (programme java ad-hoc, par Inria) ;
— 1 modèle de réseaux IP (avec NS-3 pour GNU/Linux, par Inria).

Les interactions souhaitées entre les différents modèles sont illustrées dans la Figure 7.26 (pour des raisons de lisibilité, seules deux maisons sont visibles). On y retrouve les interactions décrites au niveau du système, dans la partie précédente. Le modèle de réseau électrique intègre directement les sondes de mesure.

Le multi-modèle MECSYCO utilisé pour la simulation est disponible dans la Figure 7.27. Puisque les différents modèles reliés au modèle de réseau IP représentent des comportements applicatifs, les connexions avec NS-3 correspondront à des couplages structurels au niveau du modèle. Les artéfacts de couplage entre le modèle NS-3 et les autres modèles utiliseront une opération de transformation du temps, puisque NS-3 utilise une représentation du temps à la nanoseconde, tandis que les autres modèles utilisent une représentation à la seconde.

14. À noter que cette représentation simpliste du réseau Internet correspondrait en fait à un réseau pour lequel tous les systèmes autonomes ont réussi à se rejoindre sur un même point de *peering* mondial.

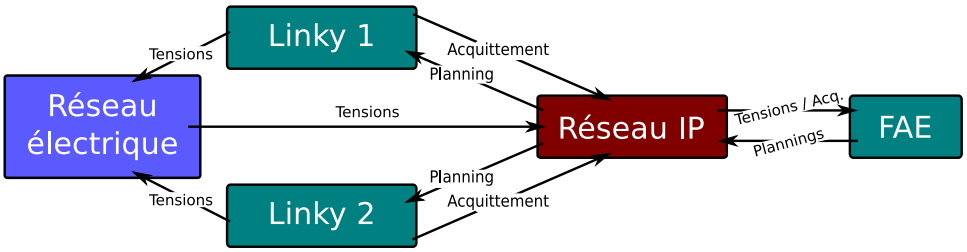



FIGURE 7.26 – Vue intuitive du multi-modèle pour la simulation du réseau électrique intelligent (avec deux maisons au lieu de cinq).

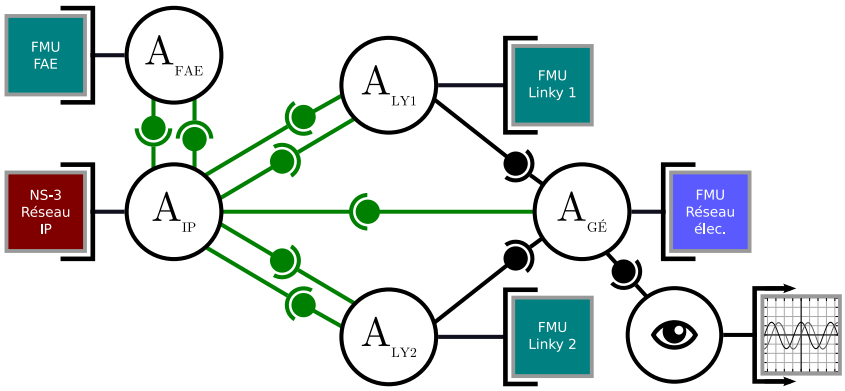



FIGURE 7.27 – Multi-modèle MECSYCO pour la simulation du réseau électrique intelligent (avec deux maisons au lieu de cinq).

### Modèle IP utilisé

La topologie IP correspondant au système qui a été décrit est visible dans le schéma de la Figure 7.28.

On y retrouve les cinq compteurs, ainsi que les trois sondes et la FAE, en tant que nœuds finaux. Les commutateurs des deux réseaux sont reliés à un dixième nœud final, qui correspond au concentrateur. Celui-ci n'est pas un nœud de transit, parce qu'il ne se contente pas de transmettre des données d'une interface à l'autre, mais produit ses propres données, en créant les programmes individuels. Son comportement applicatif a été modélisé en développant directement un sous-modèle NS-3 dédié à cette application industrielle.

### Résultats de simulation

Le multi-modèle a été exécuté dans les locaux de Inria, sur une période simulée de 24h, de façon distribuée en utilisant deux machines. Les résultats ont été transmis à EDF R&D, qui les a comparé à ceux obtenus avec l'expérience grandeur nature, sans trouver de différence majeure.

Avec la participation de EIFER (*European Institute for Energy Research*), EDF R&D a produit une interface dynamique permettant de visualiser l'intégralité du déroulé de la simulation sur les 24h (cf. aperçu en Figure 7.29). Cette interface permet également de visualiser la trans-

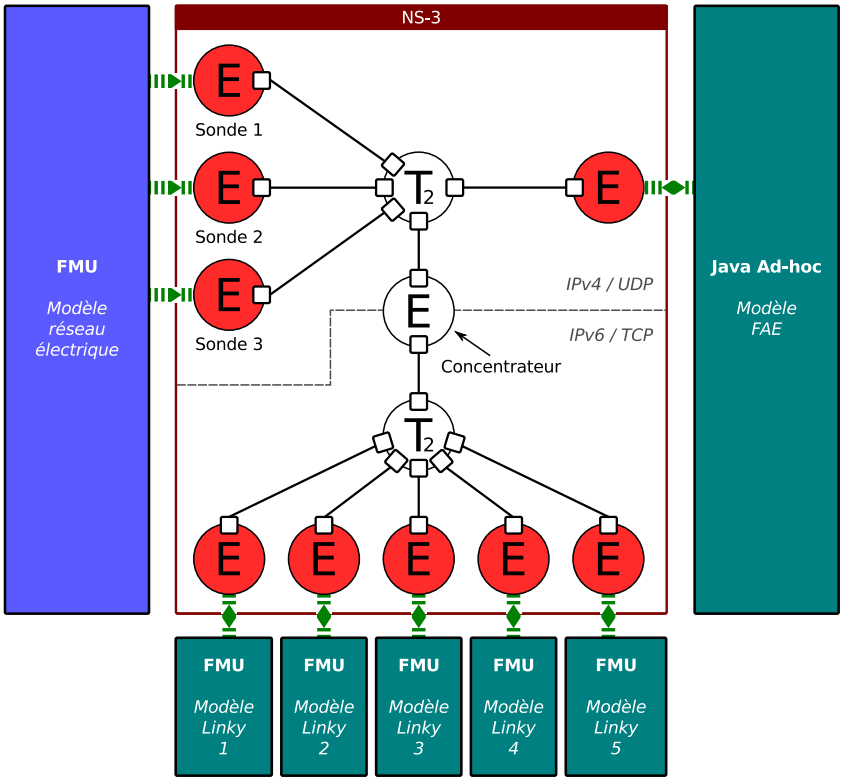



FIGURE 7.28 – Couplages structurels du modèle IP du réseau électrique intelligent.

mission des paquets IP sur le réseau, grâce à l'intégration de fichiers de traces (*pcap*) directement produits par NS-3.

#### Conclusion

Cet exemple a permis de présenter un cas d'utilisation industrielle de l'intégration de modèles IP à un multi-modèle MECSYCO, permettant de simuler un réseau électrique intelligent complet. Le multi-modèle a pu être réalisé sans connaître le contenu des modèles, et donc en intégrant des modèles industriels similaires à des boîtes noires. La livraison des résultats de simulation a permis de créer un outil de visualisation post-mortem complet.

Comme le montre le Tableau 7.30, ce cas pratique est également un exemple intéressant d'intégration de différentes formes d'hétérogénéité.

### 7.5.4 Modularité des simulations

#### Introduction

Ce dernier exemple a pour objectif d'illustrer la modularité de notre solution, en représentant un système de chauffage intelligent individuel, contrôlable par Internet. Ce cas d'utilisation nécessite de réunir des modèles des trois principaux domaines d'expertise des SCP.


FIGURE 7.29 – Interface de visualisation des données de simulation, réalisée par l'institut EIFER à la demande de EDF R&D.

| Modèle | Domaine | Simulateur | Plateforme | Langage | Formalisme | Échelle de temps |
|---|---|---|---|---|---|---|
| Réseau électrique | Physique | FMU | Windows | Java | Équationnel | Seconde |
| Linky 1-5 | S.I. | | | | | |
| FAE | | Ad-hoc | | | Automate | |
| Réseau IP | Com. | NS-3 | GNU/Linux | C++ | Événementiel | Nanoseconde |

TABLEAU 7.30 – Hétérogénéité du multi-modèle pour la simulation du réseau électrique intelligent.

Il est à la fois un exemple :

— de modèle IP mélangeant des couplages structurels et spatiaux ;
— de comparaison entre modèles similaires issus de différentes bibliothèques IP ;
— de remplacement plug-n-play d'une technologie IP par une autre, grâce aux couplages spatiaux et à MECSYCO ;
— et d'estimation du surcoût en temps d'exécution, de l'utilisation de couplages spatiaux.

Le multi-modélisation de cet exemple se fera progressivement en trois étapes :

1. multi-modélisation du système en utilisant un seul modèle pour modéliser le réseau IP ;
2. éclatement du modèle IP précédent, pour simuler un fragment de la topologie en utilisant un modèle issu d'un autre simulateur IP ;
3. second éclatement du modèle IP, pour pouvoir tester rapidement différentes configurations du système.

Cet exemple a été présenté dans [Vaubourg et al., 2016].

### Présentation du système

Le système qu'on souhaite modéliser est schématisé dans la Figure 7.31. Il s'agit d'un système de chauffage intelligent individuel, qui gère la température dans deux chambres adjacentes d'une maison.

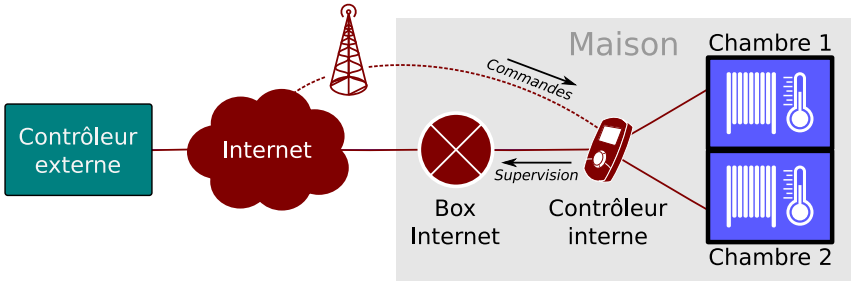



FIGURE 7.31 – Système de chauffage intelligent, pour une seule maison.

Chacune des deux chambres est équipée d'un radiateur, qui dispose lui-même d'un thermomètre intégré. Ces équipements sont reliés en IP à un contrôleur interne, qui centralise les systèmes de chauffage de la maison. Ce contrôleur interne est lui-même connecté à la box Internet de la maison, elle-même reliée à Internet, tout comme un serveur faisant office de contrôleur externe.

Le scénario est le suivant. Le propriétaire de la maison a la possibilité de régler une consigne de température, pour les deux chambres, depuis une interface web accessible via Internet. Ces consignes sont prises en compte par le contrôleur externe, qui les transmet au contrôleur interne de la maison, à chaque changement. Le contrôleur interne les transmet ensuite aux radiateurs, qui seront chargés de stabiliser la température des chambres en fonction de la consigne ainsi enregistrée. Afin de pouvoir afficher l'évolution de la température des chambres sur l'interface web du propriétaire, le contrôleur interne reçoit régulièrement les températures mesurées par les thermomètres des radiateurs, qu'il transmet via Internet au contrôleur externe.

Au niveau de la topologie IP, tous les équipements décrits ci-avant sont reliés entre eux en filaire et en TCP. Le contrôleur interne peut accéder à Internet via la box de la maison, mais est également équipé d'une carte SIM, et dispose donc de sa propre connexion 4G. Cette connexion a pour premier objectif de permettre au contrôleur externe de le contacter pour lui envoyer les consignes (commandes), sans risquer d'être bloqué par le pare-feu de la box de la maison. Par contre, les températures (données de supervision) reçues par le contrôleur interne sont envoyées au contrôleur externe via Internet en utilisant la box de la maison. Dans ce sens, un pare-feu domestique ne pose aucun problème, et les quotas du forfait 4G sont ainsi préservés. Si l'accès à Internet de la maison n'est plus fonctionnel, les données de supervision pourront également être envoyées via la connexion 4G, en tant que lien de secours.

Le multi-modèle qui est présenté dans la prochaine partie évoluera au fur et à mesure de l'évolution de l'expérience.

### Multi-modèle initial

Sans compter les modèles IP, le multi-modèle à réaliser est composée de cinq modèles :

— 2 modèles thermiques représentant les radiateurs équipés de thermomètres (FMU pour Windows produites avec OpenModelica) ;
— 1 modèle thermique représentant le mur qui sépare les deux chambres, modélisant les échanges de températures qui ont lieu entre les chambres (FMU pour Windows produite avec OpenModelica) ;
— 1 modèle thermique représentant l'évolution de la température extérieure en fonction de l'heure de la journée, et modélisant l'influence de la température extérieure sur la température des chambres (FMU pour Windows produite avec OpenModelica) ;
— 1 modèle de contrôleur extérieur (programme java ad-hoc).

Les interactions souhaitées entre les différents modèles sont illustrées dans la Figure 7.32 (en considérant dans ce schéma qu'il n'y a qu'un seul modèle pour représenter le réseau IP). On y retrouve les interactions décrites au niveau du système, dans la partie précédente, avec en plus les échanges de températures entre les modèles thermiques des chambres.

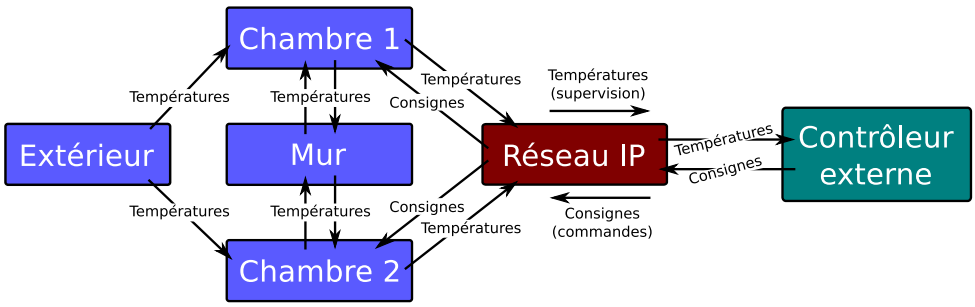



FIGURE 7.32 – Vue intuitive du multi-modèle pour la simulation de système de chauffage intelligent individuel.

Dans un premier temps, on représente le réseau IP en n'utilisant qu'un seul modèle.

### Modélisation du réseau IP uniquement avec NS-3

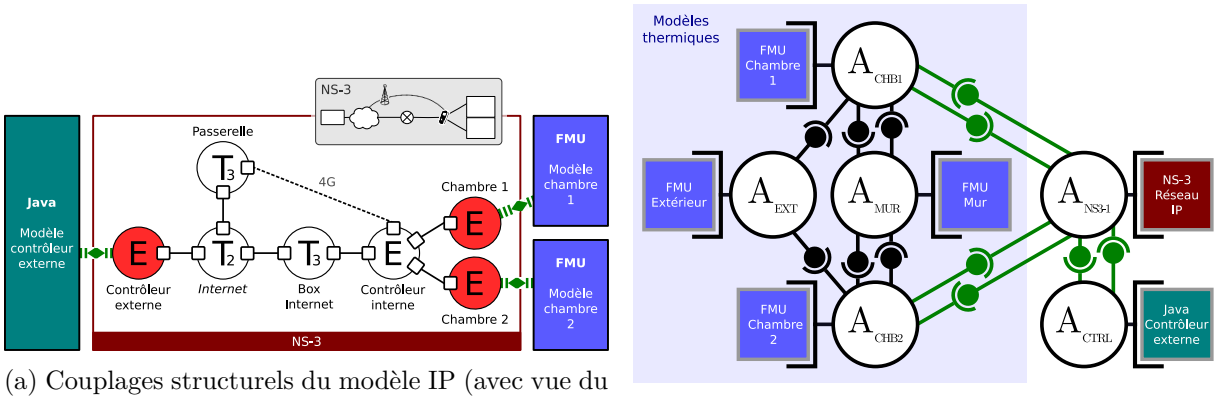
Dans cette première version, on décide de modéliser l'intégralité du modèle IP avec NS-3, comme dans les exemples précédents.

Le modèle IP utilise uniquement des couplages structurels bidirectionnels, et correspond au schéma de la Figure 7.33a (une vue schématique du système est rappelée en haut à droite). Internet est de nouveau modélisé comme un commutateur et le contrôleur interne agit comme un serveur mandataire. Son comportement est décrit directement via l'implémentation d'un sous-modèle d'application NS-3.

Le multi-modèle MECSYCO correspondant est disponible dans la Figure 7.33b. Par la suite, afin d'améliorer la lecture des multi-modèles MECSYCO, les modèles thermiques couplés qui sont regroupés dans un rectangle bleu seront abstraits, en les remplaçant par ce rectangle seul.

Après exécution de cette co-simulation, distribuée sur deux machines (une machine GNU/-Linux avec les modèles IP et le contrôleur extérieur, et une machine Windows avec les modèles thermiques), nous nous demandons si le sous-modèle de 4G fourni dans la bibliothèque de OMNeT++/INET pourrait être performant [15] que celui qui est fourni par NS-3.

15. On considère qu'un modèle est plus performant qu'un autre, si les deux modèles représentent la même chose,




(a) Couplages structurels du modèle IP (avec vue du système en haut à droite).

(b) Multi-modèle MECSYCO correspondant.

FIGURE 7.33 – Couplages du modèle IP pour la simulation du système de chauffage intelligent individuel, en utilisant uniquement NS-3.

Pour autant, nous ne souhaitons pas réécrire tout le modèle IP de la co-simulation avec OMNeT++/INET, notamment parce que le modèle applicatif du contrôleur interne a été écrit spécifiquement pour NS-3. Nous décidons donc d'utiliser des couplages spatiaux, pour confier à OMNeT++/INET la charge de modéliser et simuler uniquement la partie 4G du réseau.

### Comparaison des modèles entre simulateurs IP

Le modèle IP NS-3 produit dans la partie précédente est modifié, afin de remplacer l'utilisation du sous-modèle 4G par une paire de ports spatiaux. La modification est visible dans la Figure 7.34a. On peut également voir sur cette figure le modèle OMNeT++/INET qui a été créé pour modéliser uniquement le lien 4G, avec son propre sous-modèle. L'emplacement des ports et l'utilisation des appendices correspondent aux recommandations données dans les chapitres précédents.

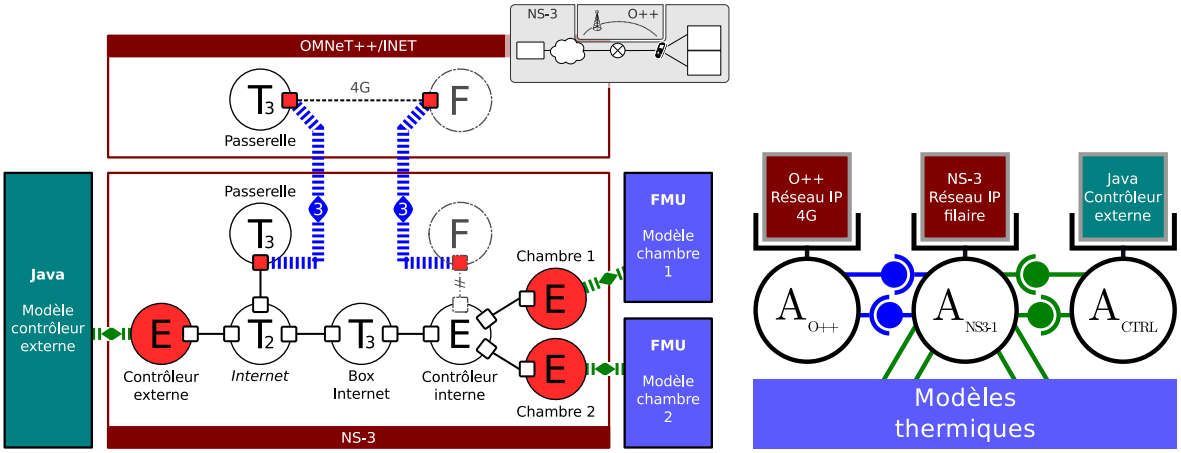
Le nouveau multi-modèle MECSYCO à utiliser est proposé dans la Figure 7.34b. Cette nouvelle version correspond simplement à l'ajout d'un m-agent OMNeT++/INET avec son modèle, et d'une paire d'artéfacts de couplage reliés au m-agent NS-3 (le reste du multi-modèle n'a pas été modifié).

Afin de comparer les performances de l'ancienne et de la nouvelle version, nous décidons de faire passer l'exemple à l'échelle, en reliant plusieurs maisons à Internet. Nous considérons que chaque maison est identique, est reliée au « même Internet », et dispose de sa propre passerelle 4G (cf. illustration de la Figure 7.35, avec trois maisons).

À chaque ajout de maison, nous faisons les modifications suivantes :

— la représentation du réseau IP complet de la maison est ajoutée à l'unique modèle NS-3, et la nouvelle box Internet est reliée au commutateur *Internet* ;
— les deux nouveaux couplages structurels de la nouvelle maison sont reliés à de nouvelles instances des modèles thermiques, dans le multi-modèle ;

mais que l'un permet à la co-simulation de se terminer plus rapidement que l'autre. Une meilleure performance peut par contre parfois engendrer des résultats moins précis.




(a) Couplages structurels du modèle IP (avec vue du système découpé en haut à droite).

(b) Multi-modèle MECSYCO correspondant.

FIGURE 7.34 – Couplages du modèle IP pour la simulation du système de chauffage intelligent individuel, en utilisant OMNeT++/INET pour la partie 4G.

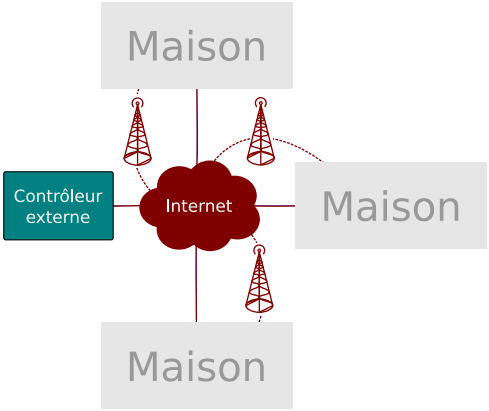



FIGURE 7.35 – Passage à l'échelle de l'exemple de système de chauffage intelligent individuel, avec trois maisons.

— dans la version avec 4G séparée, le nouveau couplage spatial de la nouvelle maison est relié à une nouvelle instance du modèle 4G de OMNeT++/INET, dans le multi-modèle ;
— toutes les nouvelles instances du modèle 4G de OMNeT++/INET font également l'objet d'un ajout de port spatial, au niveau du commutateur *Internet* dans le modèle NS-3.

Nous exécutons des tests de multi-modèle, intégrants jusqu'à 20 maisons. Pour 20 maisons, le modèle NS-3 utilise donc 41 couplages structurels. Dans la version avec 4G séparée, il utilise en plus 40 couplages spatiaux (avec 122 nœuds instanciés).

Les tests sont réalisés avec trois versions du multi-modèle :

1. utilisation de NS-3 seul pour modéliser l'intégralité des réseaux IP (correspondant à la version présentée dans la partie précédente) ;
2. utilisation de NS-3 pour modéliser l'intégralité des réseaux IP sauf les liens 4G, et d'ins-

tances de OMNeT++/INET pour modéliser ces derniers (correspondant à la version présentée dans cette partie) ;

3. utilisation d'une instance de NS-3 pour modéliser l'intégralité des réseaux IP sauf les liens 4G, et d'autres instances de NS-3 pour modéliser ces derniers (correspondant donc à la version précédente, mais en remplaçant le choix OMNeT++/INET par NS-3).

La troisième version est réalisée afin de permettre de comparer les performances des modèles 4G des bibliothèques NS-3 et OMNeT++/INET, sans que le surcoût en temps d'exécution des couplages spatiaux ne fausse les résultats. La comparaison de la première version et de la troisième version permettra également d'avoir une estimation de ce surcoût. Les résultats sont disponibles dans la Figure 7.36, pour une minute de temps simulé. La version de NS-3 au moment des tests est la 3.24, la version de OMNeT++ est la 4.6 et la version de INET est la 2.6.

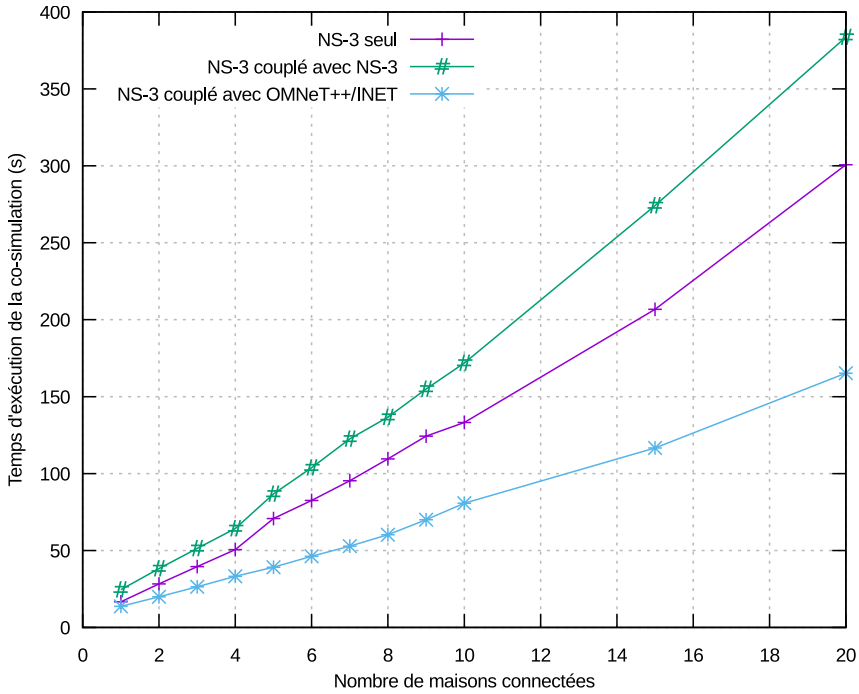



FIGURE 7.36 – Comparaison des modèles de 4G des bibliothèques de NS-3 et OMNeT++/INET, et estimation du surcoût induit par les couplages spatiaux.

On peut tirer de ce graphique trois constats :

1. **Performances générales :** Le premier constat est que le temps d'exécution de la co-simulation dépasse le temps simulé, pour toutes les versions, avant que les 10 maisons connectées ne soient atteintes. Pour analyser ce résultat, il convient de savoir que les protocoles sans fil sont par nature très verbeux, et que par conséquent leurs modèles génèrent énormément d'événements internes. Par exemple, pour une simulation du même réseau avec 20 maisons, sans les couplages spatiaux (les ports sont remplacés par des applications NS-3 triviales), NS-3 produit 6 936 511 événements internes pour une minute de simulation. Au niveau de la charge à l'exécution, la version avec 20 maisons nécessite la participation de pas moins de 102 simulateurs à la même co-simulation (principalement à cause des modèles thermiques séparés).

2. **Surcoût des couplages spatiaux :** Le second constat, lorsqu'on compare la courbe de NS-3 seul avec celle de NS-3 couplé avec NS-3, c'est que l'éclatement d'un modèle en deux et l'ajout de couplages spatiaux peuvent coûter relativement cher en temps d'exécution (un peu plus de 25% en plus dans cet exemple, quand on atteint les 20 maisons). Dans ce cas, le gain de temps apporté par la parallélisation des calculs des événements, n'a pas compensé le temps consommé par MECSYCO pour la synchronisation des modèles. Ce constat variera selon si les modèles représentent ou des portions de réseaux suffisamment indépendantes les unes des autres, pour profiter plus ou moins de la distribution des calculs.
3. **Comparaison des modèles de 4G :** Le troisième constat est que malgré le surcoût induit par les couplages spatiaux, confier la partie 4G de la simulation à OMNeT++/INET plutôt que de laisser NS-3 tout modéliser seul, permet tout de même de quasiment diviser les temps d'exécution par deux. Attention toutefois, ce résultat est intéressant pour prouver qu'on peut tirer un bénéfice des couplages spatiaux, y compris au niveau des performances, mais il ne peut pas être utilisé pour prouver que le modèle de OMNeT++/INET est réellement plus performant que celui de NS-3. Il faudrait pour cela s'assurer que les deux modèles offrent des résultats aussi précis et pertinents, et qu'ils sont configurés de la façon la plus similaire possible.

#### Comparaison des technologies pour un même simulateur IP

Cette dernière version du multi-modèle a pour objectif d'illustrer un cas d'utilisation des couplages spatiaux, permettant de tester rapidement différentes configurations pour une partie d'un réseau IP.

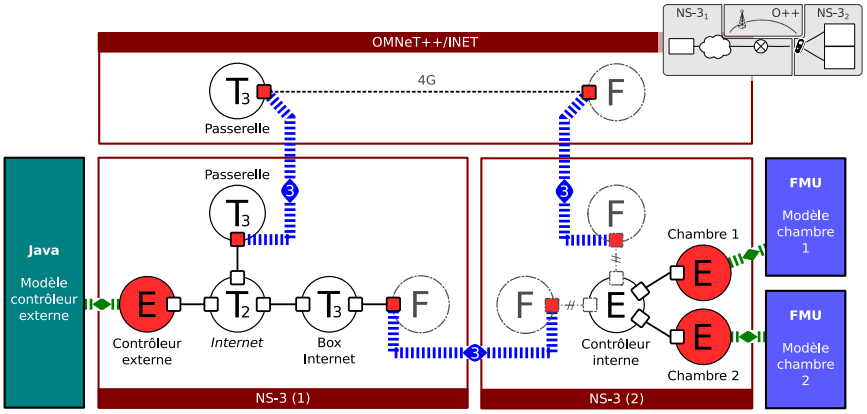
Les couplages IP sont visibles dans la Figure 7.37, avec le multi-modèle MECSYCO. Grâce à cette configuration, il est par exemple possible de passer d'un réseau filaire à un réseau Wifi dans la maison, uniquement en exécutant un m-agent différent au moment de l'exécution de la co-simulation avec MECSYCO. Aucune modification du multi-modèle ne sera nécessaire, y compris si la version Wifi était exécutée par OMNeT++/INET plutôt que NS-3.

#### Conclusion

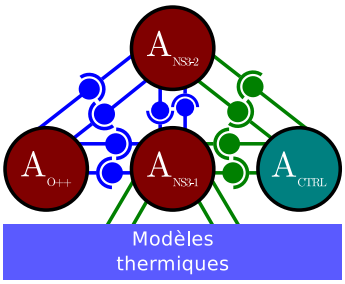
Cet exemple a permis de présenter un cas d'utilisation particulièrement riche en couplages structurels et spatiaux, tout en intégrant de nombreuses formes d'hétérogénéité (cf. Tableau 7.38).

Nous avons notamment pu voir comment nos travaux pouvaient être utilisés pour modéliser un réseau IP complet, sans devoir choisir un simulateur IP en particulier et donc sans être restreint à une seule bibliothèque de modèles. Ainsi, nous avons réussi par exemple à utiliser NS-3 pour utiliser le modèle de contrôleur interne qui était écrit pour lui, en même temps que OMNeT++/INET pour son modèle de lien 4G qui semble plus performant. Et ce alors que NS-3 et OMNeT++/INET sont initialement incompatibles entre eux et ne permettent pas de participer à des co-simulations.

Nous avons également vu comment la modularité de notre solution permettait de passer d'une technologie à l'autre pour une partie du réseau IP, grâce aux couplages spatiaux et à la méthode plug-n-play de changement de modèles permise par MECSYCO.




(a) Couplages structurels du modèle IP (avec vue du système découpé en haut à droite).




(b) Multi-modèle MECSYCO correspondant (les modèles et leurs artéfacts ne sont représentés, pour des raisons de lisibilité).

FIGURE 7.37 – Couplages du modèle IP pour la simulation du système de chauffage intelligent individuel, en utilisant OMNeT++/INET pour la partie 4G et en permettant de passer rapidement d'un modèle à l'autre pour représenter le réseau IP interne.

| Modèle | Domaine | Simulateur | Plateforme | Lang. | Formalisme | Éch. de temps |
|---|---|---|---|---|---|---|
| Chambre 1 | Physique | FMU | Windows | Java | Équationnel | Seconde |
| Chambre 2 | | | | | | |
| Extérieur | | | | | | |
| Mur | | | | | | |
| Contrôleur ext. | S.I. | Ad-hoc | GNU/Linux | | Automate | |
| Réseau IP filaire | Com. | NS-3 | | C++ | Événementiel | Nanoseconde |
| Réseau IP 4G | | OMNeT++/INET | | | | |

TABLEAU 7.38 – Hétérogénéité du multi-modèle pour la simulation du système de chauffage intelligent individuel (seconde version).

### 7.5.5 Conclusion

Au travers de ces trois exemples concrets de co-simulation de SCP complets (i.e. réunissant les trois principaux domaines d'expertise, que sont le domaine physique, les réseaux de communication et les systèmes d'information), nous avons pu illustrer trois principales caractéristiques relatives à nos travaux :

1. l'importance de la représentation des réseaux de communication dans la représentation des SCP ;
2. la possibilité d'intégrer des modèles industriels existants et disponibles sous forme de boîtes noires ;
3. la modularité de la solution finale, qui permet notamment de changer de façon de représenter le réseau IP de différentes façons, sans avoir à modifier le reste de la co-simulation.

Les cas d'utilisation qui ont fait l'objet d'exploitation au niveau industriel permettent de plus de confirmer que la réalisation des concepts et outils issus de nos travaux, trouve un intérêt pratique directement au niveau opérationnel.

## 7.6 Conclusion

Ce chapitre a permis de présenter des exemples d'application des contributions qui ont été proposées dans ce manuscrit.

Ainsi, au travers de la description de la démarche attendue lorsqu'on souhaite créer une co-simulation de SCP complet, nous avons vu de quelle façon il était possible d'utiliser nos travaux pour mener à bien une expérience complète, en partenariat avec un commanditaire. Ce dernier est alors libre de confier la création de certains de ses modèles à des experts du domaine, tout en confiant la partie conception du multi-modèle et des modèles IP, à un autre prestataire.

L'intégration des simulateurs IP NS-3 et OMNeT++/INET a permis de démontrer concrètement que les concepts et outils proposés dans les autre chapitres, sont utilisables et directement transposables au niveau du code. Ainsi, en suivant nos propres recommandations, nous avons pu entièrement intégrer ces deux simulateurs IP à des co-simulations hybrides, tout en leur permettant de communiquer pour représenter différentes parties d'une même topologie. Les extraits de code qui ont été présentés, permettront à l'avenir d'aider à trouver des solutions pour les contraintes purement techniques, lors de l'intégration de nouveaux simulateurs IP. En utilisant ce travail, nous avons également prouvé expérimentalement que l'introduction de couplages spatiaux au milieu de la représentation d'une topologie IP, ne faussait pas les résultats de simulation.

Enfin, nous avons présentés différents exemples de simulation de SCP complets, qui ont été réalisées en utilisant nos travaux, et qui ont pour certains d'entre eux permis à des industriels de mener à bien des expériences.

# Chapitre 8

# Conclusion

**Sommaire**



## 8.1 Objectifs et problématiques

Dans ce travail de thèse, nous nous sommes intéressés à l'étude des modèles IP dans le cadre de la co-simulation.

En nous situant dans le domaine des systèmes cyber-physiques (SCP), nous nous sommes demandés comment il était possible d'intégrer des modèles IP à des co-simulations, qui seraient alors capables de représenter à la fois :

— des systèmes d'information,
— des systèmes physiques,
— et des réseaux de communication (IP).

Nous avons souhaité être en capacité de réutiliser des modèles existants, disponibles dans les communautés liées à ces trois domaines d'expertise.

L'intégration de modèles aussi hétérogènes à un même multi-modèle nous a alors amené à la problématique suivante :

— Comment intégrer l'hétérogénéité d'un multi-modèle hybride, au niveau des formalismes, de la représentation des données de simulation et des implémentations logicielles utilisées ?

En nous intéressant aux possibilités offertes par l'utilisation de la multi-modélisation dans le cadre de la modélisation IP, et aux usages de la co-simulation avec des modèles IP référencés dans la littérature, nous en sommes arrivés à ajouter deux autres problématiques :

— Comment intégrer des modèles IP de façon à ce qu'ils soient en capacité de représenter le transport de données applicatives, produites par des modèles issus d'autres domaines d'expertise ?
— Comment intégrer des modèles IP de façon à ce qu'ils soient en capacité de se compléter, pour représenter ensemble une topologie de réseau IP complète ?

Enfin, nous nous sommes imposés comme contrainte que le multi-modèle créé soit modulaire, afin de pouvoir rapidement échanger les modèles et expérimenter aisément. Nous avons également étudié nos solutions de façon à ce qu'elles permettent d'éviter au maximum d'avoir à modifier le code des simulateurs eux-mêmes.

## 8.2 Contributions

Afin de répondre aux problématiques sus-citées, nous avons choisi de nous appuyer sur le formalisme DEVS, et en particulier sur les travaux réalisés dans le cadre de la plateforme de co-simulation MECSYCO.

Grâce à son usage du formalisme DEVS, cette plateforme permet de créer des multi-modèles hybrides et ainsi d'intégrer toutes les formes d'hétérogénéité qu'on peut trouver dans un multi-modèle de SCP. Cependant, l'utilisation du formalisme DEVS pour la co-simulation, nécessite d'être en capacité de wrapper en DEVS tous les modèles non DEVS, qui sont à intégrer au multi-modèle. Il n'existe pas actuellement, à notre connaissance, de solution permettant d'intégrer en DEVS des modèles IP et leurs simulateurs.

En réponse à ce besoin, nous avons proposé un **cadre général, permettant de wrapper les modèles IP et leurs simulateurs de façon non ad-hoc**, composé de :

1. Une structuration des différents niveaux de problèmes pour l'intégration de modèles IP dans une co-simulation, permettant de délimiter les objectifs et contraintes du wrapping.
2. Une stratégie de wrapping DEVS de modèles IP et leurs simulateurs.

Grâce au wrapping DEVS et aux différents types de couplages IP que nous avons proposés (spatial et structurel), nos contributions permettent notamment de :

— **simuler le transfert de données applicatives** générées par des modèles non IP, sur un réseau IP ;
— **créer un multi-modèle IP à partir du schéma d'une topologie IP** à représenter en plusieurs parties ;
— **distribuer l'exécution de modèles IP**, pour des simulateurs IP qui n'ont pas été conçus pour le faire ;
— **interconnecter des sous-modèles IP issus de différents bibliothèques IP**, proposées par des simulateurs IP incompatibles entre eux.

Ces contributions permettent par conséquent de **simuler des SCP complets**, en réutilisant des modèles existants issus des trois principaux domaines d'expertise : systèmes d'informations, réseaux de communication (IP), et systèmes physiques.

Nous avons étudié cette démarche **du niveau théorique au niveau pratique**, en wrappant deux simulateurs IP réputés dans le monde scientifique : NS-3 et OMNeT++/INET. Ces intégrations ont donné lieu à la **mise à disposition de deux bibliothèques C++**, qui peuvent être directement utilisées, pour intégrer des modèles IP à un multi-modèle DEVS exécuté par MECSYCO. Nous avons également proposé une preuve de concept, permettant de prouver expérimentalement que les couplages spatiaux entre modèles IP (permettant de faire communiquer ensemble deux simulateurs IP différents) n'introduisaient pas de biais dans les résultats de simulation.

Nous avons fait la démonstration de l'**utilité de ce travail au niveau industriel**, grâce à un partenariat avec EDF R&D. Nous avons utilisé les implémentations issues de nos travaux, pour répondre à des besoins de multi-modélisation de SCP dans le cadre des réseaux électriques intelligents, et permettre ainsi d'obtenir des résultats de simulation exploitables. Nous avons notamment étudié (1) l'impact que peut avoir un réseau IP dans le fonctionnement d'un système de chauffage intelligent, (2) comment des modèles industriels de type « boîte noire » peuvent être utilisés avec notre solution et (3) comment **la modularité de notre solution** et les possibilités d'interconnexion entre deux simulateurs IP différents, peuvent permettre d'obtenir de meilleurs temps de simulation.

Enfin, nous avons mis à profit notre expérience avec EDF R&D, pour **proposer une méthodologie**, permettant de comprendre la démarche attendue lorsqu'on souhaite co-simuler un SCP avec des réseaux IP et des modèles déjà existants, notamment en illustrant les interactions qui sont nécessaires entre le maître d'ouvrage et le maître d'œuvre.

Le développement des artéfacts de modèle pour NS-3 et OMNeT++/INET, ainsi que la réécriture de MECSYCO en C++ et la participation à l'évolution de sa version Java, ont permis de contribuer à quatre dépôts de propriété intellectuelle de type APP (dont le code est actuellement ou sera à terme distribué sous licence libre AGPL 3.0).

## 8.3 Perspectives

Les avancées proposées par nos travaux permettent désormais de mener de nouvelles études, aussi bien destinées :

1. à obtenir plus de résultats concrets sur des scénarios complexes de SCP, dans divers domaines (e.g. réseaux électriques intelligents, villes intelligentes, etc) ;
2. qu'à obtenir plus d'éléments de comparaison, permettant d'évaluer les performances et la précision des différentes bibliothèques IP existantes.

De plus, outre la nécessité de confronter nos travaux à une plus grande diversité de simulateurs IP, en les wrappant en DEVS, nos travaux ne permettent pas actuellement de répondre à un certain nombre des problèmes évoqués dans la conclusion du Chapitre 3.

Ainsi par exemple, les solutions pour les problèmes liés à l'absence de vue globale du réseau dans le cas des couplages spatiaux sont imparfaites (cf. Section 6.4). L'échange d'informations liées aux routes et aux adresses qui sont connues pour chacun des modèles, pourrait idéalement être automatiquement organisé dans le multi-modèle, dans le cadre d'une co-initialisation.

Afin d'étendre la portée de nos travaux, certaines des hypothèses sur lesquelles nous nous sommes appuyés pourraient également être supprimées :

1. L'intégration des solutions proposées par [Riley et al., 2001a] (basées sur le Dynamic Simulation Backplane présentée à la Section 3.4.1) pour réussir à coupler spatialement deux simulateurs IP qui utilisent des implémentations non standards des protocoles, pourraient être intégrées, afin de pouvoir intégrer plus de simulateurs IP différents.
2. La gestion de l'hétérogénéité de MECSYCO devrait pouvoir permettre d'intégrer des simulateurs IP qui ne reposent pas sur le formalisme discret. Une expérimentation a déjà été effectuée dans ce sens : celle-ci permet à deux modèles NS-3 de s'échanger des paquets

IP simulés, en les faisant transiter via un modèle hybride qui modélise la couche physique d'un câble Ethernet 10 Mb/s (avec un codage Manchester pour la partie discrète et une équation différentielle représentant le signal électrique correspondant – bruité – pour la partie continue).

3. La modularité de MECSYCO permettant à terme de changer son algorithme de synchronisation du temps conservatif par une version optimiste, les wrappeurs DEVS pourraient d'ors et déjà intégrer la possibilité de faire des rollbacks sur les simulateurs IP. La solution basée sur la duplication des processus, utilisée dans le projet FNCS (cf. Section 3.5.3), pourrait être utilisée comme solution concrète pour son implémentation.
4. La capacité de certains simulateurs IP à être reliés à des logiciels ou des équipements réseau réels, pourrait être exploitée afin de pouvoir les intégrer à des co-simulations, par l'intermédiaire des wrappeurs DEVS.

Enfin, les différents concepts qui ont été proposés dans ces travaux, pourraient faire l'objet à terme de la création d'un standard d'interopérabilité pour les modèles de réseaux IP.

Ce standard pourrait prendre la forme d'un DSL (*Domain Specific Language*), permettant de décrire des réseaux IP de façon non-ambiguë, afin de pouvoir automatiquement proposer un multi-modèle exécutable. En utilisant plusieurs bibliothèques IP différentes grâces aux couplages spatiaux, avec des modèles d'applications tierces grâces aux couplages structurels, cette description pourrait aller jusqu'à permettre de produire le code complet d'une co-simulation de SCP. Des langages comme celui proposé par [Perumalla et al., 1998], pourraient permettre de formaliser cette DSL.

# Annexe A

# Simulateurs IP et co-simulations : autres travaux

**Sommaire**



## A.1 Présentation

Cette annexe complète la liste des travaux cités dans le Chapitre 3. Les travaux présentés ici ne sont pas présentés en détail, mais simplement catégorisés en fonction de leur mode d'intégration de modèles IP à des co-simulations.

## A.2 Distribution et parallélisation de simulateurs IP

### A.2.1 Simulateurs IP distribués par conception

— wDSEnv/wDSLang [Iazeolla et al., 2010] – Simulateur spécialisé dans les modèles IP sans-fil.
— SimArch/jEQN [Gianni et al., 2008] – Simulateur généraliste mais utilisable pour des modèles IP.

### A.2.2 Surcouche pour la distribution de simulateurs IP séquentiels existants

— DEVS/NS-2 [Kim et al., 2008] – Surcouche à NS-2 utilisant DEVS (synchronisation conservative à l'aide d'une file d'attente partagée).

— Genesis [Szymanski et al., 2002] – Surcouche à NS-2, qui a l'originalité d'être un algorithme de synchronisation de type optimiste (algorithme *Time-Space Mappings*).
— U.P.S. [Nicol and Heidelberger, 1996] – Surcouche au simulateur CSIM (synchronisation conservative, avec les algorithmes YAWNS, WHOA ou PUCS).
— Avec OpNet [Wu et al., 2001] – Surcouche au simulateur commercial et propriétaire OpNet (synchronisation conservative avec FDK/BRTI).
— Avec NS-2 [Jones and Das, 2000] – Surcouche à NS-3, avec une synchronisation conservative basée sur l'algorithme Chandy-Misra-Bryant (utilisé par MECSYCO).

## A.3 Interconnexion de simulateurs IP hétégorènes

— Parallel ODE [Liu, 2007] – Interconnexion de ODE Fluid TCP et d'un simulateur IP maison.
— Avec Simulink [Yeung et al., 2004] – Interconnexion de QualNet et Simulink.

## A.4 Interconnexion de simulateurs IP avec des simulateurs d'autres domaines

— FSKIT [Kelley et al., 2015] – Interconnexion de GridDyn et de NS-3 (synchronisation conservative avec l'algorithme YAWNS modifié).
— GECO [Lin et al., 2012][Lin, 2012] – Interconnexion de PSLF et de NS-2 (synchronisation en exploitant la pile de NS-2).
— Multidisciplinary SM Expe. [Lévesque et al., 2012] – Interconnexion de OpenDSS et OMNeT++/INET (synchronisation en partageant une pile d'événéments).
— Pacific Northwest National Laboratory [Fuller et al., 2013] – Interconnexion de GridLAB-D et de NS-3 (synchronisation basée sur un système de statistiques).
— PowerWorld/NS-3 [Tariq et al., 2014] – Interconnexion de PowerWorld et NS-3.
— HLA-OMNET++ [Galli et al., 2008] – Interconnexion de OMNeT++/INET et de simulateurs compatibles HLA.
— Matlab/OMNeT++ [Mets et al., 2011] – Interconnexion de Matlab et OMNeT++/INET.
— Gnutella [He et al., 2003] – Interconnexion de Gnutella et NS-2/PDNS/GTNetS.
— OpenDSS/NS-2 [Godfrey et al., 2010] – Interconnexion de OpenDSS et NS-2.
— PowerNet [Liberatore and Al-Hammouri, 2011] – Interconnexion de Modelica et NS-2 (synchronisation en exploitant la pile de NS-2).
— VPNET [Li et al., 2011] – Interconnexion de VTB et OpNet (synchronisation conservative).

# Annexe B

# Détail des algorithmes des fonctions du protocole de simulation DEVS, pour un simulateur IP

**Sommaire**



## B.1 Présentation

Cette annexe complète les indications fournies dans le Chapitre 6, pour implémenter les fonctions du protocole de simulation DEVS d'un artéfact de modèle (wrappeur DEVS), dans le cadre de l'intégration d'un simulateur IP. Elle détaille également la nature des interactions attendues entre le thread principal du simulateur IP, et celui de l'artéfact de modèle.

## B.2 Environnement prérequis

L'artéfact de modèle doit contenir la liste des ports DEVS disponibles dans le modèle, associés à un identifiant unique et à un rôle (i.e. port d'entrée ou de sortie). Cette liste est constituée à l'initialisation du modèle, lorsque les applications correspondant aux ports DEVS sont instanciées.

La fonction $getDevsOutputPort(y_k^i) : Port$ de l'artéfact de modèle doit pouvoir retourner l'objet application correspondant au port associé à l'identifiant $y_k^i$. Les objets *Port* doivent au minimum offrir ces fonctions :

— Pour les ports d'entrée :
  — $send(data) : void$ — Envoie la donnée applicative *data* via le réseau IP simulé.
— Pour les ports de sortie :

— $getNewOutputExternalEvent() : Event$ — Retourne un objet correspondant à un événement externe de sortie, qui contient la dernière donnée reçue en provenance du réseau IP simulé ainsi que le temps de simulation auquel elle a été interceptée par le port.
— $isThereANewOutputExternalEvent() : bool$ — Retourne vrai si le port a reçu une donnée en provenant du réseau IP simulé, depuis la dernière fois que la fonction $getNewOutputExternalEvent$ a été invoquée.

L'artéfact de modèle doit également avoir lui-même à sa disposition les fonctions suivantes :

— $setNewInputExternalEvent(x_k^i, data) : void$ — Enregistre la donnée $data$, avec une référence vers son port d'entrée de destination $x_k^i$, en tant que dernier événement externe d'entrée reçu en provenance du m-agent.
— $getNewInputExternalEvent() : Event$ — Retourne un objet correspondant à un événement externe d'entrée, qui contient la dernière donnée reçue en provenance du m-agent. L'objet $Event$ retourné contient une référence vers le port à qui est destiné la donnée d'entrée, ainsi que la donnée elle-même.
— $isThereANewInputExternalEvent() : bool$ — Retourne vrai si l'artéfact de modèle a reçu une donnée en provenance de son m-agent, depuis la dernière fois que la fonction $getNewInputExternalEvent()$ a été invoquée.

La fonction $runOneMainLoopIteration() : void$ sera également utilisée dans les algorithmes qui suivent. Cette fonction ordonne au simulateur IP d'exécuter un tour de la boucle principale qui est utilisée par son ordonnanceur. Le détail de cette fonction sera étudié par la suite.

Enfin, on considère que le simulateur IP propose lui-même les fonctions suivantes (ou équivalentes) :

— $setCurrentTime(t_i) : void$ — Règle l'horloge du simulateur IP au temps de simulation $t_i$.
— $getNextEventTime() : double$ — Retourne le temps de simulation correspondant à l'événement interne le plus imminent dans la pile des événements du simulateur IP. Ce type de fonction peut généralement être obtenu en ayant accès à la pile des événements, e.g. avec des objets comme $getEventStack().getFirst().getTime()$.

Si ces deux dernières fonctions ne sont pas accessibles en dehors de l'ordonnanceur, des fonctions permettant un accès facile sont ajoutées lors de la surcharge de l'ordonnanceur (c'est le rôle de la partie proxy qui est représentée dans la Figure 6.10 du Chapitre 6, page 105).

La section suivante présente les fonctions maîtresses de l'artéfact de modèle, qui sont à implémenter.

## B.3 Fonctions du protocole de simulation DEVS

Comme indiqué à la Section 2.5.5, concevoir un artéfact de modèle consiste à réussir à implémenter les fonctions du protocole de simulation DEVS, pour le simulateur et ses modèles.

La fonction *init* est facultative et ne sera nécessaire que si le modèle a besoin d'être initialisé (e.g. passage de valeurs par défaut).

La fonction $getNextInternalEventTime$ doit pouvoir retourner le temps de simulation lié au prochain événement interne du modèle IP, en interrogeant le simulateur IP (cf. Algorithme B.1).

```
1 Function getNextInternalEventTime() : double is
2     return simulator.getNextEventTime();
```

ALGORITHME B.1 – Fonction de l'artéfact de modèle : retourne le temps du prochain événement interne du modèle IP.

La fonction *getExternalOutputEvent* doit pouvoir retourner l'événement externe de sortie $eout_i^k$ qui serait présent dans le port enregistré avec l'identifiant $y_k^i$. Cette fonction consiste donc à consulter la liste des ports DEVS connus par l'artéfact, pour récupérer le pointeur qui est associé au port DEVS de sortie qui porte le nom $y_k^i$. Si le port a effectivement produit un nouvel événement externe de sortie depuis la dernière fois qu'il a été interrogé, il doit être récupéré et directement retourné. S'il n'a produit aucun événement de sortie, la fonction doit simplement renvoyer la valeur *null*. L'Algorithme B.2 illustre une implémentation de cette fonction.

```
1 Function getExternalOutputEvent(y_k^i) : Event is
2     port ⟵ getDevsOutputPort(y_k^i)
3     eout_i^k ⟵ null
4     if port.isThereANewOutputExternalEvent() then
5         eout_i^k ⟵ port.getNewOutputExternalEvent()
6     return eout_i^k
```

ALGORITHME B.2 – Fonction de l'artéfact de modèle : retourne l'événement externe de sortie $eout_i^k$ éventuellement présent dans le port $y_k^i$.

La fonction *processExternalInputEvent* doit pouvoir transmettre l'événement externe d'entrée $ein_i$, au port d'entrée enregistré avec l'identifiant $x_i^k$, à qui il est destiné. Comme précédemment, cette fonction consiste donc à récupérer le pointeur associé au port DEVS qui est concerné, pour lui transmettre l'événement. L'horloge du simulateur IP doit être réglée à la date exacte qui est associée à l'événement à traiter (ce qui correspondra toujours à un bond dans le futur), avant que le port ne le traite. Le traitement de l'événement externe d'entrée par le port consiste à utiliser la fonction appropriée du simulateur, par exemple avec l'appel d'une fonction du type *socket.send*(*data*).

On constate que dans l'Algorithme B.3, la demande de traitement de la donnée par le port concerné est réalisée de façon indirecte. Le port et la donnée associés à l'événement externe sont simplement enregistrés dans l'artéfact, avant d'appeler la fonction *runOneMainLoopIteration*, qui fera la suite du travail. Cette indirection aura pour conséquence de faire exécuter ce traitement par le thread du simulateur IP lui-même. Cette façon de faire, qui consiste à ne jamais manipuler le modèle directement, permet de ne jamais risquer de rendre l'état du modèle incohérent, dans le cas où le simulateur ne serait pas *thread-safe*.

La dernière fonction $processInternalEvent(t_i) : void$ doit pouvoir être capable de demander au simulateur IP de traiter son prochain événement interne (le temps $t_i$ correspondra toujours au temps du prochain événement, tel qu'il a été renvoyé par la fonction $getNextInternalEventTime$). Cette opération consiste simplement à faire un appel à la même fonction $runOneMainLoop\ Iteration()$ que celle qui a été utilisée dans $processExternalInputEvent$, comme illustré dans l'Algorithme B.4.

```
1 Function processExternalInputEvent(ein_i, x_i^k) : void is
2     port ⟵ getDevsInputPort(x_i^k)
3     time ⟵ ein_i["time"]
4     data ⟵ ein_i["data"]
5     setNewInputExternalEvent(port, data)
6     simulator.setCurrentTime(time)
7     runOneMainLoopIteration()
```

Algorithme B.3 – Fonction de l'artéfact de modèle : exécute l'événement externe d'entrée $ein_i$, en le transmettant au port $x_i^k$.

```
1 Function processInternalEvent(t_i) : void is
2     runOneMainLoopIteration()
```

Algorithme B.4 – Fonction de l'artéfact de modèle : exécute l'événement interne du modèle IP qui est le plus imminent.

La fonction *runOneMainLoopIteration* n'est généralement pas triviale à obtenir, selon le simulateur qu'on souhaite intégrer, et le type d'ordonnanceur qu'il utilise. Elle nécessite des modifications sur ce dernier.

## B.4 Synchronisation avec l'ordonnanceur

Pour pouvoir utiliser une fonction du type *runOneMainLoopIteration* et donc contraindre l'avancée de la simulation, il faut modifier cet ordonnanceur, tel que détaillé dans le Chapitre 6 et illustré de nouveau dans l'Algorithme B.5.

L'ajout de verrous dans la boucle principale (lignes 2, 3 et 15) permet de pouvoir arrêter et relancer celle-ci à loisir. L'objectif est que le wrappeur DEVS puisse utiliser ces verrous pour ordonner l'exécution du prochain événement interne du simulateur IP, et uniquement celui-ci. Pour ce faire, le wrappeur DEVS disposera de la fonction *runOneMainLoopIteration*, dont le contenu est détaillé dans l'Algorithme B.6. Le système de double verrou correspond à l'utilisation d'un sémaphore binaire (un seul jeton), permettant de rendre l'exécution de deux threads séquentielle (alternance), comme illustré dans le schéma d'exemple de la Figure B.1.

Lorsque la fonction *isThereANewInputExternalEvent* retourne *vrai* ligne 5, cela signifie que la fonction *runOneMainLoopIteration* a été appelée par la fonction *processExternalInput Event* pour demander le traitement d'un événement externe d'entrée à un port DEVS. Sinon, cela signifie qu'elle a été appelée par la fonction *processInternalEvent*, pour faire exécuter le prochain événement de la pile d'événements du simulateur.

```
1  repeat
       /* Contrôle de l'exécution de la boucle (1/2) */
2      waitForWrapperLockOpening()
3      closeWrapperLock()
4      if ¬isSimulationStopRequested() then
           /* Injection des événements externes */
5          if modelArtifact.isThereANewInputExternalEvent() then
6              externalEvent ⟵ modelArtifact.getNewInputExternalEvent()
7              devsPort ⟵ externalEvent["port"]
8              data ⟵ externalEvent["data"]
9              devsPort.send(data)
           /* Traitement des événements internes */
10         else if ¬isEventStackEmpty() then
11             eventTime ⟵ getNextEventTime()
12             repeat
13                 processNextEvent()
14             until isEventStackEmpty() || getNextEventTime() ≠ eventTime ||
                   isSimulationStopRequested()
       /* Contrôle de l'exécution de la boucle (2/2) */
15     openSimulatorLock()
16 until isEventStackEmpty() || isSimulationStopRequested()
```

Algorithme B.5 – Exemple de boucle principale utilisée par un simulateur IP, modifiée pour pouvoir être contrôlée par un wrappeur DEVS.

```
1 Function runOneMainLoopIteration() : void is
2     openWrapperLock()
3     waitForSimulatorLockOpening()
4     closeSimulatorLock()
```

Algorithme B.6 – Fonction interne à l'artéfact de modèle : demande l'exécution d'une itération de la boucle principale du simulateur IP.

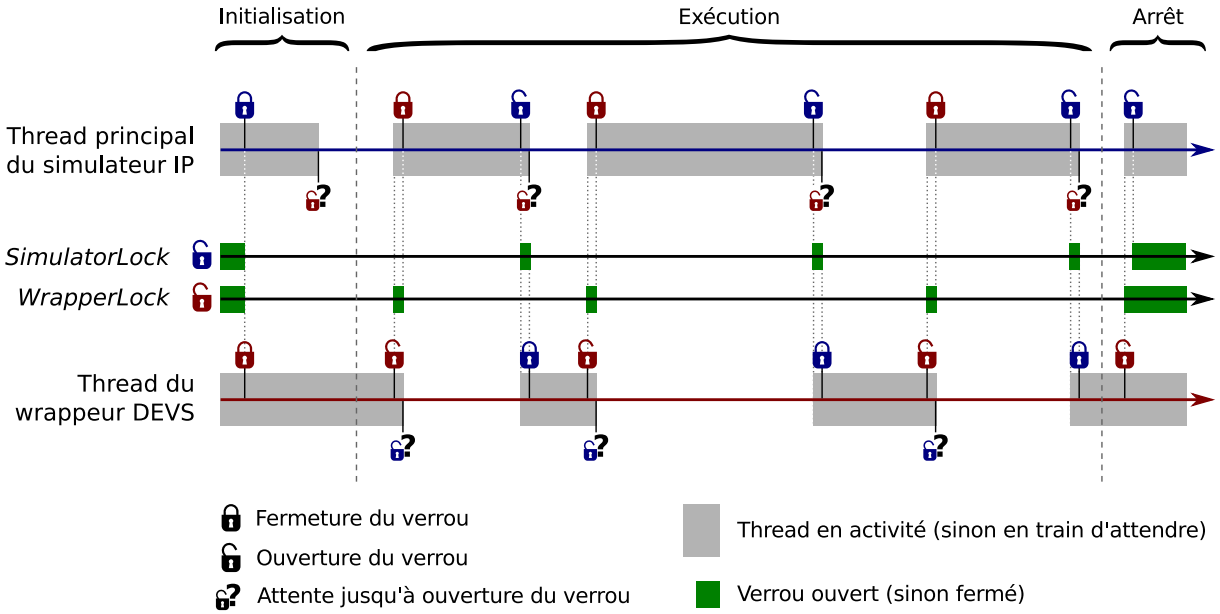



FIGURE B.1 – Exemple d'interactions en fonction du temps, entre le thread principal du simulateur IP (contenant la boucle principale) et le thread du wrappeur DEVS, correspondant à une synchronisation par exclusions mutuelles.

# Annexe C

# Détail de l'implémentation des bibliothèques MECSYCO pour NS-3 et OMNeT++/INET

**Sommaire**



## C.1 Présentation

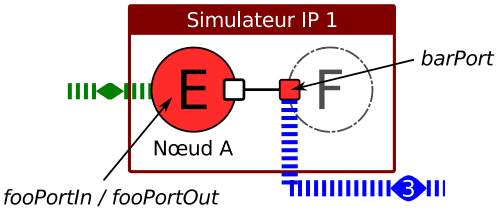
Cette annexe complète les indications fournies dans la Section 7.3, et explique la façon dont NS-3 et ONMNeT++/INET ont été wrappés en DEVS, en implémentant les solutions proposées dans ce manuscrit, tout en détaillant le fonctionnement des bibliothèques créées.

Pour chacun des deux simulateurs IP, sont décrits :

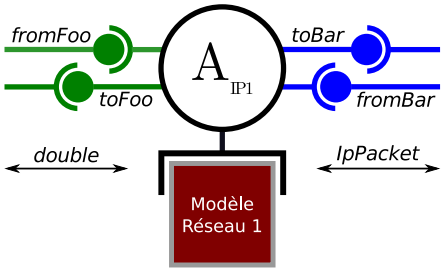
— comment l'artéfact de modèle a été intégré au simulateur (contrôle de la boucle principale) ;
— comment les ports structurels et spatiaux ont été implémentés ;
— comment les appendices ont été implémentés ;

— comment les différents composants MECSYCO (e.g. m-agent, artéfacts de couplages) peuvent être définis et instanciés au lancement de la simulation.

La présentation des bibliothèques s'appuiera sur l'exemple de modèle IP couplé minimaliste qui est proposé dans la Figure C.1. Le format des trames et paquets IP qui a été choisi pour faire les échanges spatiaux entre les deux simulateurs, est le format standardisé par les RFC. Le simulateur OMNeT++/INET pourra également parfois être abrégé en *O++*.




(a) Modèle réseau, avec le nom des ports et sans les modèles couplés.




(b) Multi-modèle MECSYCO correspondant, avec le nom des artéfacts de couplage et le type de données échangées, et sans les autres m-agents.

FIGURE C.1 – Exemple de réseau simple structurellement et spatialement couplé.

L'intégration des simulateurs IP présentée ci-après s'appuie sur l'existence de la version C++ de MECSYCO.

## C.2 NS-3

### C.2.1 Organisation

Écrire et simuler un modèle NS-3 consiste à écrire un programme C++ depuis sa fonction `main`, à le compiler et à directement exécuter le binaire qui est produit. Le topologie IP, la configuration des différents composants et les événements qui auront lieu au cours de la simulation, sont tous décrits dans le code C++, à l'aide d'objets proposés par les différentes bibliothèques de NS-3 qui sont à lier au projet.

Écrire et simuler un modèle NS-3, en l'intégrant à une co-simulation MECSYCO, consiste simplement à ajouter une bibliothèque MECSYCO à l'édition des liens. Cette bibliothèque contient le cœur de MECSYCO, ainsi que les différentes classes spécifiques à l'intégration NS-3, qui ont été développées en s'appuyant sur le travail décrit dans ce manuscrit de thèse. L'Implémentation C.7 est un exemple de fonction `main`, utilisée pour intégrer un modèle NS-3 à une co-simulation MECSYCO, en utilisant les outils fournis par cette bibliothèque. Cet exemple correspond à celui proposé par la Figure C.1.

On constate ligne 5 que tout ce qui est lié au modèle NS-3 a été déplacé de la fonction *main* à une classe `MyModel`, créée pour l'occasion. Le reste du `main` est utilisé pour permettre à ce modèle de s'interconnecter avec la plateforme MECSYCO.

Le modèle NS-3 est fourni à l'artéfact de modèle NS-3, à la ligne 8. Comme nous l'avons vu dans les chapitres précédents, cet artéfact de modèle est un wrappeur DEVS, qui va permettre

```
1  int main() {
2      thread_group threadGroup;

4      // Implémentation du modèle NS-3 définie dans une classe MyModel
5      MyModel myModel;

7      // Artéfact de modèle MECSYCO (wrappeur DEVS)
8      Ns3ModelArtefact modelArtefact(myModel, threadGroup);

10     // M-agent MECSYCO
11     EventMAgent magent(LOOKAHEAD, MAX_SIMULATION_TIME);
12     magent.setModelArtefact(exampleModel);

14     // Artéfacts de couplage à relier aux ports DEVS
15     CouplingArtifactReceiver<double> fooReceiver("fromFoo", threadGroup);
16     CouplingArtifactReceiver<MecsycoIpPacket> barReceiver("fromBar", threadGroup);
17     CouplingArtifactSender fooSender("toFoo");
18     CouplingArtifactSender barSender("toBar");

20     // Liens entre les ports DEVS et les artéfacts de couplage
21     magent.addInputCouplingArtifact((CouplingArtifactReader&) fooReceiver, "fooPortIn");
22     magent.addOutputCouplingArtifact((CouplingArtifactWriter&) fooSender, "fooPortOut");
23     magent.addInputCouplingArtifact((CouplingArtifactReader&) barReceiver, "barPort");
24     magent.addOutputCouplingArtifact((CouplingArtifactWriter&) barSender, "barPort");

26     // Démarrage de la simulation
27     magent.start(threadGroup);

29     threadGroup.join_all();
30 }
```

Implémentation C.7 – Exemple de programme principal pour intégrer un modèle NS-3 à un multi-modèle exécuté par MECSYCO.

d'ouvrir le simulateur au monde extérieur, en lui ajoutant des fonctions DEVS. Ce même artéfact de modèle est fourni à un m-agent à la ligne 12. Ce m-agent va prendre en charge la dynamique de la simulation, et organiser les échanges d'événements avec l'extérieur, au rythme de la co-simulation. Il est instancié à la ligne précédente, avec deux macros correspondant à des temps qui doivent être exprimés dans l'unité de temps utilisée par NS-3. Le lookahead doit correspondre ici au temps simulé minimum que prendra un paquet pour traverser le seul et unique lien du modèle (correspondant donc au double du délai des cartes réseau utilisées).

Les lignes 15 à 18 instancient les artéfacts de couplage. Les `Receiver` correspondent à des artéfacts entrants, tandis que les `Sender` correspondent à des artéfacts sortants. Un type de donnée est défini pour les artéfacts entrants : il correspond au type de donnée que l'artéfact sera chargé de réceptionner, depuis un autre simulateur impliqué dans la co-simulation. On devine donc que l'artéfact de la ligne 15 est utilisé pour le couplage structurel (réception de doubles), tandis que celui de la ligne 16 est utilisé pour le couplage spatial de niveau 3 (réception de paquets IP).

Tous les artéfacts de couplage sont instanciés en leur passant un nom, sous forme de chaîne de caractères. Ce nom sera utilisé par l'intergiciel de communication de MECSYCO, pour relier les couplages sortants avec les couplages entrants, d'un m-agent à l'autre. Ainsi, par exemple, il existera ailleurs dans cette co-simulation, un artéfact de couplage sortant qui aura pour nom *fromBar* et qui enverra des doubles, depuis un autre simulateur.

Ces artéfacts de couplage sont ensuite liés au m-agent dans les lignes 21 à 24, en prenant soin de les associer à des noms de port DEVS. Ces noms de port sont fournis sous forme de chaînes de caractères, et ont été définis dans le modèle `myModel`.

Enfin, l'exécution du m-agent MECSYCO est demandée à la ligne 27, ce qui aura pour conséquence de lancer l'exécution du modèle NS-3 `myModel`, au travers de son artéfact de modèle.

Les premières et dernières lignes montrent l'existence d'un groupe de threads. Celui-ci est donné en argument à tous les objets qui seront amenés à lancer un thread différent (cf. Section 6.3.2), et qui l'ajouteront donc à ce groupe. En dernière ligne, on attend que l'exécution de tous les threads soit terminée, avant de quitter le `main`. Cette pratique permet simplement de simplifier l'écriture de la simulation MECSYCO, en s'épargnant de devoir gérer l'utilisation de la mémoire.

Il ne reste plus qu'à implémenter le modèle NS-3 dans la classe `MyModel`, et à définir l'emplacement des ports DEVS.

### C.2.2 Implémentation du modèle

La classe `MyModel`, utilisée dans l'exemple précédent, est dérivée de la classe abstraite `MecsycoNs3Model`, fournie par MECSYCO. Cette dernière se contente d'ajouter une fonction `start` utilisée par le m-agent, permettant d'exécuter la classe (et donc le modèle NS-3) dans un thread séparé et relié au groupe de threads. La modèle NS-3 est décrit lors de la surcharge de l'opérateur *parenthèses* de la classe, comme proposé dans l'Implémentation C.8.

```
1  void MyModel::operator()() {
2
3      // Modèle de réseau simple décrit avec la bibliothèque NS-3
4      nodes.Create(2);
5      ipStack.Install(nodes);
6      netDevices = pointToPoint.Install(nodes);
7      interfaces = ipAddresses.Assign(netDevices);
8
9      // Installation du port DEVS structurel d'entrée
10     MecsycoStructuralDevsPortInHelper mecsycoClientApp;
11     mecsycoClientApp.SetAttribute("MecsycoPort", StringValue("fooPortIn"));
12     mecsycoClientApp.SetAttribute("L3RemoteAddress", AddressValue(interfaces.GetAddress(1)));
13     mecsycoClientApp.SetAttribute("L4Protocol", StringValue("tcp"));
14     mecsycoClientApp.SetAttribute("L4Port", UintegerValue(1021));
15     mecsycoClientApp.Install<double>(nodes.Get(0));
16
17     // Installation du port DEVS structurel de sortie
18     MecsycoStructuralDevsPortOutHelper<double> mecsycoServerApp;
19     mecsycoServerApp.SetAttribute("MecsycoPort", StringValue("fooPortOut"));
20     mecsycoServerApp.SetAttribute("L3Version", UintegerValue(6));
21     mecsycoServerApp.SetAttribute("L4Protocol", StringValue("tcp"));
22     mecsycoServerApp.SetAttribute("L4Port", UintegerValue(1042));
23     mecsycoServerApp.Install(nodes.Get(0));
24
25     // Installation du port spatial bidirectionnel (port DEVS d'entrée + de sortie)
26     MecsycoSpatial3DevsPortsInOut::create(nodes.Get(1)->GetDevice(0), "barPort", PORT_INOUT);
27
28     // Utilisation de l'ordonnanceur NS-3 fourni par la bibliothèque MECSYCO
29     GlobalValue::Bind("SimulatorImplementationType", StringValue("ns3::MecsycoSimulatorImpl"));
30
31     // Exécution du modèle NS-3
32     Simulator::Run();
33     Simulator::Destroy();
34 }
```

Implémentation C.8 – Exemple de modèle NS-3 avec des ports structurels et spatiaux.

Ce modèle correspond à la description de la topologie IP illustrée par la Figure C.1. L'inté-

gralité du réseau IP est décrit dans les lignes 4 à 7, à l'aide d'objets fournis par NS-3.

Le premier nœud devant accueillir un port structurel bidirectionnel, il nécessite l'installation d'un port structurel DEVS d'entrée, et d'un port structurel DEVS de sortie. Les lignes 10 à 15 décrivent la création et l'installation du port structurel DEVS d'entrée, correspondant également à une application cliente pour NS-3.

Quatre attributs sont donnés au port d'entrée (ils sont spécifiques à l'application de port fournie par MECSYCO), avant de l'installer sur le nœud :

— **MecsycoPort :** Il s'agit du nom permettant d'identifier le port DEVS, vis-à-vis de MECSYCO. Lorsque le port sera installé sur le nœud, celui-ci ira automatiquement s'enregistrer auprès de l'artéfact de modèle, sous ce nom.
— **L3RemoteAddress :** Toutes les données applicatives qui seront fournies au port par l'artéfact de modèle, seront envoyées via une socket simulée, à un autre nœud du réseau IP. Ce nœud est désigné par son adresse IP, qui doit être fournie ici.
— **L4Protocol :** La socket simulée peut être soit du type `"tcp"`, soit du type `"udp"`. Cette valeur doit être en accord avec le type de socket sur laquelle écoute le nœud portant l'adresse *L3RemoteAddress*.
— **L4Port :** Il s'agit du port applicatif à utiliser, pour établir la liaison UDP ou TCP. Cette valeur doit être en accord avec le port sur lequel écoute la socket du nœud portant l'adresse *L3RemoteAddress*.

La création et l'installation du port structurel DEVS de sortie, correspondant également à une application serveur pour NS-3, sont décrites dans les lignes 18 à 23. Les attributs sont identiques à la version d'entrée, mais correspondent cette fois-ci au côté serveur d'une socket. L'attribut *L3Version*, qui définit si IPv6 ou IPv4 doit être utilisé pour communiquer, n'était pas présent côté client parce que cette information peut directement être déduite de la valeur de l'attribut *L3RemoteAddress*.

La ligne 26 illustre la création et l'installation du port spatial bidirectionnel de niveau 3, sur la carte réseau du second nœud. La chaîne de caractères qui est passée correspond au nom du port spatial pour MECSYCO, qui désigne à la fois le port DEVS d'entrée et celui de sortie. La macro passée en dernier argument indique le sens du port.

Enfin, la ligne 29 remplace l'implémentation de l'ordonnanceur qui sera utilisé pour simuler ce modèle, par une version modifiée contrôlable par l'artéfact de modèle. Les deux dernières lignes du modèle sont les instructions habituelles de NS-3, qui permettent de démarrer la simulation, puis d'effacer tous les objets créés une fois qu'elle est achevée.

### C.2.3 Implémentation des ports spatiaux

Le choix qui a été fait pour implémenter les ports spatiaux, en fonction des possibilités offertes par NS-3, a été de créer des wrappeurs pour carte réseau.

Lorsque la fonction `create` de la classe `MecsycoSpatial3DevsPortsInOut` est appelée, la carte réseau qui doit accueillir le port spatial, ainsi que le nom du port souhaité, sont passés en argument. Cette fonction statique se contente de créer une instance de `MecsycoSpatial3 DevsPortsInOut`, en lui passant la carte réseau en argument (ainsi que d'enregistrer le port auprès de l'artéfact de modèle).

**Port d'entrée** La fonction `send` de la couche 2 ou 3 associée à la carte réseau, est directement utilisée pour envoyer sur le lien les trames ou paquets transmis par l'artéfact de modèle.

**Port de sortie** Lorsqu'un nouvel objet `MecsycoSpatial3DevsPortsInOut` est instancié, son constructeur utilise la fonction *SetReceiveCallback* de la carte réseau, pour lui passer un pointeur vers l'une de ses fonctions. Ainsi, chaque fois que la carte réseau recevra une trame depuis le lien, c'est cette fonction qui sera appelée et qui devra prendre la responsabilité de son traitement, avec la possibilité de l'intercepter.

Il n'y a aucune manipulation à faire sur le format des trames et paquets pour les échanger avec l'extérieur, puisque NS-3 utilise directement leur représentation standardisée en interne.

### C.2.4 Appendices

Rien de spécifique ne désigne le second nœud comme un faux nœud, parce que les nœuds créés par NS-3 sont par défaut considérés comme des coquilles vides. Il peuvent donc directement faire office d'appendices.

Si le lien avait été un lien parfait, il aurait été configuré de façon similaire à l'exemple de l'Implémentation C.9.

```
1 PointToPointHelper pointToPoint;
2 NetDeviceContainer netDevices = pointToPoint.Install(nodes);

4 pointToPoint.SetChannelAttribute("Delay", 0);
5 pointToPoint.SetDeviceAttribute("InterframeGap", 0);
6 pointToPoint.SetDeviceAttribute("DataRate", StringValue("1000000Tbps"));
```

Implémentation C.9 – Création d'un lien parfait, avec NS-3.

## C.3 OMNeT++/INET

### C.3.1 Organisation

OMNeT++ sépare la définition des modèles, de la définition des paramètres des modèles, qui peuvent être modifiés au gré des expériences.

Les modèles sont écrits avec C++, et avec le langage de description de réseaux NED. Le langage NED permet de décrire la topologie IP, en s'appuyant sur des éléments eux-même écrits en NED. Les sous-modèles atomiques, en bout de course de la hiérarchie des fichiers NED, sont écrits en C++. Les paramètres des modèles, ainsi que toutes les informations liées à l'exécution de la simulation, sont stockés dans des fichiers INI. Ces derniers correspondent à des listes de clés et de valeurs, et peuvent parfois s'appuyer sur des fichiers XML annexes.

Écrire un modèle OMNeT++/INET consiste donc à décrire son réseau IP dans un fichier NED (écrit manuellement ou généré par une interface graphique), en combinant les sous-modèles fournis par la bibliothèque INET avec ceux éventuellement écrits pour l'occasion en C++. Le projet est ensuite compilé via l'environnement de compilation de OMNeT++, qui ajoute lui-même la fonction principale (`main`) du programme, et qui lie à l'exécutable les différentes bibliothèques

C++ qui sont nécessaires. Le binaire ainsi obtenu est indépendant, mais devra obligatoirement avoir un fichier INI passé en paramètre pour exécuter la simulation.

Modifier directement le `main` du programme pour ajouter les différents composants MECSYCO, comme c'est le cas avec NS-3, n'est donc pas possible avec OMNeT++. Le point d'entrée est le fichier INI.

### C.3.2 Fichier INI pour MECSYCO

Le principal intérêt du fichier INI pour intégrer un modèle OMNeT++/INET à une co-simulation MECSYCO, est qu'il permet de redéfinir l'ordonnanceur à utiliser.

En plus de permettre de modifier la boucle principale du programme, cette option permet d'obtenir un endroit où il est possible d'ajouter du code C++, en ayant l'assurance qu'il sera exécuté avant le traitement du premier événement interne du simulateur. Les différents composants de MECSYCO sont instanciés depuis l'ordonnanceur, et les paramètres de la co-simulation sont également récupérés depuis le fichier INI. Un exemple de fichier INI pour MECSYCO est donné par l'Implémentation C.10. Cet exemple correspond à celui proposé dans la Figure C.1.

```
1  // Utilisation de l'ordonnanceur OMNeT++ fourni par la bibliothèque MECSYCO
2  scheduler-class = MecsycoScheduler

4  // Paramètres de configuration spécifiques à MECSYCO pour instancier le m-agent et les
       artéfacts de couplage
5  mecsyco-couplings = xmldoc("couplings_net1.xml")
6  mecsyco-maximulationtime = 1d
7  mecsyco-lookahead = 20ms

9  // Modèle de réseau décrit avec OMNeT++/INET
10 network = mecsyco.examples.net1

12 *.NodeA.numTcpApp = 2

14 // Installation du port DEVS structurel d'entrée
15 *.NodeA.tcpApp[0].typename = "MecsycoStructuralDevsPortIn"
16 *.NodeA.tcpApp[0].mecsycoPort = "fooPortIn"
17 *.NodeA.tcpApp[0].l3RemoteAddress = "2001:db8:42::2"
18 *.NodeA.tcpApp[0].l4Port = 1021

20 // Installation du port DEVS structurel de sortie
21 *.NodeA.tcpApp[1].typename = "MecsycoStructuralDevsPortOut"
22 *.NodeA.tcpApp[1].mecsycoPort = "fooPortOut"
23 *.NodeA.tcpApp[1].l3Version = 6
24 *.NodeA.tcpApp[1].l4Port = 1042
```

Implémentation C.10 – Exemple de fichier INI avec des ports structurels.

La première ligne indique que l'ordonnanceur utilisé est celui qui est implémenté par la classe `MecsycoScheduler`. Dans cette classe, avant d'accepter le moindre événement interne, les composants de MECSYCO sont instanciés. Ces composants sont créés en fonction du contenu du fichier de description *couplings_net1.xml* qui est donné à `mecsyco-couplings` à la ligne 5, pour ce qui est des artéfacts de couplage, et en fonction du temps maximum de simulation et du lookahead qui sont fournis aux lignes 6 et 7, pour ce qui est du m-agent.

La ligne 10 indique le réseau IP décrit en NED, dans un fichier séparé, qui est à utiliser.

Le reste du fichier INI correspond à l'installation et la configuration des applications des nœuds du réseau qui sont décrits dans la topologie IP. Il s'agit donc de l'endroit approprié pour

définir et positionner les ports structurels DEVS. Ces derniers sont installés sur le nœud désigné comme *NodeA* dans le réseau NED, en tant qu'applications TCP. Les lignes 15 à 18 correspondent à la description du port DEVS d'entrée, tandis que les lignes 21 à 24 correspondent à la description du port DEVS de sortie. Les paramètres (spécifiques à MECSYCO) sont identiques à ceux décrits dans la section précédente, pour NS-3.

Les ports spatiaux seront définis directement via le fichier NED, qui sera présenté juste après la description du contenu du fichier XML.

### C.3.3 Fichier XML pour les artéfacts de couplage

Le fichier XML décrit ici correspond à celui qui est pointé par la ligne 5 du fichier INI présenté dans l'Implémentation C.10. Son contenu aurait pu être décrit directement dans le fichier INI, mais OMNeT++ permettant de déporter une partie de la configuration dans des fichiers XML, nous avons retenu ce choix pour la description des artéfacts de couplage de la simulation.

Un exemple de contenu du fichier XML *couplings_net1.xml* est proposé avec l'Implémentation C.11.

```
1 <coupling_artifacts>
2    <receiver id="fromFoo" type="double" port="fooPortIn" />
3    <receiver id="fromBar" type="MecsycoIpPacket" port="barPort" />
4    <sender id="toFoo" port="fooPortOut" />
5    <sender id="toBar" port="barPort" />
6 </coupling_artifacts>
```

Implémentation C.11 – Exemple de fichier XML à utiliser pour décrire les artéfacts de couplage via le fichier INI.

Les différents éléments qui décrivent les artéfacts de couplage, sont identiques à ceux utilisés pour NS-3 dans la section précédente. Ainsi, le nom des artéfacts de couplage pour MECSYCO est donné via le paramètre `id`, tandis que le nom du port DEVS auquel il doit être associé dans le modèle, est donné via le paramètre `port`. Les artéfacts de couplage d'entrée (`receiver`) possèdent un type supplémentaire, qui indique le type de donnée qu'ils recevront de la part du m-agent MECSYCO.

La dernière étape est de décrire la topologie IP, tout en positionnant les ports spatiaux DEVS.

### C.3.4 Fichier NED pour la topologie IP

Un exemple topologie IP décrite par un fichier NED est proposé avec l'Implémentation C.12.

Le réseau décrit est identique à celui présenté dans la Figure C.1, avec deux nœuds directement interconnectés. Le nœud *NodeB* est un faux nœud, qui accueille un port spatial de niveau 3, comme expliqué ci-dessous.

### C.3.5 Implémentation des ports spatiaux

L'implémentation des ports spatiaux utilise en partie la méthode du wrappeur, qui a été utilisée pour NS-3.

```
1  package mecsyco.examples;

3  // Modèle de réseau décrit avec OMNeT++/INET
4  network net1 {
5     types:
6        channel C extends DatarateChannel {
7           delay = 10ms;
8        }
9     submodules:
10       NodeA: StandardHost;

12       // Installation du port spatial bidirectionnel (port DEVS d'entrée + de sortie)
13       NodeB: MecsycoSpatial3DevsPortsInOutHost {
14          portDevices = "1";
15          portNames = "barPort";
16          portDirections = "inout";
17       }
18    connections:
19       NodeA.pppg++ <--> C <--> NodeB.pppg++;
20 }
```

Implémentation C.12 – Exemple de fichier NED, avec un port spatial de niveau 3.

Il y a toutefois deux différences à prendre en considération :

1. l'API de OMNeT++/INET ne propose pas de définir une fonction personnalisée pour intercepter les trames et paquets sur l'une de ses cartes réseau simulées ;
2. le modélisateur n'a pas directement accès aux objets C++, et ne décrit la topologie IP qu'en utilisant le langage NED (ou son interface graphique, qui est équivalente).

La solution retenue a été de surcharger l'objet NED qui définit le nœud standard, afin de surcharger l'élément atomique qu'il utilise pour modéliser sa couche IP en C++. Dans le fichier NED de l'exemple de l'Implémentation C.12, le nœud surchargé est appelé `MecsycoSpatial3DevsPortsInOutHost`. Les paramètres qui lui sont passés permettent de désigner les cartes réseau qui sont concernées, et de nommer les ports (des virgules peuvent être utilisées).

**Port d'entrée** Puisque le port spatial correspond à un wrappeur de couche IP, la fonction *handlePacketFromHL* (équivalente à une méthode *send*, dans notre cas) est directement utilisée pour transmettre le paquet IP sur le réseau. Cet emplacement ne permet pas d'envoyer des trames et limite donc pour l'instant notre solution pour OMNeT++/INET, aux couplages spatiaux de niveau 3.

**Port de sortie** Depuis le wrappeur de couche IP, il est également possible de surcharger les méthodes *handleIncomingDatagram* et *handleIncomingICMP* qui permettent d'intercepter les paquets, lorsqu'il sont à destination d'un port spatial de sortie.

La représentation des paquets IP n'étant pas au format standardisé dans la mémoire interne de OMNeT++, l'artéfact de modèle a également pour charge de convertir les paquets qui transitent par les artéfacts de couplage. Cette transformation se fait aisément, puisque l'API propose les fonctions adéquates, qui sont d'ordinaire utilisées pour faire communiquer les modèles avec des équipements réels.

### C.3.6 Appendices

Comme pour NS-3, rien de spécifique ne désigne le second nœud comme un faux nœud, parce que les nœuds créés par OMNeT++/INET sont par défaut considérés comme des coquilles vides : il peuvent donc directement faire office d'appendices.

Si le lien avait été un lien parfait, il aurait suffit de le configurer dans le fichier NED, en suivant l'exemple proposé par l'Implémentation C.13.

```
1 channel C extends DatarateChannel {
2     datarate = 1000000Tbps;
3     delay = 0;
4 }
```

Implémentation C.13 – Création d'un lien parfait, avec OMNeT++/INET.

# Publications de l'auteur

## Conférences internationales avec actes

[1] 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 Communications and Networking Simulation Symposium*, pages 193–200, Pasadena, LA, USA. Society for Computer Simulation International/ACM.

[2] 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., Pérez, J. B., and Prieta, F. d. l., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, number 9086 in Lecture Notes in Computer Science, pages 240–251. Springer International Publishing.

[3] Camus, B., Galtier, V., Caujolle, M., Chevrier, V., Vaubourg, J., Ciarletta, L., and Bourjot, C. (2016). Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO. In *Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 16), SCS/ACM (2016), pages 568–575, Pasadena, United States.

[4] Duflot, M., Quinson, M., Masseglia, F., Roy, D., Vaubourg, J., and Viéville, T. (2015). When sharing computer science with everyone also helps avoiding digital prejudices. In *Proceedings of the 7th international Scratch conference (Scratch2015AMS)*.

## Revues nationales avec comité de lecture

[1] Masseglia, F., Quinson, M., Vaubourg, J., Poirel, V., Taillant, É., Arias, S., and Viennot, L. (2015). Incitation à la découverte de la programmation. *Bulletin de la société informatique de France, numéro HS1, février 2015*, pages 51–58.

## Démonstrations

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

number 9086 in Lecture Notes in Computer Science, pages 320–323. Springer International Publishing.

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

## Soumission de revues

[1] Camus, B., Paris T., Vaubourg, J., Presse Y., Bourjot C., Ciarletta L. and Chevrier V. (2017). Co-simulation of Cyber-Physical Systems with the DEVS Wrapping Platform MECSYCO. Submitted to *Simulation : Transactions of the Society for Modeling and Simulation International*.

[2] 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. Submitted to *Technique et Science Informatiques (TSI)*.

## Rapports techniques

[1] Vaubourg, J., Chevrier, V., and Ciarletta, L. (2015). Intégration de simulateurs existants à une plateforme de co-simulation basée sur DEVS. In *Archives Ouvertes HAL*.

[2] Vaubourg, J., Ciarletta, L., and Chevrier, V. (2015). Concept Grid : Réseau de télécommunications. Non-Public Industrial Deliverable.

## Dépôts APP

- Contributions au développement du cœur Java de MECSYCO : dépôt APP *[MECSYCO-re-Java]* du 25/03/2015 (réf. *IDDN FR 001 14008 000 S A 2015 000 10000*).
- Contributions au développement du cœur C++ de MECSYCO : dépôt APP *[MECSYCO-re-C++]* du 25/11/2016 (réf. *IDDN FR 001 520035 000 SP 2016 000 10000*).
- Développement de l'artéfact de modèle NS-3 pour MECSYCO C++ : dépôt *[MMA4NS3]* en cours.
- Développement de l'artéfact de modèle OMNeT++/INET pour MECSYCO C++ : dépôt *[MMA4Omnet++]* en cours.

# Bibliographie


[Blockwitz et al., 2012] Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauß, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A. (2012). Functional Mockup Interface 2.0 : The Standard for Tool independent Exchange of Simulation Models. In *Proceedings of the 9th International MODELICA Conference ; September 3-5 ; 2012 ; Munich ; Germany*, pages 173–184.

[Blumsack and Fernandez, 2012] Blumsack, S. and Fernandez, A. (2012). Ready or not, here comes the smart grid ! *Energy*, 37(1) :61–68.

[Camus, 2015] Camus, B. (2015). *Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes*. phdthesis, 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*, DEVS '15, pages 85–90, San Diego, CA, USA. Society for Computer Simulation International.

[Camus et al., 2016] Camus, B., Galtier, V., Caujolle, M., Chevrier, V., Vaubourg, J., Ciarletta, L., and Bourjot, C. (2016). Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO. In *Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*, Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium (TMS/DEVS 16), SCS/ACM (2016), pages 568–575, Pasadena, United States.

[Chandy and Misra, 1979] Chandy, K. and Misra, J. (1979). Distributed Simulation : A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5) :440–452.

[Chang, 1999] Chang, X. (1999). Network Simulations with OPNET. In *Proceedings of the 31st Conference on Winter Simulation : Simulation—a Bridge to the Future - Volume 1*, WSC '99, pages 307–314, New York, NY, USA. ACM.

[Chavalarias et al., 2009] Chavalarias, D., Bourgine, P., Perrier, E., et al. (2009). French Roadmap for complex Systems 2008-2009. *French Complex Systems Roadmap by the French National Network for Comple*.

[Chazelle, 2012] Chazelle, B. (2012). Natural Algorithms and Influence Systems. *Commun. ACM*, 55(12) :101–110.

[Ciraci et al., 2014] Ciraci, S., Daily, J., Fuller, J., Fisher, A., Marinovici, L., and Agarwal, K. (2014). FNCS : A Framework for Power System and Communication Networks Co-simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, DEVS '14, pages 36 :1–36 :8, San Diego, CA, USA. Society for Computer Simulation International.

[Cowie et al., 1999] Cowie, J., Nicol, D., and Ogielski, A. (1999). Modeling the global Internet. *Computing in Science Engineering*, 1(1) :42–50.

[D. Wu et al., 2002] D. Wu, E. Wu, J. Lai, Y. A. Sekercioglu, A. Varga, and G. K. Egan (2002). Implementing MPI Based Portable Parallel Discrete Event Simulation Support in the OMNeT++ Framework. In *Proceedings of the 14th European Simulation Symposium (ESS'02)*, pages 243–248, Dresden, Germany.

[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*, WSC '97, pages 142–149, Washington, DC, USA. IEEE Computer Society.

[Desmeulles, 2006] Desmeulles, G. (2006). *Réification des interactions pour l'expérience in virtuo de systèmes biologiques multi-modèles*. Brest.

[Éric Ramat, 2006] Éric Ramat (2006). *Modélisation et Simulation Multi-agents, Application pour les Sciences de l'Homme et de la Société*, volume Chapitre 2. Hermes Sciences.

[Farhangi, 2010] Farhangi, H. (2010). The path of the smart grid. *IEEE Power and Energy Magazine*, 8(1) :18–28.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes multi-agents : vers une intelligence collective*. InterEditions.

[Ferber, 1997] Ferber, J. (1997). Les systèmes multi-agents : un aperçu général. *Techniques et sciences informatiques*, 16(8).

[Francisco, Varela, 1989] Francisco, Varela (1989). *Autonomie et Connaissance ; Essai sur le vivant*. Paris, Seuil.

[Fujimoto, 2001] Fujimoto, R. M. (2001). Parallel Simulation : Parallel and Distributed Simulation Systems. In *Proceedings of the 33Nd Conference on Winter Simulation*, WSC '01, pages 147–157, Washington, DC, USA. IEEE Computer Society.

[Fuller et al., 2013] Fuller, J., Ciraci, S., Daily, J., Fisher, A., and Hauer, M. (2013). Communication simulations for power system applications. In *2013 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6.

[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*.

[Galli et al., 2008] Galli, E., Cavarretta, G., and Tucci, S. (2008). HLA-OMNET++ : An HLA Compliant Network Simulator. In *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008*, pages 319–321.

[George Riley, 2014] George Riley (2014). PDNS - Parallel/Distributed NS.

[Gianni et al., 2008] Gianni, D., D'Ambrogio, A., and Iazeolla, G. (2008). A Layered Architecture for the Model-driven Development of Distributed Simulators. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 61 :1–61 :9, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Godfrey et al., 2010] Godfrey, T., Mullen, S., Dugan, R., Rodine, C., Griffith, D., and Golmie, N. (2010). Modeling Smart Grid Applications with Co-Simulation. In *2010 First IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 291–296.

[He et al., 2003] He, Q., Ammar, M., Riley, G., Raj, H., and Fujimoto, R. (2003). Mapping peer behavior to packet-level details : a framework for packet-level simulation of peer-to-peer systems. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003*, pages 71–78.

[Henderson et al., 2006] Henderson, T. R., Roy, S., Floyd, S., and Riley, G. F. (2006). Ns-3 Project Goals. In *Proceeding from the 2006 Workshop on Ns-2 : The IP Network Simulator*, WNS2 '06, New York, NY, USA. ACM.

[Hopkinson et al., 2006] Hopkinson, K., Wang, X., Giovanini, R., Thorp, J., Birman, K., and Coury, D. (2006). EPOCHS : a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *IEEE Transactions on Power Systems*, 21(2) :548–558.

[Iazeolla et al., 2010] Iazeolla, G., Pieroni, A., D'Ambrogio, A., and Gianni, D. (2010). A Distributed Approach to Wireless System Simulation. In *2010 Sixth Advanced International Conference on Telecommunications (AICT)*, pages 252–262.

[Jones and Das, 2000] Jones, K. and Das, S. (2000). Parallel execution of a sequential network simulator. In *Simulation Conference, 2000. Proceedings. Winter*, volume 1, pages 418–424 vol.1.

[Karnouskos and de Holanda, 2009] Karnouskos, S. and de Holanda, T. (2009). Simulation of a Smart Grid City with Software Agents. In *Third UKSim European Symposium on Computer Modeling and Simulation, 2009. EMS '09*, pages 424–429.

[Kelley et al., 2015] Kelley, B., Top, P., Smith, S., Woodward, C., and Min, L. (2015). A federated simulation toolkit for electric power grid and communication network co-simulation. In *2015 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6.

[Kim et al., 2009] Kim, S., Sarjoughian, H. S., and Elamvazhuthi, V. (2009). DEVS-suite : A Simulator Supporting Visual Experimentation Design and Behavior Monitoring. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 161 :1–161 :7, San Diego, CA, USA. Society for Computer Simulation International.

[Kim et al., 2008] Kim, T., Hwang, M. H., and Kim, D. (2008). DEVS/NS-2 Environment : An Integrated Tool for Efficient Networks Modeling and Simulation. *The Journal of Defense Modeling and Simulation : Applications, Methodology, Technology*, 5(1) :33–60.

[Kuhn, 1996] Kuhn, T. S. (1996). *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, IL, 3rd edition edition.

[Kurose, James F, 2005] Kurose, James F (2005). *Computer Networking : A Top-Down Approach Featuring the Internet*. Pearson Education India, 3e edition.

[Lee, 2008] Lee, E. A. (2008). Cyber Physical Systems : Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369.

[Lévesque et al., 2012] Lévesque, M., Xu, D. Q., Joós, G., and Maier, M. (2012). Communications and Power Distribution Network Co-simulation for Multidisciplinary Smart Grid Experimentations. In *Proceedings of the 45th Annual Simulation Symposium*, ANSS '12, pages 2 :1–2 :7, San Diego, CA, USA. Society for Computer Simulation International.

[Li et al., 2011] Li, W., Monti, A., Luo, M., and Dougal, R. (2011). VPNET : A co-simulation framework for analyzing communication channel effects on power systems. In *2011 IEEE Electric Ship Technologies Symposium (ESTS)*, pages 143–149.

[Liberatore and Al-Hammouri, 2011] Liberatore, V. and Al-Hammouri, A. (2011). Smart grid communication and co-simulation. In *2011 IEEE Energytech*, pages 1–5.

[Lin, 2012] Lin, H. (2012). *Communication infrastructure for the smart grid : A co-simulation based study on techniques to improve the power transmission system functions with efficient data networks*. PhD thesis, Virginia Polytechnic Institute and State University.

[Lin et al., 2012] Lin, H., Veda, S., Shukla, S., Mili, L., and Thorp, J. (2012). GECO : Global Event-Driven Co-Simulation Framework for Interconnected Power System and Communication Network. *IEEE Transactions on Smart Grid*, 3(3) :1444–1456.

[Liu, 2007] Liu, J. (2007). Parallel Simulation of Hybrid Network Traffic Models. In *21st International Workshop on Principles of Advanced and Distributed Simulation, 2007. PADS '07*, pages 141–151.

[Mets et al., 2011] Mets, K., Verschueren, T., Develder, C., Vandoorn, T., and Vandevelde, L. (2011). Integrated simulation of power and communication networks for smart grid applications. In *2011 IEEE 16th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 61–65.

[Minsky, 1965] Minsky, M. (1965). Matter, Mind and Models. *AI Memos (1959 - 2004)*.

[Nicol and Heidelberger, 1996] Nicol, D. and Heidelberger, P. (1996). Parallel Execution for Serial Simulators. *ACM Trans. Model. Comput. Simul.*, 6(3) :210–242.

[Nicol, 1998] Nicol, D. M. (1998). Scalability, locality, partitioning and synchronization in PDES. In *Twelfth Workshop on Parallel and Distributed Simulation, 1998. PADS 98. Proceedings*, pages 4–11.

[Nutaro, 2010] Nutaro, J. (2010). adevs : A Discrete EVent System simulator.

[Nutaro, 2011] Nutaro, J. (2011). Designing power system simulators for the smart grid : Combining controls, communications, and electro-mechanical dynamics. In *2011 IEEE Power and Energy Society General Meeting*, pages 1–5.

[Nutaro et al., 2007] Nutaro, J., Kuruganti, P., Miller, L., Mullen, S., and Shankar, M. (2007). Integrated Hybrid-Simulation of Electric Power and Communications Systems. In *IEEE Power Engineering Society General Meeting, 2007*, pages 1–8.

[Omicini et al., 2008] Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17(3) :432–456.

[Pelkey and Riley, 2011] Pelkey, J. and Riley, G. (2011). Distributed Simulation with MPI in Ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 410–414, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Perumalla et al., 1998] Perumalla, K., Fujimoto, R., and Ogielski, A. (1998). TED—a Language for Modeling Telecommunication Networks. *SIGMETRICS Perform. Eval. Rev.*, 25(4) :4–11.

[Quesnel, 2006] Quesnel, G. (2006). *Approche formelle et opérationnelle de la multi-modélisation et de la simulation des systèmes complexes : apports pour la simulation de Systèmes Multi-Agents*. PhD thesis, Littoral.

[Quesnel et al., 2009] Quesnel, G., Duboz, R., and Ramat, É. (2009). The Virtual Laboratory Environment – An operational framework for multi-modelling, simulation and analysis of complex dynamical systems. *Simulation Modelling Practice and Theory*, 17(4) :641–653.

[Quesnel, G. et al., 2005] Quesnel, G., Duboz, R., Versmisse, D., and Ramat, E. (2005). DEVS coupling of spatial and ordinary differential equations : VLE framework. *OICIMS*, 5 :281–294.

[Rajkumar et al., 2010] Rajkumar, R. R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical Systems : The Next Computing Revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA. ACM.

[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*, AAMAS '07, pages 150 :1–150 :3, New York, NY, USA. ACM.

[Riley et al., 2001a] Riley, G., Ammar, M., Fujimoto, R., Xu, D., and Perumalla, K. (2001a). Distributed network simulations using the dynamic simulation backplane. In *Distributed Computing Systems, 2001. 21st International Conference on.*, pages 181–188.

[Riley et al., 2001b] Riley, G., Ammar, M., and Zegura, E. (2001b). Efficient routing using NIx-Vectors. In *2001 IEEE Workshop on High Performance Switching and Routing*, pages 390–395.

[Riley et al., 1999] Riley, G., Fujimoto, R., and Ammar, M. (1999). A generic framework for parallelization of network simulations. In *7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 1999. Proceedings*, pages 128–135.

[Riley, 2003] Riley, G. F. (2003). The Georgia Tech Network Simulator. In *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research*, MoMeTools '03, pages 5–12, New York, NY, USA. ACM.

[Riley and Ammar, 2002] Riley, G. F. and Ammar, M. H. (2002). Simulating Large Networks - How Big is Big Enough ? In *In Conference on Grand Challenges for Modeling and.*

[Riley et al., 2004] Riley, G. F., Ammar, M. H., Fujimoto, R. M., Park, A., Perumalla, K., and Xu, D. (2004). A Federated Approach to Distributed Network Simulation. *ACM Trans. Model. Comput. Simul.*, 14(2) :116–148.

[Sekercioglu et al., 2003] Sekercioglu, Y. A., Varga, and Egan, G. K. (2003). Parallel Simulation Made Easy With OMNeT++. In *ESS 2003 : 15th European Simulation Symposium.*

[Siebert, 2011] Siebert, J. (2011). *Approche multi-agent pour la multi-modélisation et le couplage de simulations. Application à l'étude des influences entre le fonctionnement des réseaux ambiants et le comportement de leurs utilisateurs.* PhD thesis, Université Henri Poincaré - Nancy I.

[Stoffers et al., 2014] Stoffers, M., Bettermann, R., Gross, J., and Wehrle, K. (2014). Enabling Distributed Simulation of OMNeT++ INET Models. *arXiv :1409.0994 [cs].* arXiv : 1409.0994.

[Stoffers and Riley, 2012] Stoffers, M. and Riley, G. (2012). Hybrid Simulation of Packet-Level Networks and Functional-Level Routers. In *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 111–119.

[Szymanski et al., 2002] Szymanski, B. K., Saifee, A., Sastry, A., Liu, Y., and Madnani, K. (2002). Genesis : A System for Large-scale Parallel Network Simulation. In *Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation*, PADS '02, pages 89–96, Washington, DC, USA. IEEE Computer Society.

[Tariq et al., 2014] Tariq, M. U., Swenson, B. P., Narasimhan, A. P., Grijalva, S., Riley, G. F., and Wolf, M. (2014). Cyber-physical Co-simulation of Smart Grid Applications Using Ns-3.

In *Proceedings of the 2014 Workshop on Ns-3*, WNS3 '14, pages 8 :1–8 :8, New York, NY, USA. ACM.

[Vangheluwe, 2000a] Vangheluwe, H. (2000a). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *IEEE International Symposium on Computer-Aided Control System Design, 2000. CACSD 2000*, pages 129–134.

[Vangheluwe, 2000b] Vangheluwe, H. (2000b). An introduction to multiparadigm modelling and simulation. In *Proceedings of the AIS'2002 Conference 9–20 (2002). 35.* Press.

[Varaiya et al., 2011] Varaiya, P. P., Wu, F. F., and Bialek, J. W. (2011). Smart Operation of Smart Grid : Risk-Limiting Dispatch. *Proceedings of the IEEE*, 99(1) :40–57.

[Varga and Hornig, 2008] Varga, A. and Hornig, R. (2008). An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools '08, pages 60 :1–60 :10, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[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 Communications and Networking Simulation Symposium*, pages 193–200, Pasadena, LA, USA. Society for Computer Simulation International/ACM.

[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., Pérez, J. B., and Prieta, F. d. l., editors, *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, number 9086 in Lecture Notes in Computer Science, pages 240–251. Springer International Publishing.

[Wolf, 2007] Wolf, W. (2007). News Briefs. *Computer*, 40(11) :104–105.

[Wu et al., 2001] Wu, H., Fujimoto, R., and Riley, G. (2001). Experiences parallelizing a commercial network simulator. In *Simulation Conference, 2001. Proceedings of the Winter*, volume 2, pages 1353–1360 vol.2.

[Xu et al., 2001] Xu, D., Riley, G., Ammar, M., and Fujimoto, R. (2001). Split protocol stack network simulations using the dynamic simulation backplane. In *Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings*, pages 158–165.

[Yeung et al., 2004] Yeung, G., Takai, M., Bagrodia, R., Mehrnia, A., and Daneshrad, B. (2004). Detailed OFDM Modeling in Network Simulation of Mobile Ad Hoc Networks. In *Proceedings of the Eighteenth Workshop on Parallel and Distributed Simulation*, PADS '04, pages 26–34, New York, NY, USA. ACM.

[Yoon and Kim, 2009] Yoon, S. and Kim, Y. B. (2009). A Design of Network Simulation Environment Using SSFNet. In *First International Conference on Advances in System Simulation, 2009. SIMUL '09*, pages 73–78.

[Zacharewicz, 2006] Zacharewicz, G. (2006). *Un environnement G-DEVS/HLA : application à la modélisation et simulation distribuée de workflow.* PhD thesis, Université de droit, d'économie et des sciences - Aix-Marseille III.

[Zeigler et al., 2000] Zeigler, B. P., Kim, T. G., and Praehofer, H. (2000). *Theory of Modeling and Simulation.* Academic Press, Inc., Orlando, FL, USA, 2nd edition.

[Zeigler, B. P. et al., 1998] Zeigler, B. P., Cho, H., Lee, J., and Sarjoughian, H. (1998). The DEVS/HLA distributed simulation environment and its support for predictive filtering.

[Zeng et al., 1998] Zeng, X., Bagrodia, R., and Gerla, M. (1998). GloMoSim : a library for parallel simulation of large-scale wireless networks. In *Twelfth Workshop on Parallel and Distributed Simulation, 1998. PADS 98. Proceedings*, pages 154–161.

[Zhou et al., 2003] Zhou, J., Ji, Z., Takai, M., and Bagrodia, R. (2003). Maya : A Multi-Paradigm Network Modeling Framework. In *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation*, PADS '03, pages 163–, Washington, DC, USA. IEEE Computer Society.

## Résumé

Modéliser et simuler (M&S) un système cyber-physique (SCP) peut nécessiter de représenter des éléments provenants de trois domaines d'expertise à la fois : systèmes physiques, systèmes d'informations et réseaux de communication (IP). Le simulateur universel disposant de toutes les compétences nécessaires n'existant pas, il est possible de regrouper des modèles issus des différentes communautés, à l'aide d'un multi-modèle. Les défis sont alors 1) intégrer toute l'hétérogénéité du multi-modèle (formalismes, représentations, implémentations), 2) intégrer des modèles IP de façon à ce qu'ils soient en capacité de représenter le transport de données applicatives produites par des modèles externes et 3) les intégrer de façon à ce qu'ils puissent se compléter, pour représenter ensemble les réseaux IP parfois hétérogènes d'un SCP. Pour parvenir à répondre à ces défis, nous nous inscrivons dans la continuité des travaux de M&S autour de MECSYCO, une plateforme de co-simulation basée sur la notion de wrapping DEVS. Nous proposons de définir un cadre général pour réussir à wrapper en DEVS des modèles IP, avec 1) une structuration des différents niveaux de problèmes pour l'intégration de modèles IP dans une co-simulation (délimitation des objectifs et contraintes du wrapping), et 2) une proposition de stratégie de wrapping DEVS de modèles IP et leurs simulateurs. Nous évaluerons notre approche à travers la démonstration de l'intégration de deux simulateurs IP populaires, et d'exemples concrets de M&S de SCP (avec notamment une interconnexion de modèles entre NS-3 et OMNeT++/INET, et une application industrielle utilisée par EDF R&D).



**Mots-clés:** système cyber-physique, réseau ip, devs, multi-modélisation, co-simulation, mecsyco, ns-3, omnet++



## Abstract

Modeling and simulation (M&S) of cyber-physical systems (CPS) can require representing components from three expertise fields : physics, information systems, and communication networks (IP). There is no universal simulator with all of the required skills, but we can gather and interconnect models provided by the communities, with a multi-model. The challenges are 1) integrating all heterogeneities in a multi-model (formalisms, representations, implementations), 2) integrating IP models in a way enabling them to represent the transport of application data produced by external models, and 3) integrating IP models in a way enabling them to complete each other, to be able to represent CPS heterogeneous IP networks. In order to meet these challenges, we relied our solution on the works around MECSYCO, a co-simulation platform based on the DEVS wrapping principle. We propose to define a comprehensive framework enabling to achieve DEVS wrapping of IP models, with 1) a structuration of different issue levels when integrating IP models in a co-simulation (goals and constraints of the wrapping) and 2) a proposition of a DEVS wrapping strategy for IP models and their simulators. We propose some evaluations of our approach, through the integration of two popular IP simulators, and concrete examples of CPS M&S (inter alia, with an example of a models interconnection between NS-3 and OMNeT++/INET, and an industrial application used by EDF R&D).



**Keywords:** cyber-physical system, ip network, devs, multi-modeling, co-simulation, mecsyco, ns-3, omnet++

https://julien.vaubourg.com