INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2020IPPAS002

Thèse de doctorat

Application de l'ingénierie dirigée par les modèles à la conception de Smart Grids : Approche par cosimulation avec FMI

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626 Ecole Doctorale de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Palaiseau, le 7 mai 2020, par

DAVID OUDART

Composition du Jury :

Antoine BEUGNARD Professeur, IMT Atlantique, IRISA, Université Bretagne-Loire	Rapporteur
Yamine AïT AMEUR Professeur, ENSEEIHT, IRIT, Toulouse	Rapporteur
Virginie GALTIER Associate Professor, CentraleSupélec, LORIA, Univ. de Lorraine	Examinatrice
Stéphane VIALLE Professeur, CentraleSupélec, LRI, Univ. Paris Saclay	Examinateur
Jérôme CANTENOT Ingénieur de recherche, EDF R&D	Examinateur
Sophie CHABRIDON Directrice d'études, Télécom SudParis, SAMOVAR, IP Paris	Directrice de thèse
Frédéric BOULANGER Professeur, CentraleSupélec, LRI, Univ. Paris Saclay	Co-directeur

Résumé

Les *Smart Grids* sont des systèmes cyberphysiques qui interfacent les réseaux électriques avec les technologies de l'information et de la communication afin de les surveiller, d'automatiser la prise de décision et d'équilibrer la production avec la consommation. Nous souhaitons utiliser la simulation pour évaluer et comparer facilement plusieurs solutions avant un déploiement dans un environnement réel. L'objectif de cette thèse est ainsi de proposer des outils et méthodes pour modéliser et simuler un *Smart Grid* dans un contexte industriel. Nous avons identifié deux problématiques principales : Comment combiner les modèles hétérogènes d'un Smart Grid pour le simuler ? Comment assurer la cohérence entre les modèles produits par différents intervenants lors de la conception d'un *Smart Grid* ? Pour répondre à ces problématiques, nous proposons une approche de cosimulation, en utilisant la norme *Functional Mockup Interface* (FMI). FMI fournit un format standard d'interface pour un composant de simulation, appelé *Functional Mockup Unit* (FMU), constitué d'un modèle et de son simulateur. L'une des limitations actuelles de FMI est son incompatibilité avec la manipulation de signaux discrets entraînant notamment l'absence de simulateurs Télécoms compatibles. Pour pallier cette limitation, nos deux premières contributions sont la proposition d'une méthode pour permettre l'échange de signaux discrets entre plusieurs FMUs, et d'une extension du logiciel de simulation de télécommunications *OMNeT++* implémentant cette méthode, appelée **fmi4omnetpp**. Une troisième contribution est la réalisation de l'environnement outillé ***Smart Grid Simulation Framework***, qui automatise un certain nombre de tâches répétitives afin d'assurer la cohérence entre différents modèles de simulation. Cet environnement est constitué d'un langage spécifique de modélisation d'une unité de cosimulation, appelé **CosiML** pour *Cosimulation Modeling Language*, d'un outil implémentant différentes règles de transformation pour générer les fichiers et les scripts d'exécution d'une cosimulation, d'un langage spécifique de modélisation de l'architecture fonctionnelle d'un Smart Grid, appelé **SGridML** pour *Smart Grid Modeling Language*, de deux langages spécifiques de modélisation, appelés **AllocationML** pour *Allocation Modeling Language* et **CatalogML** pour *Catalog Modeling Language* permettant de définir une transformation depuis un modèle écrit en **SGridML** vers un modèle écrit en **CosiML**, et enfin d'un outil implémentant la transformation depuis trois modèles écrits en **SGridML**, **AllocationML** et **CatalogML** vers un modèle écrit en **CosiML**. Enfin, une quatrième contribution est la formalisation de la démarche itérative de conception dans laquelle s'inscrit la cosimulation d'un *Smart Grid*, et la façon d'y intégrer notre environnement *Smart Grid Simulation Framework*. Pour cela, nous explicitons les différentes étapes de la démarche et le rôle des acteurs de la conception, puis nous présentons son application sur un cas d'étude réel pour lequel nous utilisons ***Smart Grid Simulation Framework***.

Mots-clés : Cosimulation, FMI, IDM, smart grid, système cyber-physique

Abstract

Smart Grids are cyber-physical systems that interface power grids with information and communication technologies to monitor them, automate decision making and balance production with consumption. We want to use simulation to easily evaluate and compare several solutions before deployment in a real environment. The objective of this thesis is thus to propose tools and methods to model and simulate a Smart Grid in an industrial context. We have identified two main issues : How to combine heterogeneous models of a Smart Grid to simulate it ? How to ensure consistency between the models produced by different stakeholders during the design of a Smart Grid ? To address these issues, we propose a cosimulation approach, using the Functional Mockup Interface (FMI) standard. FMI provides a standard interface format for a simulation component, called Functional Mockup Unit (FMU), consisting of a model and its simulator. One of the current limitations of FMI is its incompatibility with the manipulation of discrete signals, resulting in the absence of compatible Telecom simulators. To overcome this limitation, our first two contributions are the proposal of a method to allow the exchange of discrete signals between several FMUs, and an extension of the OMNeT++ telecommunications simulation software implementing this method, called **fmi4omnetpp**. A third contribution is the development of the **Smart Grid Simulation Framework** tooled environment, which automates a number of repetitive tasks in order to ensure consistency between different simulation models. This environment consists of a domain specific language for modeling a cosimulation unit, called **CosiML** for Cosimulation Modeling Language, a tool implementing different transformation rules to generate the files and scripts for executing a cosimulation, a specific language for modeling the functional architecture of a Smart Grid, called **SGridML** for Smart Grid Modeling Language, two specific modeling languages, called **AllocationML** for Allocation Modeling Language and **CatalogML** for Catalog Modeling Language allowing to define a transformation from a model written in **SGridML** to a model written in **CosiML**, and finally a tool implementing the transformation from three models written in **SGridML**, **AllocationML** and **CatalogML** to a model written in **CosiML**. Finally, a fourth contribution is the formalization of an iterative design approach for the cosimulation of a Smart Grid, and how to integrate our **Smart Grid Simulation Framework** into it. To do so, we explain the different steps of the approach and the role of the actors involved in the design process, then we present its application to a real case study for which we use our **Smart Grid Simulation Framework**.

KeyWords : Cosimulation, FMI, MDE, smart grid, cyber-physical system

Table des matières

Table des matières v

Liste des figures ix

Liste des tableaux xi

Liste des publications xiv

1 Introduction 1

1.1 Des réseaux électriques aux Smart Grids 1

1.1.1 Fonctionnement du réseau électrique 2

1.1.2 Les technologies de l'information au service de la modernisation du réseau . . . 3

1.2 Enjeux et difficultés de la simulation industrielle des Smart Grids 3

1.2.1 Un contexte industriel 3

1.2.2 Un simulateur de Smart Grid 4

1.3 Problématiques des travaux et contributions apportées 5

1.3.1 Identification des problématiques 5

1.3.2 Contributions 6

1.4 Plan de la thèse 7

I Détail de la Problématique et État de l'art 9

2 Contexte et défis posés par la simulation des Smart Grids 11

2.1 Notions fondamentales de la modélisation et simulation . . . 11

2.1.1 Modèle . . . 11

2.1.2 Langage de modélisation . . . 12

2.1.3 Simulation informatique . . . 13

2.1.4 Modèle de calcul . . . 14

2.2 Défis posés par la simulation des Smart Grid . . . 15

2.2.1 Un système complexe et cyber-physique . . . 15

2.2.2 Intégration dans une démarche industrielle . . . 16

2.2.3 Modélisation hétérogène du comportement . . . 18

2.3 Conclusion . . . 19

3 Cohérence entre modèles 21

3.1 Ingénierie Dirigée par les Modèles, vers des modèles productifs . . . 22

3.1.1 Principes . . . 22

3.1.2 IDM et ingénierie logicielle . . . 23

3.1.3 Métamodèles et *Domain Specific Language* . . . 23

3.1.4 Transformations de modèles . . . 26

3.2 Cohérence des modèles à différents niveaux d'abstraction . . . 28

3.2.1 Architecture d'Entreprise . . . 28

3.2.2 *Model Driven Architecture* . . . 30

3.2.3 Application aux CPS et aux Smart Grids . . . 32

3.3 Approches de simulation de modèles hétérogènes . . . 35

3.3.1 Formalismes hybrides . . . 36

3.3.2 Transformation de modèles . . . 36

3.3.3 Composition de modèles . . . 37

3.3.4 Cosimulation . . . 37

3.3.5 Le cas du *Software* et *Hardware-in-the-loop* . . . 38
3.3.6 Bilan . . . 38
3.4 Conclusion du chapitre . . . 40

4 FMI et la cosimulation pour les Smart Grids 43
4.1 Vocabulaire et définitions de la cosimulation . . . 44
4.2 Technologies et standards de cosimulation . . . 45
4.2.1 Limites sur les interfaces propriétaires . . . 45
4.2.2 *High Level Architecture* . . . 46
4.2.3 *Functional Mockup Interface* . . . 47
4.2.4 Comparaison et limites . . . 49
4.3 Outils compatibles avec FMI pour cosimuler le Smart Grid . . . 50
4.3.1 Simuler le réseau électrique avec FMI . . . 51
4.3.2 Simuler le système d'information avec FMI . . . 51
4.3.3 Simuler le réseau de télécommunications avec FMI . . . 53
4.4 Conclusion du chapitre . . . 55

II Contribution : Un Environnement de Cosimulation de Smart Grid 57

5 Mise en place du cadre des travaux : une démarche de simulation d'un Smart Grid avec FMI 59
5.1 Choix de se limiter au standard FMI . . . 60
5.2 Formalisation d'une démarche pour cosimuler un Smart Grid . . . 60
5.2.1 Positionnement dans le processus de conception . . . 60
5.2.2 Les acteurs . . . 61
5.2.3 Les étapes de la démarche . . . 62
5.2.4 Contraintes et difficultés dans l'exécution de la démarche . . . 65
5.3 Gérer les échanges discrets avec FMI . . . 66
5.3.1 Problème et définitions . . . 66

5.3.2 Solution proposée . . . 67
5.3.3 Discussion . . . 71
5.4 Conclusion du chapitre . . . 72

6 Modéliser et exécuter un scénario de cosimulation de Smart Grid 75

6.1 fmi4omnetpp : Cosimuler le réseau de télécommunications avec OMNeT++ et FMI . . . 76
6.1.1 Rôle du simulateur télécoms au sein de la simulation du Smart Grid . . . 76
6.1.2 Étendre le fonctionnement d'OMNeT++ . . . 77
6.1.3 Principe et structure de fmi4omnetpp . . . 79
6.2 CosiML : un DSL pour la cosimulation de CPS . . . 81
6.2.1 Objectif de CosiML . . . 81
6.2.2 Métamodèle de CosiML . . . 82
6.3 Génération des artefacts de cosimulation FMI . . . 86
6.3.1 Processus . . . 86
6.3.2 Générateurs d'adaptateur . . . 87
6.3.3 Choix de DACCOSIM en tant que *master* de cosimulation FMI . . . 92
6.3.4 Générateur de l'exécutable de la cosimulation . . . 93
6.4 Conclusion du chapitre . . . 94

7 L'architecture fonctionnelle au service de la simulation d'un Smart Grid 95

7.1 Lier architecture fonctionnelle et simulation du Smart Grid . . . 96
7.1.1 Trois nouveaux DSLs . . . 96
7.1.2 Comportements fonctionnels et comportements de transmission . . . 97
7.2 SGridML : Un DSL pour modéliser l'architecture fonctionnelle de la simulation du Smart Grid . . . 98
7.2.1 Métamodèle de SGridML . . . 98
7.2.2 Syntaxe concrète . . . 100
7.3 Transformation de modèle entre SGridML et CosiML . . . 100
7.3.1 AllocationML et CatalogML : deux DSL pour définir la transformation . . . 100

7.3.2 Génération d'un modèle de cosimulation CosiML 101

7.4 Conclusion du chapitre 103

8 Présentation de *Smart Grid Simulation Framework* sur un cas d'étude et observations 107

8.1 Contexte du cas d'étude 108

8.1.1 Présentation du problème 108

8.1.2 Utiliser le *Smart Grid Simulation Framework* 109

8.2 Déroulement de la démarche 110

8.2.1 Phase 1 : modélisation du système de l'île de Sein 110

8.2.2 Phase 2 : Configurer la cosimulation FMI 117

8.2.3 Phase 3 : Analyser les résultats 119

8.3 Observations 121

8.4 Conclusion du chapitre 123

III Conclusions et Perspectives 125

9 Conclusions et perspectives 127

9.1 Conclusion générale 127

9.1.1 Rappel du contexte et des objectifs de nos travaux 127

9.1.2 Résumé des contributions 128

9.2 Perspectives 129

Table des figures

2.1 Démarche de modélisation et simulation itérative jusqu'à obtention d'un modèle valide 16

2.2 Processus de raffinement successifs depuis la spécification d'un système cible vers un modèle exécutable de son comportement . . . 17

3.1 Relations entre modèle, langage et métamodèle . . . 24

3.2 Pyramide de modélisation de l'OMG . . . 24

3.3 Exemple des 4 niveaux hiérarchiques de métamodélisation . . . 25

3.4 Liens entre modèles et métamodèles dans le cas d'une transformation de modèles *one-to-one* . . . 27

3.5 Les 4 vues de l'urbanisation des SI . . . 30

3.6 Composition et transformations de modèles dans l'approche MDA . . . 31

3.7 SGAM, et ses différentes perspectives . . . 33

3.8 Liens entre SGAM et MDA dans l'outil *SGAM Toolbox* . . . 34

3.9 Représentation simplifée en UML du PSAL . . . 35

4.1 Constitution d'une unité de cosimulation . . . 45

4.2 Architecture logicielle avec HLA . . . 46

4.3 Différence entre FMU-ME et FMU-CS . . . 48

4.4 Fréquence de citation de quelques uns des principaux logiciels de simulation télécoms parmi les publications de l'IEEE . . . 54

5.1 Enchaînement des phases de cosimulation d'un Smart Grid 63

5.2 Décisions d'itération en fonction de l'analyse des résultats de cosimulation 65

5.3 Utilisation d'un signal discret en tant que signal FMI 67

5.4 Encodage d'un signal x discret en deux signaux u_{info} et u_{sync} continus 69

5.5 Décodage de deux signaux v_{info} et v_{sync} continus en un signal y discret 70

5.6 Différence entre un signal encodé produit par un modèle et le signal reçu par un autre modèle après échange FMI 71

5.7 Les évènements trop proches sont manqués 72

6.1 Ajout d'une FMU télécoms pour évaluer l'impact de la transmission dans la simulation du système 76

6.2 Modules simples et composés 77

6.3 Boucle principale d'exécution de OMNeT++ 78

6.4 Boucle principale d'exécution de OMNeT++ 80

6.5 CosiML simplifié 83

6.6 SimulationUnit détaillée 84

6.7 Processus de génération des artefacts de simulation 87

6.8 Processus du *script de simulation* permettant d'obtenir les résultats de simulation . . . 88

7.1 Métamodèle Ecore de SGridML 98

7.2 Métamodèle de AllocationML 101

7.3 Métamodèle de UnitCatalogML 101

7.4 Modèles nécessaires à la génération du modèle CosiML, et liens de dépendance 102

7.5 Exemple de connexion impliquée dans trois unités de simulation 104

7.6 Exemple de scénario de cosimulation généré à partir d'une connexion 104

8.1 `Connection` entre les `Function` du modèle SGridML de l'île de Sein 113

8.2 `SimuationUnit` instanciées dans le modèle CatalogML 114

8.3 Capture d'écran du logiciel Dymola et du modèle de *loadflow* du réseau électrique de l'île de Sein 116

8.4 Évolution de la puissance consommée et de la puissance photovoltaïque disponible . . 117

8.5 Modèle OMNeT++ de TelecomUnit 118

8.6 Diagramme des classes Java du modèle de EMSUnit 119

8.7 Évolution de la puissance consommée et de la puissance photovoltaïque disponible . . 120

Liste des tableaux

2.1 Différents cas d'hétérogénéité de la modélisation 18

3.1 Comparaison des alternatives de simulation de modèles hétérogènes 40

4.1 Description des principales fonctions C de FMI, implémentées par chaque FMU 49

7.1 Les différentes possibilités de transformation d'un élément `Connection` vers des éléments `Port` et `Link` 103

8.1 Correspondance entre les étapes de la démarche et les outils de *Smart Grid Simulation Framework* 110

8.2 `Function`, `Input` et `Output` du modèle SGridML de l'île de Sein 112

8.3 Liste des `Mapping`s du modèle d'allocation AllocationML de l'Île de Sein 114

Liste des publications

Les travaux publiés dans le cadre du déroulement de cette thèse sont listés ci-dessous :

"*A Model Based Toolchain for the Cosimulation of Cyber-Physical Systems with FMI*".
Article accepté en tant que *long paper* pour MODELSWARD 2020 (8th Conference on Model-Driven Engineering and Software Development) [1].
Sélectionné comme candidat au prix du meilleur article étudiant (*best student paper award*).

"*An Approach to Design Smart Grids and their IT System by Cosimulation*".
Article publié en tant que *short paper* et présenté à MODELSWARD 2019 (7th Conference on Model-Driven Engineering and Software Development) [Oudart et al., 2019].

"Démarche de conception d'un réseau électrique intelligent et de son système d'information par cosimulation".
Article publié et présenté à CIEL 2018 (7ieme conférence en Ingéniérie du Logiciel) [Oudart et al., 2018].

"*Model Driven Design of IT Systems for Smart Grids*".
Article publié et présenté au Doctoral Symposium de MODELS 2017 (20th International Conference on Model Driven Engineering Languages and Systems) [Oudart, 2017].

1. Au moment de la rédaction de ce manuscrit, l'article n'est pas encore publié. La référence sera ajoutée dans la version de publication de ce document

1 Introduction

Sommaire

1.1 Des réseaux électriques aux Smart Grids 1
1.1.1 Fonctionnement du réseau électrique 2
1.1.2 Les technologies de l'information au service de la modernisation du réseau . . . 3
1.2 Enjeux et difficultés de la simulation industrielle des Smart Grids 3
1.2.1 Un contexte industriel 3
1.2.2 Un simulateur de Smart Grid 4
1.3 Problématiques des travaux et contributions apportées 5
1.3.1 Identification des problématiques 5
1.3.2 Contributions 6
1.4 Plan de la thèse 7

Ces travaux de thèse sont réalisés dans le cadre d'une thèse CIFRE, en collaboration avec EDF R&D, entité de recherche et développement du groupe EDF SA. EDF est le premier producteur et fournisseur d'électricité en France, et est notamment composé de l'entreprise RTE [1] gestionnaire du réseau de transport français, et ENEDIS [2] exploitant des réseaux de distribution français. Ainsi EDF R&D est l'un des acteurs principaux de l'évolution des réseaux électriques, et fortement impliqué dans les études de conception et déploiement des smart grids.

1.1 Des réseaux électriques aux Smart Grids

Après plusieurs décennies de lente évolution, les réseaux électriques connaissent un développement de grande envergure avec l'émergence des nouveaux usages (véhicules électriques), des nouvelles sources de production renouvelables et décentralisées (éolien, photovoltaïque), ainsi que la multiplication des acteurs due à la libéralisation des marchés de l'énergie. En soutien de ce développement, les réseaux

1. Réseau de Transport d'Électricité
2. Anciennement ERDF, Électricité Réseau Distribution France

se modernisent et introduisent massivement les technologies de l'information et de la communication, introduisant le concept de réseaux électriques intelligents, dits "*Smart Grids*". La présentation du fonctionnement du réseau électrique et de son intelligence est nécessaire à la compréhension des enjeux de cette thèse. Les explications ci-dessous proviennent notamment de deux ouvrages, le premier rédigé par plusieurs auteurs universitaires et industriels [Sabonnadière and Hadjsaïd, 2012], le second par différents chercheurs et experts de EDF [Électricité de France, 2018].

1.1.1 Fonctionnement du réseau électrique

L'un des principaux enjeux du réseau électrique est **l'équilibre permanent de l'offre et de la demande**, c'est-à-dire la correspondance entre l'énergie injectée sur le réseau par les unités de production et l'énergie consommée par ses utilisateurs. En effet, l'électricité ne se stockant pas encore à grande échelle et à long terme, cet équilibre est l'un des principes fondamentaux du bon fonctionnement des réseaux électriques. Cela requiert des ajustements constants, traditionnellement réalisés via la gestion de la demande à distance (comme la commande des ballons d'eau chaude), et le pilotage des moyens de production centralisés.

Le fonctionnement du système électrique global en France est fondé historiquement sur quatre segments :

- **la production**, assurée majoritairement par des unités de grande puissance telles que les centrales nucléaires, installées en un nombre restreint de lieux stratégiques ;
- **le réseau de transport,** acheminant la production vers les centres de consommation, et bénéficiant d'une gestion hiérarchisée et centralisée ;
- **les réseaux de distribution**, interfaçant le réseau de transport et les consommateurs. Jusqu'à présent, ceux-ci étaient caractérisés par la faible présence de sources de production locales, et donc par un sens de circulation de l'énergie unidirectionnel, depuis les "postes sources" (nœuds d'interconnexion avec le réseau de transport) vers les consommateurs ;
- **les consommateurs**, considérés comme des contraintes de charges pas ou peu pilotables.

Dans ce fonctionnement, l'électricité circule du "haut vers le bas", c'est-à-dire du segment de production (tension autour de 400 kV) jusqu'au segment de consommation (tension usuelle de 230 V) via le segment de transport (400 - 90 kV) puis de distribution (20 kV - 230 V).

Les évolutions technologiques, ainsi que la prise de conscience de la question environnementale au sein de la société moderne a contribué à l'essor des énergies renouvelables et de nouveaux usages et équipements de production, dont le développement a un impact fort sur le fonctionnement de nos réseaux traditionnels, sur tous les segments.

Les énergies renouvelables sont destinées à être raccordées au réseau de transport, comme les grandes fermes éoliennes ou photovoltaïques, au réseau de distribution et même chez le consommateur, engendrant un nouveau système de production décentralisé. Cette décentralisation, couplée à une nature le plus souvent intermittente, implique un besoin de gestion grandissant ainsi que des moyens de stockage

tels que des batteries.

Le segment des consommateurs subit également une profonde mutation, notamment grâce à l'apparition des compteurs communicants, indispensables à l'insertion des moyens de production et de stockage domestiques. Jusqu'alors considérés comme "passifs", les consommateurs participent désormais à la gestion des contraintes du réseau en offrant des fonctionnalités de modulation de charge, mais aussi de nouvelles contraintes de gestion. C'est d'autant plus vrai avec le développement attendu des véhicules électriques, nouvel usage gourmand en énergie mais offrant un autre levier de flexibilité par sa capacité de stockage.

Ces transformations impactent les performances du réseau électriques, les moyens de gestion actuels n'étant plus adaptés. De plus, en cas de non respect de l'équilibre offre-demande ou des plages de fonctionnement des équipements, des accidents peuvent se produire, avec un risque de cascade pouvant conduire au *black-out*.

1.1.2 Les technologies de l'information au service de la modernisation du réseau

Le respect des contraintes de fonctionnement du réseau électrique ainsi que l'équilibre entre l'offre et la demande est de plus en plus difficile avec les solutions actuelles de contrôle-commande local sur les équipements. L'intégration des EnRV multiplie les sources de production à surveiller et piloter, crée des pics de tension difficiles à localiser, inverse parfois le sens de circulation de l'électricité. D'un point de vue commercial, le comptage devient un nouveau défi, les utilisateurs pouvant soutirer mais également injecter de l'énergie sur le réseau. La gestion de ce nouveau réseau électrique demande le traitement rapide et automatisé d'un nombre de plus en plus important de données.

1.2 Enjeux et difficultés de la simulation industrielle des Smart Grids

1.2.1 Un contexte industriel

Le réseau électrique est l'un des systèmes physiques les plus larges et complexes conçus par l'être humain. En effet, la réalisation de la plupart des activités humaines et industrielles dépendent de son fonctionnement, ce qui en fait l'un des systèmes avec les contraintes de qualité de service les plus importantes (fonctionnement 24h sur 24, 7 jours sur 7). Ainsi, chaque opération, chaque évolution du réseau doit être fortement étudiée et analysée avant d'être réalisée, car toute défaillance peut occasionner de lourdes conséquences économiques, mais également humaines.

Le secteur de l'énergie met en place et fait évoluer le réseau électrique depuis plusieurs dizaines d'années, et dispose d'un fort retour d'expérience sur ses comportements électrotechniques et sa gestion, favorisant les partenariats entre acteurs spécialisés dans leur domaine. Des experts du réseau de distribution, du réseau de transport, de la production, du contrôle-commande, etc. ont l'habitude de

travailler ensemble à l'amélioration de la fiabilité et de la résilience du réseau électrique.

Or, l'introduction des sources de production renouvelables et décentralisées, ainsi que les technologies de l'information et de la communication (**ICT**) sur le réseau fait intervenir de plus en plus de composants, toujours plus couplés entre eux et formant un système complexe. Cette réalité intensifie l'aspect transverse des études, tout en faisant intervenir de nouveaux domaines de spécialisation et donc de nouveaux collaborateurs avec leurs propres outils et méthodes.

Les contraintes de respect des délais et du budget des projets industriels favorisent les solutions les moins coûteuses en investissement d'effort et de temps. Les profils multi-spécialisés (les fameux "*moutons à cinq pattes*") étant rares, on observe les conséquences suivantes :

— La tendance est à la réutilisation des compétences, la formation étant un processus long et coûteux.
— Les entreprises font appel à des prestataires externes lorsqu'elles ne disposent pas des compétences en interne.
— Les tâches et l'avancement du travail sont parallélisés autant que possible entre les différents collaborateurs.

Pour les entreprises, l'une des difficultés des études sur les nouveaux réseaux électriques intelligents, ou Smart Grids, réside donc dans la mise en relation d'un nombre important de collaborateurs aux compétences, connaissances et outils disjoints. Ceux-ci pouvant également provenir de différentes entreprises, des contraintes de respect de la propriété intellectuelle rendent ces échanges d'autant plus difficiles.

1.2.2 Un simulateur de Smart Grid

La simulation permet de réaliser des analyses prédictives du comportement d'un système à déployer. Elle permet de mesurer les conséquences d'une modification de l'existant sur le fonctionnement d'un système, et de faire l'économie de déploiements et de tests matériels coûteux en temps et en argent. Sachant que les études industrielles requièrent rarement un degré d'exactitude parfait, la simulation qui donne des résultats approchés mais avec moins d'efforts que les méthodes de résolution analytiques, est la pratique généralement retenue pour parvenir à une solution satisfaisante.

Néanmoins, la simulation ne permet que de prédire et d'évaluer le comportement d'une solution particulière. L'objectif d'obtention d'une solution finale impose une méthodologie itérative dans laquelle la conception de la solution évolue progressivement au cours des simulations successives, jusqu'à l'obtention d'une solution acceptable.

Par ailleurs, les techniques de modélisation des réseaux électriques étant bien maîtrisées dans le monde industriel, la difficulté de la simulation des Smart Grids provient de l'intégration du comportement des ICT [3]. La combinaison de composants physiques et numériques est un phénomène qui

3. *Information and Communication Technologies*

touche de nombreux autres secteurs industriels, comme l'automobile avec le véhicule autonome, la santé avec la supervision médicale, la robotique, et l'automatisation en général. On nomme cette catégorie de systèmes des Systèmes Cyber-Physiques – *Cyber-Physical Systems* (CPS) en anglais. Pour beaucoup [Mosterman and Zander, 2016, Hermann et al., 2016, Suri et al., 2017], l'étude et la conception des CPS font partie des défis de "l'Industrie 4.0". La simulation des Smart Grids doit faire face aux mêmes problématiques que la simulation des CPS, principalement le couplage entre des modèles aux lois d'évolution différentes :

— continues pour les composants physiques du système ;
— discrètes pour les composants numériques.

1.3 Problématiques des travaux et contributions apportées

1.3.1 Identification des problématiques

L'apparition d'un nouveau besoin lié au réseau électrique donne lieu à une étude, qui permettra de définir la solution à implémenter pour répondre à ce besoin. Nous souhaitons utiliser la simulation pour évaluer et comparer plusieurs solutions entre elles. De ce fait, notre problématique générale pourrait s'énoncer ainsi :

Problématique générale : Quels outils et méthodes mettre en place pour modéliser et simuler un Smart Grid dans un contexte industriel ?

La simulation complète d'un Smart Grid demande de pouvoir modéliser et simuler les comportements électriques, les comportements de traitement de l'information, et les comportements de télécommunication. Or ces comportements doivent être analysés tous ensemble afin de détecter les influences des uns sur les autres, et identifier les comportements émergents. La simulation d'un Smart Grid implique donc le développement de modèles hétérogènes liés entre eux, et réalisés par plusieurs collaborateurs aux connaissances différentes. Ainsi, de notre problématique générale découlent deux problématiques plus précises :

Problématique 1. Comment rassembler les modèles hétérogènes d'un Smart Grid pour le simuler ?

Problématique 2. Comment s'assurer de la cohérence entre les modèles produits par les différents collaborateurs lors de la conception d'un Smart Grid ?

Enfin, nos outils et méthodes pour simuler un Smart Grid doivent pouvoir s'intégrer à un contexte industriel et à un système potentiellement large et complexe. Cela fournit un ensemble de critères que le choix de nos outils et méthodes doivent prendre en compte et optimiser :

Critère 1 (coût) : la capacité à optimiser les coûts et l'effort d'utilisation.

Critère 2 (itération) : la capacité à s'intégrer dans une démarche itérative,

Critère 3 (collaboration) : la capacité à faire collaborer plusieurs personnes/équipes aux compétences techniques différentes,

Critère 4 (protection intellectuelle) : la capacité à garantir la propriété intellectuelle lors de la collaboration avec des prestataires externes,

Critère 5 (scalabilité) : la capacité à s'adapter à un changement d'échelle.

1.3.2 Contributions

Pour répondre à ces différentes problématiques, nous proposons d'utiliser une approche de cosimulation, en utilisant la norme *Functional Mockup Interface* (FMI). FMI fournit un format standard d'interface pour un composant de simulation constitué d'un modèle et de son simulateur, qui peut ainsi échanger des données avec d'autres composants respectant la même norme. Ce format de composant est appelé *Functional Mockup Unit* (FMU). Ainsi, les comportements d'un Smart Grid peuvent être répartis entre plusieurs modèles de simulation liés par leur interface, chacun fourni avec un simulateur approprié au format FMU. Les FMU sont alors exécutées au sein d'un cosimulateur FMI, qui synchronise l'exécution et la simulation de chaque modèle et réalise les échanges de données nécessaires.

L'une des limitations connues de FMI est son incompatibilité avec la manipulation de signaux discrets, qui apparaissent généralement dans les modèles de simulation des comportements informatiques et de communication, entraînant notamment l'absence de simulateurs télécoms compatibles. *Deux premières contributions* sont donc de proposer :

— **une méthode pour permettre l'échange de signaux discrets** entre plusieurs FMU,
— **une extension du logiciel de simulation de télécommunication OMNeT++** implémentant cette méthode, appelée `fmi4omnetpp`, permettant d'exporter un modèle de simulation au format FMU pour la cosimulation.

Une troisième contribution est de **développer un environnement outillé de simulation de Smart Grid**, qui automatise un certain nombre de tâches répétitives et valide dans une certaine mesure la cohérence entre différents modèles de simulation. Cet environnement, nommé *Smart Grid Simulation Framework*, est construit à partir de la suite d'outils de *Eclipse Modeling Framework*, et est constitué de :

— un langage spécifique de modélisation d'une unité de cosimulation, appelé *Cosimulation Modeling Language* (CosiML) et permettant de valider les interfaces de plusieurs modèles de simulation en fonction de leur contraintes de couplage,
— un outil implémentant différentes règles de transformation, permettant de générer les fichiers et les scripts nécessaires à l'exécution d'une cosimulation à partir d'un modèle écrit en CosiML et des modèles de simulation référencés,
— un langage spécifique de modélisation de l'architecture fonctionnelle d'un Smart Grid, appelé *Smart Grid Modeling Language* (SGridML), permettant de représenter de manière abstraite les

différents comportements à simuler d'un Smart Grid,
— deux langages spécifiques de modélisation, appelés *Allocation Modeling Language* (AllocationML) et *Catalog Modeling Language* (CatalogML) permettant de définir une transformation depuis un modèle écrit en SGridML vers un modèle écrit en CosiML,
— un outil implémentant la transformation depuis trois modèles écrits en SGridML, AllocationML et CatalogML vers un modèle écrit en CosiML.

Nous avons choisi d'être en particulier compatibles avec un certain nombre d'outils existants : le logiciel DACCOSIM pour l'exécution de la cosimulation, le logiciel de simulation des télécommunications OMNeT++ (au travers de notre extension), et la bibliothèque JavaFMI permettant l'export d'un code Java vers FMU.

Enfin, *une quatrième contribution* est **la formalisation de la démarche itérative de conception dans laquelle s'inscrit la simulation d'un Smart Grid**, et la façon d'y intégrer notre environnement *Smart Grid Simulation Framework*. Pour cela, nous explicitons les différentes étapes de la démarche et le rôle des acteurs de la conception, puis nous présentons son application sur un cas d'étude réel pour lequel nous utilisons *Smart Grid Simulation Framework*.

1.4 Plan de la thèse

Ce document est constitué de deux parties principales. La première partie explicite les défis de la thèse et dresse un état de l'art de la simulation des Smart Grids et plus généralement des systèmes cyber-physiques. La seconde partie présente en détail nos contributions, en présentant notamment individuellement chaque outil de notre environnement de simulation *Smart Grid Simulation Framework*.

La **partie I**, intitulée **"Détail de la problématique et état de l'art"**, est composée de trois chapitres :

Le Chapitre 2, "Contexte et défis posés par la simulation des Smart Grids", définit les notions fondamentales de modélisation et de simulation, et explicite les défis posés par le sujet de thèse en regard de ces notions.

Le Chapitre 3, "Cohérence entre modèles hétérogènes", positionne la cosimulation par rapport aux différentes techniques de mise en cohérence de modèles hétérogènes. Dans ce chapitre, nous définissons notamment les disciplines d'Architecture d'Entreprise et d'Ingénierie Dirigée par les Modèles. Nous concluons sur le manque d'une approche garantissant la cohérence entre la conception fonctionnelle d'un système et ses modèles de simulation.

Le Chapitre 4, "FMI et la cosimulation pour les Smart Grids", positionne le standard FMI par rapport aux autres méthodes d'interaction logicielles, et évalue sa compatibilité avec les outils conventionnels de simulation pouvant être utilisés dans la simulation d'un Smart Grid.

La **partie II**, intitulée "**Contributions : Un environnement de cosimulation de Smart Grid**, est composée de quatre chapitres :

Le Chapitre 5, "Mise en place du cadre de simulation d'un Smart Grid avec FMI", introduit notre contribution en présentant les raisons de notre choix de la cosimulation et de FMI par rapport à l'état de l'art, et en présentant les étapes de la démarche de simulation d'un Smart Grid avec FMI. Dans ce chapitre, nous présentons également notre contribution permettant d'échanger des signaux discrets avec FMI.

Le Chapitre 6, "Modéliser et exécuter un scénario de cosimulation de Smart Grid", présente les outils que nous avons développés afin de permettre la simulation d'un Smart Grid à partir de plusieurs modèles de simulation. Nous présentons notamment notre langage *Cosimulation Modeling Language* (CosiML) et les générateurs associés, permettant de générer les fichiers et les scripts nécessaires à la création des FMU manquantes et l'exécution de la cosimulation.

Le Chapitre 7, "L'architecture fonctionnelle au service de la simulation d'un Smart Grid", présente l'intérêt de la modélisation de l'architecture fonctionnelle d'un Smart Grid pour nos problématiques. Nous décrivons le langage *Smart Grid Modeling Language* (SGridML), ainsi que les langages *Allocation Modeling Language* (AllocationML) et *Catalog Modeling Language* (CatalogML) permettant de définir une transformation vers CosiML, et l'intérêt d'une telle transformation.

Le Chapitre 8, "Présentation et validation de *Smart Grid Simulation Framework* sur un cas d'étude", illustre l'application de la démarche et de *Smart Grid Simulation Framework* sur le cas d'étude du réseau électrique de l'Île de Sein. Il présente notamment comment l'outil s'intègre à notre démarche.

Enfin, la **partie III**, intitulée **"Conclusions et perspectives"** ne contient qu'un chapitre :

Le Chapitre 9, "Conclusions et perspectives", résume les contributions de cette thèse et vérifie en quoi celles-ci répondent aux problématiques du sujet. Puis, nous présentons différentes perspectives d'utilisation et d'évolution de ces travaux.

Première partie

Détail de la Problématique et État de l'art

2

Contexte et défis posés par la simulation des Smart Grids

Sommaire

2.1 Notions fondamentales de la modélisation et simulation 11
2.1.1 Modèle 11
2.1.2 Langage de modélisation 12
2.1.3 Simulation informatique 13
2.1.4 Modèle de calcul 14
2.2 Défis posés par la simulation des Smart Grid 15
2.2.1 Un système complexe et cyber-physique 15
2.2.2 Intégration dans une démarche industrielle 16
2.2.3 Modélisation hétérogène du comportement 18
2.3 Conclusion 19

Nous avons cité en introduction la problématique générale de nos travaux de recherche, qui est de pouvoir simuler un Smart Grid dans un contexte industriel. On détaille dans ce chapitre les éléments nécessaires à la compréhension des différents défis à relever pour atteindre cet objectif. La première section définit certaines notions fondamentales de la modélisation et simulation. Puis dans la deuxième section, nous nous appuyons sur ces définitions pour expliquer les défis posés par la simulation des Smart Grids.

2.1 Notions fondamentales de la modélisation et simulation

2.1.1 Modèle

La notion de ce qu'est un modèle est perçue différemment selon les domaines scientifiques ou les cas d'application. Le dictionnaire Larousse en ligne [1] donne une liste non-exhaustive de 15 définitions

1. www.larousse.fr/dictionnaires/francais/modèle

différentes. Il s'agit d'une notion abstraite, dont la forme n'est pas limitée. Pour un dessinateur par exemple, un modèle est l'objet ou la personne physique qu'il choisit de représenter. A l'inverse pour un architecte immobilier c'est le "dessin" – le plan – qui constitue un modèle, et non le bâtiment concret qui sera construit. Néanmoins, on retrouve cette idée du représentant et du représenté.

En ingénierie des systèmes, qui est la discipline dans laquelle s'inscrit la conception de Smart Grids, les modèles servent à représenter les systèmes étudiés. Plusieurs ont tenté d'en produire une définition générique [Bezivin and Gerbe, 2001, Seidewitz, 2003]. Nous considérons la définition suivante, donnée par [Jézéquel et al., 2012] à partir de ces précédents travaux :

Définition 1 (***Modèle***)**.** Un modèle est un ensemble de faits caractérisant un aspect d'un système dans un objectif donné. Un modèle représente un système selon un certain point de vue, à un niveau d'abstraction facilitant par exemple la conception et la validation de cet aspect particulier du système.

Nous nous accorderons notamment sur les propriétés suivantes d'un modèle [Kühne, 2006] :

— Propriété de cartographie (*mapping feature*) : Un modèle est construit à partir d'un original. Cet original constitue le système étudié.

— Propriété de réduction (*reduction feature*) : Un modèle ne représente qu'une sélection (pertinente) des propriétés de l'original. Il s'agit d'une abstraction simplifiée du système étudié.

— Propriété de pragmatisme (*pragmatic feature*) : Un modèle peut être utilisé à la place de l'original pour répondre à un certain but, à un certain nombre de questions sur le système.

2.1.2 Langage de modélisation

La notion de modèle fait implicitement référence à celle de langage de modélisation. Celui-ci définit les règles d'écriture et d'interprétation d'un modèle. C'est ce qui permet à l'information contenue dans ce modèle d'être partagée à et comprise par d'autres personnes que son auteur. Un langage peut être plus ou moins complexe, explicite et formel. Par exemple, la légende d'un schéma peut être considérée comme un langage de description visuel, mais est limitée et se permet sans doute des règles de lecture implicites. En revanche, le langage mathématique est utilisé pour décrire des objets de manière univoque et non-ambiguë.

Une manière simple de définir un langage consiste à identifier une **syntaxe** et une **sémantique** [Jézéquel et al., 2012, Seidewitz, 2003]. Nous posons ainsi les définitions suivantes :

Définition 2 (***Langage de modélisation***)**.** Un langage est l'ensemble constitué d'une syntaxe et d'une sémantique.

Définition 3 (***Syntaxe abstraite et concrète d'un langage de modélisation***)**.** La syntaxe est l'ensemble des règles définissant l'agencement entre eux des éléments manipulés par le langage. On définit plus précisément :

La syntaxe *abstraite* identifie les notions abstraites manipulables par le langage et la façon dont elles peuvent être liées ensemble dans un modèle.

La syntaxe *concrète* définit les éléments concrets du langage (les mots, les formes, etc.) qui représentent les notions de la syntaxe abstraite.

Définition 4 (***Sémantique d'un langage de modélisation***)**.** La sémantique d'un langage définit le sens des notions abstraites de ce langage, c'est-à-dire la façon dont elles doivent être interprétées (par un humain ou un ordinateur). Ce sens peut être défini par du langage naturel, ou dans un langage informatique, par une définition mathématique, etc.

Il est courant de dire que la syntaxe représente la *forme*, tandis que la sémantique représente le *fond* d'un modèle.

[Hardebolle and Boulanger, 2009] caractérise un langage comme *formalisme de modélisation* lorsqu'il possède une syntaxe et une sémantique formelles, c'est-à-dire précises et non-ambigües.

2.1.3 Simulation informatique

En ingénierie des systèmes, il n'est pas toujours pratique ni possible de réaliser des analyses et des évaluations directement à partir des systèmes réels, notamment dans les cas où ils sont encore en phase de conception et n'existent pas. Il est alors possible de construire des modèles, qui sont une représentation simplifiée du réel, et qui peuvent sous certaines conditions et hypothèses remplacer les systèmes dans les études. La simulation informatique est un moyen particulier d'exploitation d'un modèle, dit *dynamique*. Son objectif est de pouvoir prédire l'évolution dans le temps de l'état du système représenté.

Nous considérons les définitions suivantes, établies à partir de celles définies dans [Gomes et al., 2018c] :

Définition 5 (***Modèle dynamique***)**.** Modèle d'un système, caractérisé par un état et des règles d'évolution par rapport à une variable indépendante : généralement le temps.

Définition 6 (***Trace comportementale***)**.** Trajectoire suivie par l'état d'un modèle dynamique.

Un modèle dynamique peut être résolu à l'aide de méthodes analytiques pour obtenir une trace comportementale exacte, mais cela n'est pas toujours facile, rapide ou même réalisable. Dans ce cas, il est possible d'utiliser un solveur algorithmique qui calculera une trace comportementale, pouvant présenter des erreurs et approximations par rapport à la trace analytique. Toujours à partir de celles définies dans [Gomes et al., 2018c], nous posons ainsi les définitions suivantes :

Définition 7 (***Simulateur***)**.** Algorithme prenant un modèle dynamique en entrée et calculant la trace comportementale.

Définition 8 (***Simulation***)**.** L'obtention de la trace comportementale d'un modèle dynamique, calculée à partir d'un simulateur.

Ainsi, l'objectif de prédiction du comportement d'un système nécessite de réaliser un modèle dynamique et de le résoudre par la simulation si les méthodes analytiques échouent. Nous pouvons faire remarquer qu'un modèle dynamique destiné à être simulé par un ordinateur devrait être écrit dans un formalisme de modélisation. Le risque s'il y a des ambiguïtés sur l'interprétation d'un langage, c'est que l'intention du modeleur et la façon dont le modèle est effectivement simulé soient en incohérence.

2.1.4 Modèle de calcul

Un modèle dynamique représente la réalisation d'un certain nombre de comportements du système étudié. Différentes structures de données (matrices, fonctions, arbres, etc.), différentes notions de temps (continu, discontinu, réactif, etc.) et différentes façons d'enchaîner le comportement des éléments d'un modèle (séquentiel, asynchrone, concurrent, etc.) peuvent être utilisées. Ces trois composants sémantiques définissent ce qu'on appelle un **modèle de calcul** (**MoC** pour *Model of Computation*), qui fournit des règles communes d'interprétation de la structure et du comportement d'un modèle dynamique [Boulanger et al., 2011].

On peut citer en exemple trois MoC courants en simulation de systèmes [Lee and Sangiovanni-Vincentelli, 1996] :

Continuous time Dans un MoC "temps-continu", l'état de l'ensemble des composants du modèle dynamique évolue en fonction du temps, qui est lui-même continu. A chaque valeur du temps existe potentiellement un nouvel état du système. Il s'agit d'un MoC adapté à la simulation des comportements physiques, modélisés généralement par des équations différentielles. Un simulateur utilisera des méthodes de résolution numériques avec un certain pas d'intégration.

Discrete-Event En revanche, dans le MoC "événementiel discret", le temps est considéré comme une grandeur évoluant de manière discrète entre des instants particuliers, correspondant aux dates de différents évènements. L'état du système change à chacune de ces dates particulières, et n'est pas décrit entre ces dates. Il s'agit d'un MoC adapté à la simulation de comportements de télécommunications, dans lesquels chaque arrivée ou départ de paquet représente un évènement. Un simulateur adapté maintient généralement une pile d'évènements futurs et les exécute dans l'ordre chronologique.

Data flow Enfin, il est intéressant de présenter le MoC "flot de données", car celui-ci n'a tout simplement pas de notion du temps. L'exécution du comportement d'un tel modèle se fait de manière purement causale, dans lequel des éléments de calcul s'enchaînent dans un ordre particulier, selon des données d'entrée. C'est le but de la résolution de déterminer quels éléments de calcul ont été réalisés et dans quel ordre. Les diagrammes d'activité UML par exemple correspondent à ce genre de MoC.

Ces trois exemples sont intéressants car ils permettent d'illustrer que la façon dont est représenté et simulé le comportement d'un système peut grandement varier selon le MoC choisi. Néanmoins, certains comportements sont plus adaptés à un MoC qu'à un autre, et c'est ce qui influencera le choix du langage de modélisation.

En effet, c'est le rôle du langage ou du formalisme de modélisation de fixer les composants sémantiques. Un même MoC peut donc être utilisé par plusieurs langages de modélisation, mais rien n'empêche un même langage de pouvoir utiliser plusieurs MoC différents. Ainsi, simuler un modèle dynamique consiste à exécuter le langage utilisé par un algorithme qui implémente les règles d'interprétation du ou des modèles de calcul correspondants.

En définitive, le choix du MoC ou du langage de modélisation dépend de ce qui correspond le mieux au comportement à représenter. Un même système, composé de nombreux comportements, peut donc nécessiter plusieurs MoC et langages pour sa modélisation. Cela facilite l'effort conceptuel de modélisation – pas besoin de "tordre" un unique modèle de calcul pour l'adapter à tous les besoins – mais introduit une hétérogénéité dans le modèle global du comportement.

2.2 Défis posés par la simulation des Smart Grid

2.2.1 Un système complexe et cyber-physique

Les **systèmes complexes** sont des systèmes constitués de multiples composants inter-dépendants. Leur particularité est qu'il n'est pas possible de prévoir la dynamique d'un système complexe uniquement à partir de l'analyse indépendante des comportements de chacun de ses composants. En effet, les interactions et rétroactions entre les composants créent des comportements dits "émergents", qu'il n'est possible de prévoir qu'en les considérant tous à la fois.

Une autre caractéristique fréquente des systèmes complexes est due à la diversité des composants qui le composent. Leur étude nécessite alors la maîtrise de compétences provenant de plusieurs disciplines scientifiques. On identifie pour un système complexe les différents domaines techniques qui le concernent en identifiant les composants et les comportements liés à l'étude d'une même discipline scientifique. Chacun de ces domaines dispose d'une communauté d'experts, d'outils et de méthodes de résolution ayant fait leur preuve.

Dans tous les secteurs d'activité (aéronautique, transports, énergie, chaînes de production, ...) nous constatons l'émergence d'un type particulier de systèmes complexes, qui intègrent des aspects physiques, logiciels et de télécommunication [Lee, 2008, Hermann et al., 2016, Mosterman and Zander, 2016]. Dans ces systèmes, les processus physiques sont supervisés par un système informatique, généralement distribué et utilisant des réseaux de télécommunications pour s'échanger des informations. Les mesures effectuées sur les composants physiques via des capteurs ou autres équipements connectés influent sur le comportement des composants informatiques, qui agissent en retour sur le système par l'ajustement de consignes et ordres de pilotage. Ces systèmes sont appelés **systèmes cyber-physiques** (ou plus simplement **CPS**, pour *Cyber-Physical Systems*).

Les Smart Grids sont un exemple de système complexe et cyber-physique : l'objectif de création de réseaux électriques intelligents nécessite en effet la mise en relation simultanée de différents domaines

techniques formant un réseau électrique piloté grâce à un système informatique adapté. La mise en place de processus de transport et de traitement de l'information rend possible l'automatisation de la prise de décisions, permettant par exemple d'adapter au niveau local la production et la consommation d'énergie, tout en garantissant une plus grande fiabilité du réseau électrique.

Le Smart Grid fait donc intervenir au minimum :

— le domaine de l'électrotechnique, qui étudie les comportements électriques des composants du réseau électrique,

— le domaine des télécommunications, qui étudie les comportements de transmission de l'information via le réseau de télécommunications,

— le domaine du traitement de l'information, qui étudie les comportements et les processus de manipulation de l'information, et de contrôle des équipements du réseau électrique par les applications logicielles constituant le système d'information.

Enfin, l'une des particularités d'un Smart Grid est que son échelle de taille peut grandement varier, depuis l'étude des réseaux locaux à un bâtiment ou quartier (bâtiment intelligent, micro-grids) jusqu'à l'échelle nationale ou internationale (réseau électrique français, plaque européenne, etc.).

2.2.2 Intégration dans une démarche industrielle

L'étude d'un système industriel tel que le Smart Grid fait intervenir de nombreux collaborateurs – des experts techniques, des décideurs, des chefs de projet, des prestataires, etc. – qui se partagent la connaissance des enjeux et du fonctionnement de ce système. Chacun possède ses propres objectifs et responsabilités, et la réussite de leur étude dépend de leur capacité à s'entendre et travailler ensemble.

Comme évoqué précédemment, la simulation permet d'analyser le comportement d'un système particulier. La simulation industrielle s'inscrit généralement dans une démarche dont le but est d'obtenir le modèle d'une solution répondant à différents objectifs, selon certains critères : qualité de service, retour sur investissement, efficience, etc. Cette démarche peut être illustrée de manière simplifiée et générique [Quesnel, 2006, Camus, 2015, Paris, 2019] par la figure 2.1. Les étapes de la démarche sont répétées jusqu'à ce que la phase d'analyse caractérise le modèle comme étant valide.

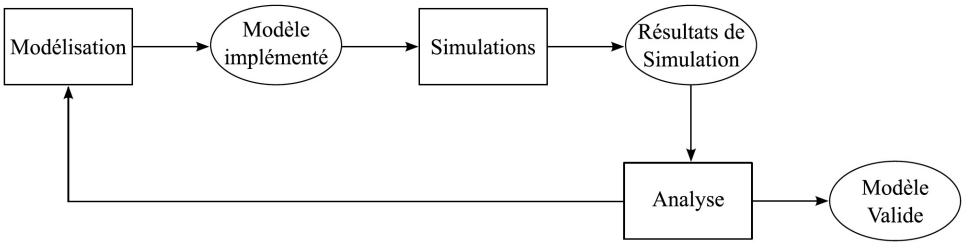

FIGURE 2.1 – Démarche de modélisation et simulation itérative jusqu'à obtention d'un modèle valide. Source : [Camus, 2015]

La figure 2.2 montre plus en détail un exemple de processus de modélisation d'un système [Galán

et al., 2009], depuis une vision abstraite du système cible jusqu'à la production d'un artefact exécutable simulant son comportement.

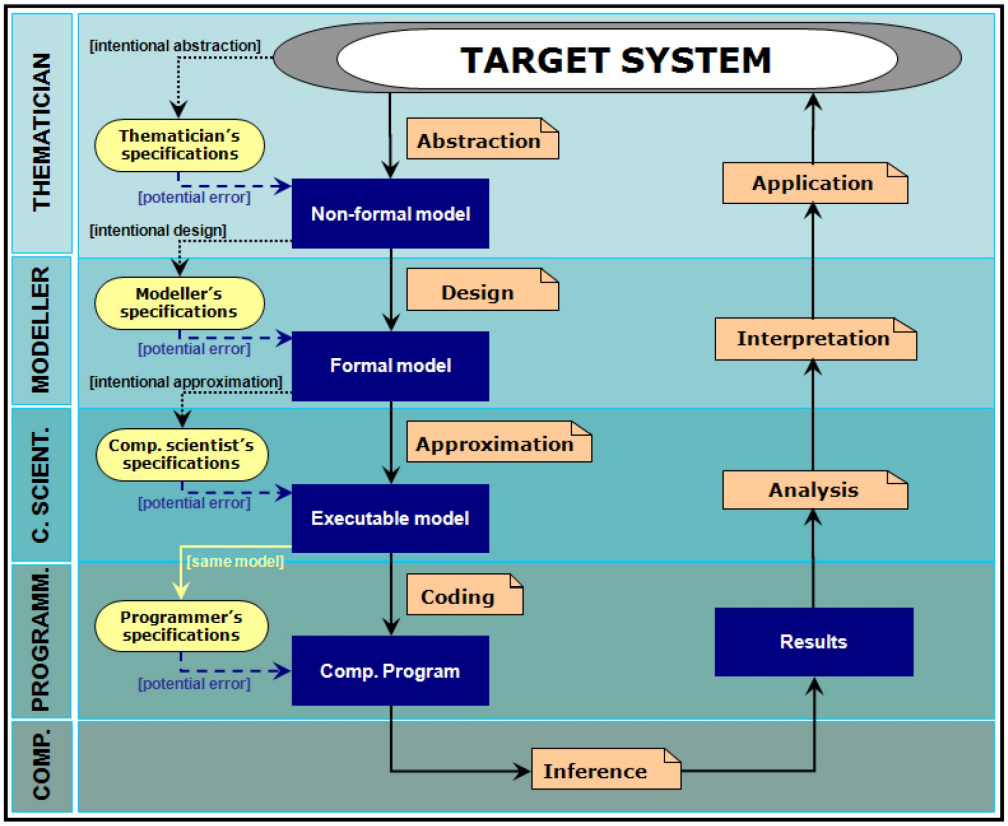

FIGURE 2.2 – Processus de raffinement successifs depuis la spécification d'un système cible vers un modèle exécutable de son comportement. Source : [Galán et al., 2009]

Cette figure a l'avantage de faire figurer de nombreux profils d'acteur, intervenant d'abord dans l'élaboration d'un modèle d'exécution final, puis dans l'analyse des résultats. Ces profils sont distingués selon le niveau d'abstraction de leur expertise, et chaque nouveau modèle détaillant le précédent peut faire intervenir des erreurs de cohérence dues par exemple à des différences d'interprétation.

A cette distinction "verticale", selon le niveau d'abstraction, s'ajoute en plus la distinction "horizontale" selon le domaine technique d'expertise, d'autant plus importante que le niveau de détail est élevé. En effet, comme nous l'avons vu précédemment, les personnes développant les modèles de simulation du comportement électrotechniques ne sont pas les mêmes personnes que celles développant les modèles de simulation du comportement de télécommunications et ceux du système d'information. Cela provoque potentiellement de nouvelles erreurs, entre les modèles de conception communs aux différents domaines et ceux dynamiques différenciés selon le domaine technique.

2.2.3 Modélisation hétérogène du comportement

D'une façon générale, on parle de **paradigme de modélisation** pour exprimer un cadre de pensée de modélisation, une façon de représenter les choses [Kuhn and Hacking, 2012]. C'est une notion abstraite, qui varie selon le domaine technique et les connaissances manipulées, le niveau de détail attendu, le type de système, etc. Plusieurs paradigmes peuvent donc se confronter dans la représentation d'un système, chacun plus adapté à certains aspects du système qu'à d'autres. C'est notamment le cas pour la conception de systèmes complexes et cyber-physiques, tels que les Smart Grids.

Nous avons présenté les Smart Grids comme des systèmes complexes, et plus particulièrement cyber-physiques. Cette caractérisation nous permet de poser deux propriétés liées à la modélisation de leur comportement :

Propriété 1 : Le comportement global du système ne peut être prédit qu'en prenant en compte l'ensemble des comportements des composants du système. Ceux-ci ne peuvent pas être traités individuellement.

Propriété 2 : Les modèles utilisés pour représenter un Smart Grid combinent des comportements physiques (souvent représentés par un MoC "temps continu"), ainsi que des comportements cybernétiques (pouvant être représentés par exemple par les MoCs "évènementiel discret" ou "flot de données").

Nous rappelons que le choix d'un modèle de calcul (MoC) pour représenter un comportement définit un ensemble de règles sur la façon dont un modèle dynamique sera écrit et interprété. Ainsi, dans le cadre de la modélisation comportementale, des MoC différents peuvent être considérés comme des paradigmes de modélisation différents. De plus, un MoC peut être utilisé par plusieurs langages ou formalismes de modélisation, de la même façon qu'un même langage peut manipuler plusieurs MoC, donc plusieurs paradigmes. La conséquence est que le système est alors décrit par une collection de modèles dynamiques **hétérogènes**, comme dans les cas indiqués dans le tableau 2.1.

	Paradigmes identiques	**Paradigmes différents**
Formalismes identiques	homogène	*hétérogène*
Formalismes différents	*hétérogène*	*hétérogène*

TABLE 2.1 – Les différents cas d'hétérogénéité. Source : [Hardebolle, 2008]

De ce rappel, ainsi que des propriétés 1 et 2 précédentes, nous déduisons que l'analyse du comportement global d'un Smart Grid requiert l'utilisation et l'analyse conjointe de plusieurs paradigmes de modélisation. La modélisation d'un Smart Grid correspond aux deux cas d'hétérogénéité de la colonne de droite du tableau 2.1. Elle entre donc dans le cadre d'application de la discipline de **modélisation hétérogène**, dont la définition est la suivante :

Définition 9 (***Modélisation hétérogène***)**.** La modélisation hétérogène, ou la modélisation multi-paradigmes, est "la discipline dont l'objectif est de faciliter et automatiser l'utilisation conjointe de modèles hétérogènes pendant le cycle de développement de manière à rendre possible un raisonnement global sur un ensemble de modèles hétérogènes" [Hardebolle, 2008] .

En conclusion, simuler un Smart Grid demande de mettre en place une plateforme de simulation qui devra calculer la trace comportementale commune à plusieurs modèles dont les paradigmes et le langages sont différents. De nombreux travaux traitent de la difficulté à résoudre les incompatibilités sémantiques (sens des concepts) et syntaxiques (forme du langage) engendrées [Vangheluwe et al., 2002,Henzinger and Sifakis, 2006,Hardebolle and Boulanger, 2009], incompatibilités que notre approche doit pouvoir résoudre.

2.3 Conclusion

Un Smart Grid est un système complexe dont la simulation nécessite le développement de plusieurs modèles dynamiques, par plusieurs personnes aux domaines, outils et compétences différentes. Ainsi, des experts du réseau électrique, du réseau de télécommunications, et du système d'information développent des modèles de simulation, qui devront être pris en compte ensemble dans l'étape de simulation. Ces modèles mélangent plusieurs paradigmes et formalismes de modélisation différents, certains manipulent par exemple des règles d'évolution continues, d'autres des règles d'évolution discrètes. Notre **problématique** 1 consiste à identifier une méthode et des outils de simulation pouvant calculer le comportement d'un Smart Grid à partir de modèles sémantiquement et syntaxiquement différents.

Par ailleurs, la simulation d'un Smart Grid s'intègre dans une démarche d'ingénierie des systèmes, dans laquelle le système à concevoir est représenté sous plusieurs niveaux d'abstraction. Il est ainsi nécessaire de s'assurer que l'ensemble formé par les modèles de simulation de chaque domaine technique forme un ensemble cohérent, implémentant les modèles de conception du Smart Grid. Notre **problématique** 2 de cohérence entre les modèles consiste donc à trouver le moyen de vérifier l'adéquation des modèles de simulation avec ceux de conception.

Notre contribution devra être évaluée notamment sur un ensemble de critères, identifiés en introduction (sous-section 1.3.1), permettant de juger sa capacité à s'intégrer à un contexte de simulation industriel :

— Notre approche devant favoriser l'environnement industriel et concurrentiel, elle doit s'intéresser en priorité aux méthodes optimisant le temps de mise en œuvre et limitant la courbe d'apprentissage de ses utilisateurs (**critère 1 de coût**) : parallélisation des tâches pour gagner du temps, réutilisation des outils déjà maîtrisés.

— Notre approche doit pouvoir s'intégrer dans une démarche itérative avec un minimum d'effort, c'est-à-dire que des ajustements sur la solution à modéliser doivent impacter au minimum les modèles déjà réalisés (il n'est pas forcément nécessaire de modifier le modèle du réseau électrique si les modifications ne concernent que les comportements de télécommunications) (**critère 2 d'itération**).

— Notre approche doit également, paradoxalement, concilier la collaboration, donc le partage, entre différentes entreprises partenaires (**critère 3 de collaboration**), et la conservation de la propriété intellectuelle des éléments échangés (**critère 4 de propriété intellectuelle**).

— Enfin, notre approche doit rester flexible afin de pouvoir s'adapter à des échelles différentes de système. Cela implique notamment de pouvoir faire varier la puissance de calcul disponible pour la simulation des modèles dynamiques (**critère 5 de scalabilité**).

3 Cohérence entre modèles

Sommaire

3.1 Ingénierie Dirigée par les Modèles, vers des modèles productifs 22
3.1.1 Principes 22
3.1.2 IDM et ingénierie logicielle 23
3.1.3 Métamodèles et *Domain Specific Language* 23
3.1.4 Transformations de modèles 26
3.2 Cohérence des modèles à différents niveaux d'abstraction 28
3.2.1 Architecture d'Entreprise 28
3.2.1.1 Définition 28
3.2.1.2 Modélisation par point de vue 28
3.2.2 *Model Driven Architecture* 30
3.2.3 Application aux CPS et aux Smart Grids 32
3.2.3.1 Le SGAM : un cadre d'architecture de Smart Grid 32
3.2.3.2 Approches impliquant l'IDM 32
3.3 Approches de simulation de modèles hétérogènes 35
3.3.1 Formalismes hybrides 36
3.3.2 Transformation de modèles 36
3.3.3 Composition de modèles 37
3.3.4 Cosimulation 37
3.3.5 Le cas du *Software* et *Hardware-in-the-loop* 38
3.3.6 Bilan 38
3.4 Conclusion du chapitre 40

La simulation d'un Smart Grid nécessite la représentation formelle de son comportement à l'aide de plusieurs modèles de simulation, hétérogènes. Ces modèles sont généralement dérivés d'autres modèles, réalisés dans le cadre de la conception du système à simuler, à des niveaux d'abstraction plus élevés. L'objectif de ce chapitre est de présenter différentes techniques permettant de :

1. garantir la cohérence les modèles de simulation avec les modèles de conception dont ils sont dérivés,

2. mettre en cohérence les modèles hétérogènes de simulation entre eux lors de leur exécution.

Nous commençons par introduire les concepts de l'Ingénierie Dirigée par les Modèles (IDM), discipline de conception de systèmes centrée sur le développement et l'exploitation de modèles productifs. Puis dans la seconde section nous présentons différentes approches de mise en cohérence de modèles à différents niveaux d'abstraction. Enfin, la troisième section liste et compare plusieurs techniques permettant l'exécution de modèles hétérogènes de simulation.

3.1 Ingénierie Dirigée par les Modèles, vers des modèles productifs

3.1.1 Principes

L'**Ingénierie Dirigée par les Modèles** (IDM) — *Model Driven Engineering* (MDE) en anglais — est une discipline de conception de systèmes industriels, centrée sur les modèles. La principale caractéristique et originalité de l'IDM par rapport à d'autres approches basées sur les modèles est d'utiliser des modèles *productifs*, c'est-à-dire exploitables informatiquement, tout au long du cycle de vie du système. Cela implique que :

— les modèles productifs possèdent une sémantique explicite et non-ambiguë, afin qu'il soit possible de définir des règles systématiques d'interprétation et de manipulation de l'information contenue dans le modèle ;

— les modèles productifs sont fournis sur un support numérique, souvent des fichiers textuels (contrairement à des fichiers binaires, comme des images, qui peuvent difficilement être parcourus).

Un modèle de simulation est un exemple de modèle productif, car il est "lu" par un programme informatique afin de produire automatiquement une trace comportementale. Cette dénomination s'oppose aux modèles dits "contemplatifs" qui n'ont qu'une portée de documentation – généralement des spécifications en langage naturel, des schémas à la syntaxe implicite, etc. – et qui sont écrits et interprétés uniquement par des humains.

L'IDM encourage notamment la création et la définition explicite de nouveaux langages spécifiques à un domaine, afin de pouvoir capitaliser les connaissances de ce domaine particulier. Ces langages sont appelés *Domain Specific (Modeling) Languages* (**DSL** ou **DSML**). Leur définition se fait notamment au travers de la pratique de **métamodélisation** (voir sous-section 3.1.3). Disposer d'une définition explicite des langages utilisés apporte plusieurs possibilités, qui sont à la base des approches IDM. Parmi ces possibilités, nous pouvons citer [Jézéquel et al., 2012] :

— la validation de modèles : définir des règles de validation au niveau d'un formalisme de modélisation permet d'évaluer automatiquement la validité d'un modèle,

— la transformation de modèles : définir des relations de transformation entre plusieurs formalismes de modélisation permet d'effectuer des traductions d'un modèle dans plusieurs langages,

— la composition de modèles : définir des relations complexes entre plusieurs formalismes de modélisation permet de faciliter la manipulation et l'analyse commune de plusieurs modèles hétérogènes.

3.1.2 IDM et ingénierie logicielle

Les modèles utilisés en IDM sont réalisés par des outils informatiques. En ingénierie logicielle, les modèles et le système partagent donc une même nature : celle d'être des artefacts numériques. Ainsi, le fossé entre modèle et système (une application informatique) n'est pas aussi important que dans le cas de l'ingénierie de systèmes matériels. Il est possible de maintenir des liens de traçabilité entre le modèle et son implémentation, et même d'effectuer des transformations de l'un à l'autre afin de conserver une cohérence entre les deux.

C'est pourquoi, aujourd'hui, l'IDM est utilisée de manière optimale dans la conception de systèmes informatiques – ou en partie informatiques comme les Smart Grids et les CPS en général. L'approche *Model Driven Architecture* (MDA) de l'OMG est un exemple de la maturité dont dispose l'ingénierie logicielle dans la mise en place de pratiques d'IDM (voir la sous-section 3.2.2). B. Combemale présente d'ailleurs l'IDM comme étant par définition "une forme d'ingénierie générative dans laquelle tout ou partie d'une application est engendrée à partir de modèles" [Combemale, 2008] .

3.1.3 Métamodèles et *Domain Specific Language*

Les possibilités de manipulation des modèles offertes par l'IDM nécessitent une définition explicite du langage utilisé, ce qui peut se faire au moyen notamment d'un **métamodèle**.

Définition 10 (***Métamodèle***)**.** "Un métamodèle est un modèle qui définit le langage d'expression d'un modèle, c'est-à-dire le langage de modélisation." [Jézéquel et al., 2012]

Un métamodèle *représente* un langage de la même manière qu'un modèle représente un objet du monde réel. En pratique, le métamodèle définit un ensemble de concepts qui pourront être instanciés dans un modèle, ainsi que des relations entre ces concepts [Bezivin and Gerbe, 2001], ce qui correspond à la définition de la syntaxe abstraite d'un langage (voir définition 2). On dit qu'un modèle est **conforme à un métamodèle** si tous ses éléments sont des instances d'éléments du métamodèle et respectent ses contraintes. Nous illustrons dans la figure 3.1 les liens entre objet réel, modèle, langage et métamodèle dans les approches d'IDM.

De même, on peut définir le langage d'un métamodèle par un **méta-métamodèle**. L'OMG organise ces différents niveaux de modélisation au sein de sa *pyramide de modélisation* (figure 3.2). Il définit un méta-métamodèle appelé *Meta Object Facility* (MOF) présentant la propriété de *métacircularité*[1] afin de limiter le nombre de niveaux d'abstraction. La figure 3.3 illustre avec un exemple simple cette hiérarchie de métamodélisation, à partir du méta-méta-"élément" `Class` défini dans le MOF.

Le fait de définir explicitement un langage permet d'appliquer automatiquement les raisonnements qui s'y rattachent à l'ensemble des modèles qui l'utilisent. On distingue deux classes de langages principales [Brambilla et al., 2017] :

1. capacité à se décrire lui-même

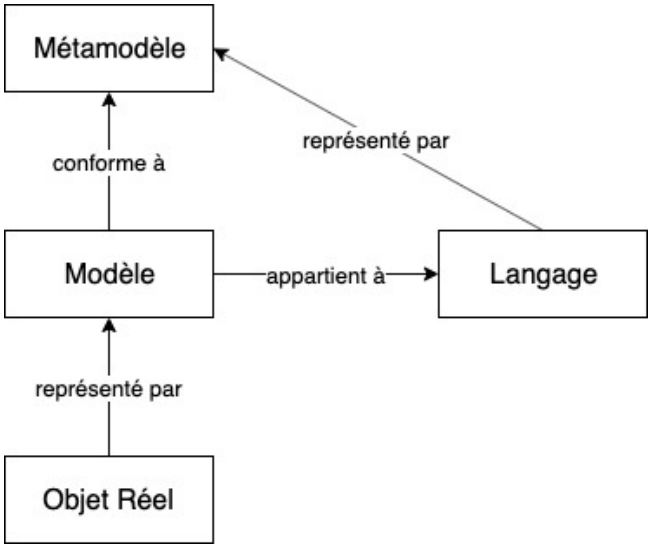

FIGURE 3.1 – Relations entre modèle, langage et métamodèle

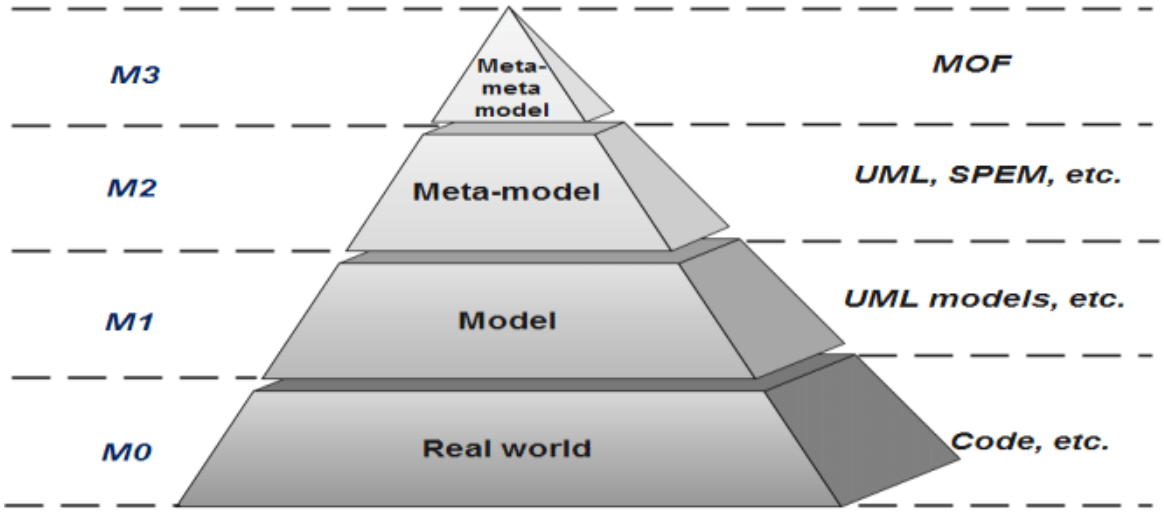

FIGURE 3.2 – Pyramide de modélisation de l'OMG. Source : [Erraissi and Belangour, 2018]

Langages(de modélisation) dédiés à un domaine (DSLs ou DSMLs). Ces langages sont conçus spécifiquement pour répondre à un besoin de modélisation dans un domaine ou contexte particulier. Ils simplifient le travail des utilisateurs qui ont besoin de représenter des concepts dans ce domaine. Parmi les exemples les plus connus se trouvent *HTML* ou *CSS* pour le développement de pages web, SQL pour l'accès aux bases de données, etc.

Langages généralistes (GPLs, GPMLs ou GMLs). Ces langages sont en revanche polyvalents et peuvent être utilisés dans n'importe quel contexte. Ils demandent en général un effort supplémentaire pour adapter les concepts généraux du langage aux concepts spécifiques du domaine, mais sont plus largement connus. Le langage UML, ou les langages de programmation comme Java ou C sont des exemples typiques de langages généralistes.

Il est reconnu que le développement de nouveaux DSL pour répondre à un besoin spécifique per-

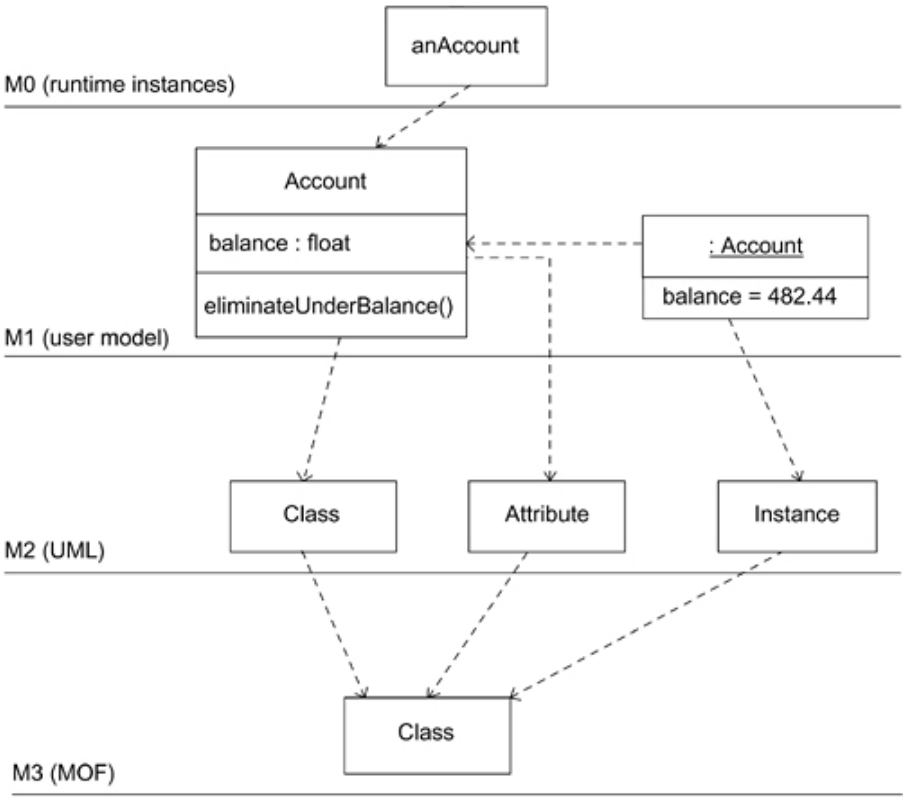

FIGURE 3.3 – Exemple des 4 niveaux hiérarchiques de métamodélisation. Source : [Mellor et al., 2004]

met aux entreprises de gagner en productivité et qualité, mais demande un investissement initial et un coût de maintenance supérieurs à l'utilisation de GPL ou de DSL existants [Kelly and Tolvanen, 2008]. En effet, il est nécessaire de créer le langage, les outils et les générateurs associés. Néanmoins des environnements existent aujourd'hui pour faciliter cette création, notamment autour de la plateforme Eclipse [2] et son environnement de modélisation associé (EMF/Ecore [3], Xtext [4], Sirius [5]) Des approches de développement de langages basées sur cet environnement sont disponibles, comme Xmodeling Studio [Le Goaer et al., 2018] ou Gemoc Studio [Combemale et al., 2017]. L'utilisation de ces plateformes d'ingénierie des DSL diminue l'investissement initial, qui est plus rapidement recouvert au fil des réutilisations, jusqu'à ce que le coût cumulé de l'approche basée sur le DSL développé devienne inférieur aux approches basées sur des GPL.

Notre problématique étant spécifique à la conception des Smart Grids, et s'intégrant à une démarche itérative basée sur la répétition, elle entre dans les critères qui rendent le développement d'une approche basée sur les DSL avantageuse.

2. www.eclipse.org
3. www.eclipse.org/modeling/emf/
4. www.eclipse.org/Xtext/
5. www.eclipse.org/sirius/

3.1.4 Transformations de modèles

La transformation de modèles est définie par certains comme "le coeur et l'âme" de l'IDM [Sendall and Kozaczynski, 2003]. Elle permet en effet d'automatiser de nombreuses activités au coeur de l'IDM, comme la "refactorisation" et le raffinement de modèles, l'extraction de vues, la génération de code, etc.

Nous choisissons d'utiliser les définitions suivantes, reprenant celles de [Kleppe et al., 2003], mais en introduisant la possibilité d'avoir plusieurs modèles en entrée et en sortie d'une transformation :

Définition 11 (***Transformation de modèles***)**.** La transformation de modèles est un processus, qui produit un ou plusieurs modèles cibles à partir d'un ou plusieurs modèles sources, selon des règles définies dans un modèle de transformation.

Définition 12 (***Modèle de transformation***)**.** Le modèle de transformation contient l'ensemble des règles définissant comment des modèles écrits dans les langages source peuvent être traduits en modèles dans les langages de destination.

Définition 13 (***Règle de transformation***)**.** Une règle de transformation décrit comment un ou plusieurs éléments de construction d'un ou plusieurs langages source peuvent être transformés en un ou plusieurs éléments de construction d'un ou plusieurs langages de destination.

Ainsi, effectuer une transformation de modèles revient à faire exécuter par un moteur de transformation un ensemble de règles de transformation entre plusieurs langages, sur des modèles conformes à ces langages. Ce genre d'opérations sur les modèles nécessite la formalisation précise des langages de modélisation, ce qui nous est fourni par l'activité de métamodélisation.

Nous illustrons dans la figure 3.4 le cas d'une transformation définie par un modèle de transformation `Mt`. Il s'agit d'une transformation dite *one-to-one*, c'est-à-dire depuis un unique modèle source `Ma` vers un unique modèle destination `Mb`. `MMt`, `MMa` et `MMb` sont leurs métamodèles respectifs, tandis que `MMM` est leur méta-métamodèle.

Dans le cas de transformations impliquant plus d'un modèle en entrée et en sortie – transformations dites *one-to-many*, *many-to-one*, ou *many-to-many* – le modèle de transformation fait simplement référence à plus de métamodèles.

Il existe un cas courant correspondant à un mode dégradé de la transformation de modèles telle que définie précédemment. Dans ce mode, le "modèle" de destination est une chaîne de caractères. Les règles de transformation explicitent ainsi comment des éléments d'un ou plusieurs métamodèles peuvent être convertis en du texte. Parmi les applications les plus courantes de ce type de transformations, on trouve par exemple la génération de code informatique (ex. génération de classes Java à partir d'un diagramme de classes UML), ou la sérialisation de modèle [6] (ex. enregistrement d'un diagramme de

6. opération d'encodage d'un objet informatique sous une forme séquentielle (comme du texte, ou une suite d'octets) contenant le minimum d'informations nécessaires à la reconstruction de cet objet en mémoire

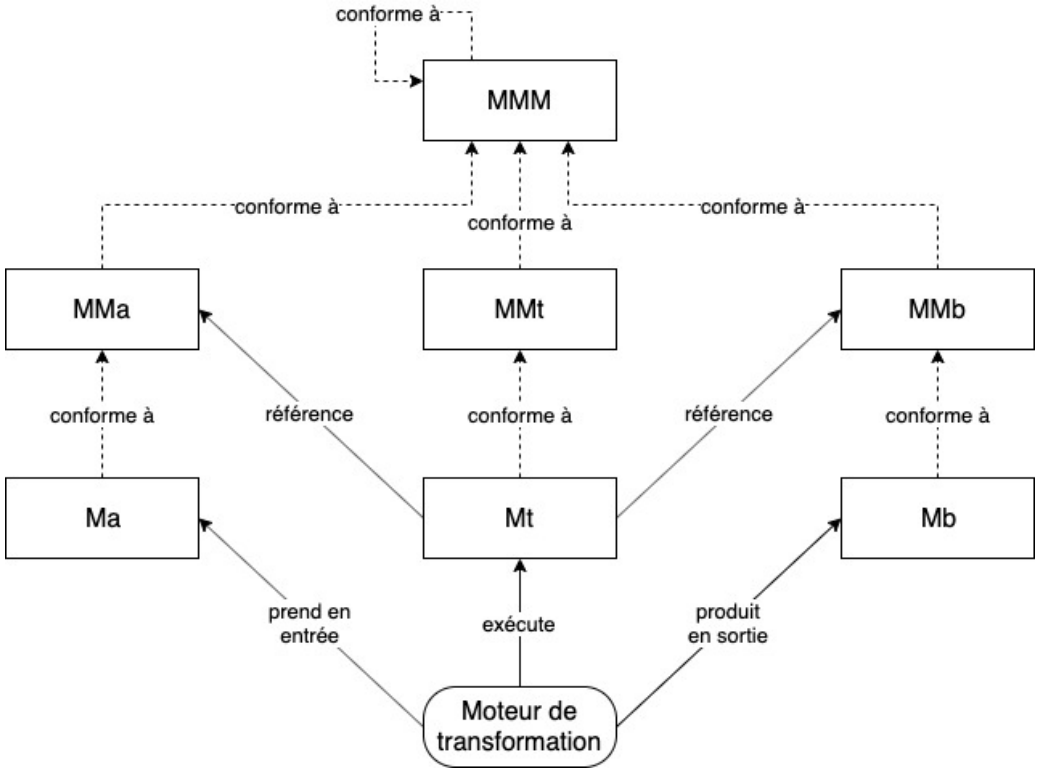

FIGURE 3.4 – Liens entre modèles et métamodèles dans le cas d'une transformation de modèles *one-to-one*

classes UML dans un fichier XML). Ce type de transformation est appelé *Model-to-Text* (**M2T**), et le cas inverse appelé *Text-to-Model* (**T2M**) peut être défini de manière analogue, et utilisé par exemple pour la visualisation de code ou la désérialisation de modèles. L'appellation *Model-to-Model* (**M2M**) est utilisée pour distinguer la transformation de modèles "classique" de son mode dégradé.

Cette distinction entre M2M et M2T reflète surtout une différence dans les outils et langages utilisés pour chaque cas. Ainsi, dans la mise en place de transformations M2M, nous utilisons des outils comme *Atlas Transformation Language* (ATL) ou le standard de l'OMG *Query View Transform* (QVT), alors que les transformations M2T utilisent des outils comme Acceleo ou le standard MOFM2T de l'OMG.

La transformation de modèles recouvre ainsi un vaste champ d'outils, de méthodes et d'utilisation. Mens & al. fournissent ainsi une classification des différents types de transformations de modèles, et des usages pouvant y être associés [Mens and Van Gorp, 2006]. Ils classifient notamment les transformations selon les critères suivants :

Verticales / horizontales : Les transformations **verticales** correspondent à un changement du niveau de détail d'un modèle. Cela correspond à une opération d'*abstraction* ou de *raffinement*. En revanche les transformations **horizontales** ne modifient pas le niveau de détail entre le modèle source et destination.

Endogènes / exogènes : Les transformations **endogènes** ne modifient pas le langage de modélisation d'un modèle, le métamodèle source est le même que celui de destination (par exemple utilisé

pour la refactorisation de modèles). Les transformations **exogènes** en revanche correspondent au cas de la figure 3.4, dans lequel le métamodèle de source et de destination sont différents.

3.2 Cohérence des modèles à différents niveaux d'abstraction

3.2.1 Architecture d'Entreprise

3.2.1.1 Définition

L'Architecture d'Entreprise (**EA** pour *Entreprise Architecture*) est une discipline apparue dans les années 80 lorsque l'informatique est arrivée au cœur des organisations. Scott A. Bernard dans son livre [Bernard, 2012] définit l'EA comme :

Définition 14 (***Architecture d'Entreprise***)**.** "L'analyse et la documentation d'une entreprise dans ses états présent et futurs, à partir de l'intégration d'une perspective stratégique, métier et technologique" (traduction depuis [Bernard, 2012, p. 31]).

Il s'agit ici de représenter l'ensemble des ressources d'une entreprise et la façon dont elles interagissent, afin d'aider au pilotage stratégique et au suivi de sa transformation. Bien que l'EA soit une discipline générale, son utilisation est en pratique centrée autour du système d'information (SI) d'une organisation, notamment en représentant le lien entre les applications et les objectifs métiers de l'entreprise. Le collectif Club Urba-EA [7] choisit ainsi de définir la discipline d'**urbanisme des systèmes d'information** comme la "déclinaison française" de l'*l'Entreprise Architecture* [URBA-EA, 2010, Préface].

Afin de permettre un alignement optimal entre la stratégie métier et le système d'information d'une entreprise, les personnes en charge de l'EA développent des modèles d'architecture qui mettent en cohérence des aspects métiers, fonctionnels, applicatifs et techniques. Comme dans notre problématique de modélisation des Smart Grids, la modélisation de l'architecture de l'entreprise fait appel à différents collaborateurs aux compétences et connaissances diverses, devant s'entendre et travailler ensemble.

3.2.1.2 Modélisation par point de vue

Dans l'EA, l'entreprise est vue comme un système, à concevoir, représenter, et faire évoluer. Plusieurs cadres (des *frameworks*) d'EA existent, dont les plus connus sont :

— le cadre de Zachman,

— *The Reference Model for Open Distributed Processing* (RM-ODP),

7. Club Urba-EA est une association inter-entreprises française – créée par AXA, FNAC, ORESYS, RATP, SUEZ Lyonnaise des Eaux en 2000 – afin de capitaliser les connaissances et les pratiques liées à l'architecture d'entreprise. www.urba-ea.org

— *The Open Group Architecture Framework* (TOGAF),

Chacun identifie et définit des concepts et méthodes pour mettre en œuvre une démarche d'EA. Néanmoins, une pratique prédominante dans la plupart des cadres d'EA est la décomposition de la modélisation de l'entreprise en plusieurs modèles selon différents points de vue. Chaque point de vue représente la perspective d'un acteur particulier de la modélisation du système. Cela permet de répartir la responsabilité entre les différents collaborateurs de la démarche de modélisation, certains acteurs ayant une vision plus proche de la stratégie et du métier de l'entreprise, tandis que d'autres ont une bonne connaissance de l'architecture technique.

Les points de vue définis varient selon les cadres, mais on retrouve dans chacun l'idée de "couches" superposées. Les points de vue des couches les plus hautes décrivent des notions conceptuelles et abstraites, proches de la stratégie et du métier de l'entreprise. Plus on "descend" dans l'ordre des couches, plus les points de vue décrivent des éléments concrets, techniques et détaillés [Steen et al., 2004, Seghiri et al., 2016, Rivière et al., 2013, p. 156].

Par exemple, la figure 3.5 illustre les différents points de vue du cadre défini par le Club URBA-EA, qui lui-même s'inspire des cadres d'EA précédents [URBA-EA, 2010] pour architecturer l'entreprise autour de son SI.

Chaque point de vue représente un "niveau de préoccupation" différent :

Le point de vue métier : ce point de vue représente la perspective métier de l'entreprise. Il décrit notamment les processus métiers implémentant la stratégie de l'entreprise.

Le point de vue fonctionnel : ce point de vue décrit la composition fonctionnelle de l'entreprise, c'est-à-dire le regroupement de services fournis par le SI en blocs modulaires, faiblement couplés et communiquant entre eux. Dans chaque bloc, ou *fonction*, existe une forte cohérence.

Le point de vue applicatif : ce point de vue décrit les applications déployées pour assister ou réaliser les services fonctionnels de l'entreprise.

Le point de vue technique ou matériel : enfin, ce point de vue décrit les ressources matérielles déployées pour implémenter les applications de l'entreprise.

La mise en place d'une démarche d'EA met donc nécessairement en relation de nombreux acteurs :
— des architectes, en charge de l'application de la démarche d'EA et de la modélisation,
— des acteurs métiers ayant une connaissance des enjeux et des besoins métiers de l'entreprise,
— des maîtres d'ouvrage ayant une connaissance précise des processus de l'entreprise,
— des experts apportant leurs connaissances sur les aspects techniques et de "bas-niveau",
— des maîtres d'œuvre pour analyser l'architecture modélisée et mettre en place des recommandations,
— etc.

Différentes perspectives sont réunies dans cette démarche, mais il est important également de conserver des liens de traçabilité entre les éléments modélisés dans chaque point de vue. Ce sont ces liens – souvent des liens de "raffinement" ou d'"implémentation", depuis les couches hautes vers les couches basses – qui permettent l'analyse et l'alignement entre le métier et les ressources (informatiques, ma-

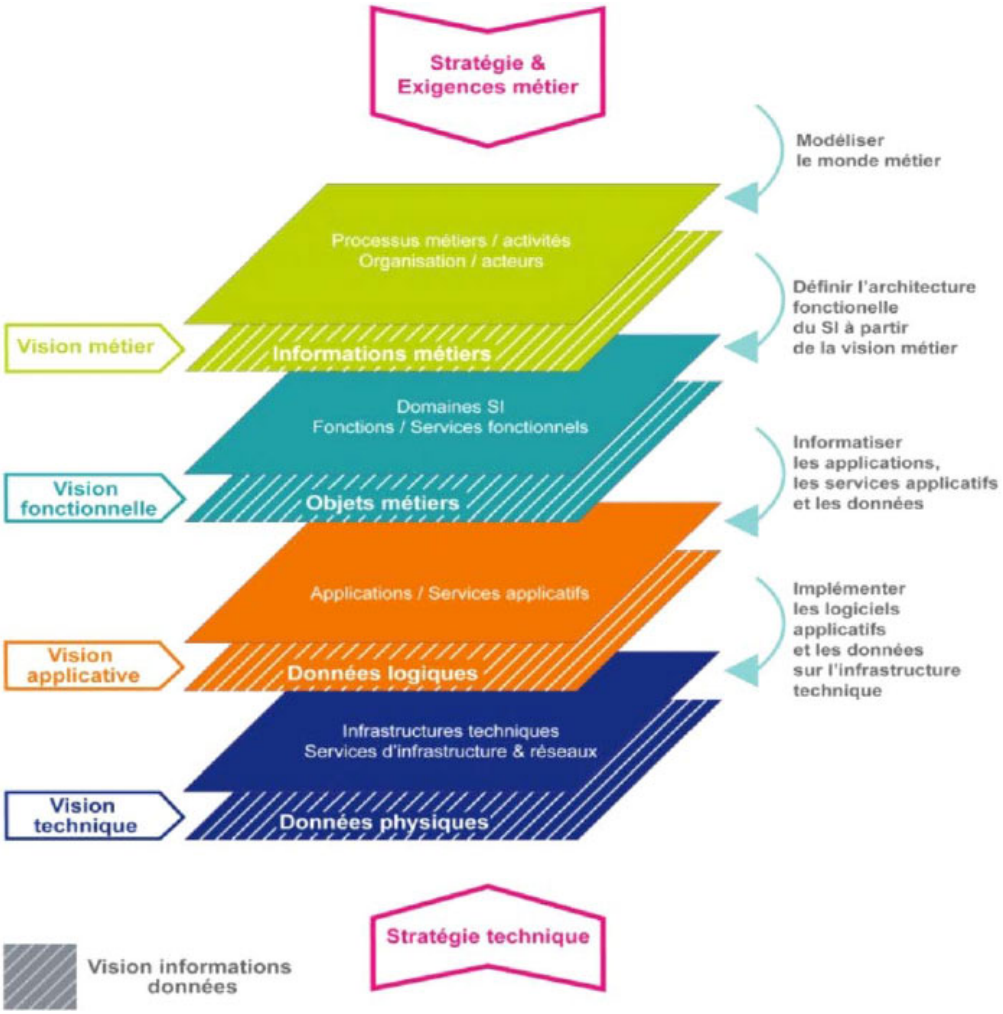

FIGURE 3.5 – Les 4 vues de l'urbanisation des SI, par le Club Urba-EA. Source [URBA-EA, 2010, p. 48]

térielles) de l'entreprise.

3.2.2 *Model Driven Architecture*

Model Driven Architecture (**MDA**) est une approche proposée par l'OMG, implémentant l'IDM à la conception de systèmes logiciels et se basant sur les standards de la modélisation de l'OMG. Parmi ces outils, on peut citer le langage généraliste UML, les outils de transformation de modèle QVT (M2M) et MOFM2T (M2T), etc. La définition d'un système pouvant faire intervenir de nombreux DSML, le métamétamodèle MOF est l'élément clé de toute approche MDA [Siegel, 2014].

Comme l'EA, MDA propose une approche en plusieurs "couches" de modélisation, selon le niveau de détail. Trois niveaux spécifiques de détails ont été définis :

***Computational Independant Model* (CIM) :** Le CIM correspond au modèle le plus abstrait. Son objectif est de modéliser le système selon son contexte, son utilisation et ses objectifs. Il

s'agit d'un modèle d'exigences, ne présentant aucune considération informatique.

***Platform Independant Model* (PIM) :** Le PIM correspond à la représentation logique du fonctionnement et de la structure du système. Il détaille les éléments du CIM correspondant à la partie logicielle du système, suffisemment détaillé pour permettre l'analyse et la conception, mais suffisamment abstrait pour pouvoir être appliqué à plusieurs plateformes différentes.

***Platform Specific Model* (PSM) :** Le PSM correspond au niveau de détail le plus élevé : il s'agit du modèle du code de l'application. Il redéfinit ainsi les éléments du PIM selon une plateforme spécifique, c'est à dire une technologie de compilation ou d'exécution.

L'objectif principal de MDA est de permettre la constitution de modèles PIM pérennes, indépendants des détails techniques propres à l'implémentation, afin de permettre la génération automatique du ou des modèles de code PSM et d'obtenir un gain de temps et d'effort important [Jézéquel et al., 2012]. Le processus classique de mise en place du MDA implique une composition de modèle avec un modèle de description de la plateforme de destination, appelé *Platform Description Model* (PDM), puis des transformations de modèles jusqu'à l'obtention du code (ou une partie du code) du système logiciel (figure 3.6).

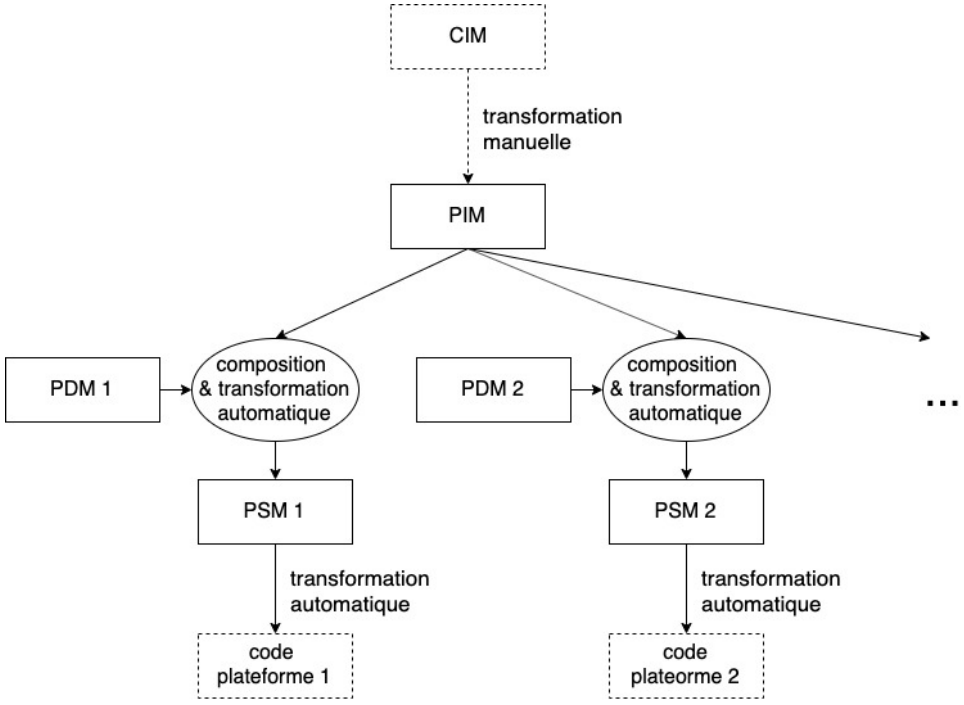

FIGURE 3.6 – Composition et transformations de modèles dans l'approche MDA

Le passage du CIM au PIM est en pratique peu automatisé. Le CIM restant à un niveau d'abstraction très élevé, il peut être constitué de modèles contemplatifs, formulé en langage naturel, ou autres langages non-formels. Les connaissances des acteurs responsables du développement d'un tel modèle n'ont en pratique pas la maîtrise de langages formels, indispensable à la constitution de modèles productifs et l'application de transformations de modèles.

En conclusion, MDA est une approche utilisant les transformations de modèles pour dériver automa-

tiquement les modèles de conception jusqu'aux modèles exécutables, qui sont les modèles dynamiques de simulation dans notre contexte. Outre le gain de temps et d'effort par rapport aux méthodes manuelles, cette automatisation limite également les erreurs d'inattention ou d'interprétation. Ces caractéristiques rendent cette approche particulièrement avantageuse dans les démarches itératives : les modifications sur les modèles de conception sont directement et facilement répercutées dans les modèles de simulation.

3.2.3 Application aux CPS et aux Smart Grids

3.2.3.1 Le *Smart Grid Architecture Model* : un cadre d'architecture de Smart Grid

Comme on l'a vu dans le chapitre 2, section 2.2, la modélisation et simulation d'un Smart Grid s'inscrit dans une démarche industrielle, composée de plusieurs acteurs avec des connaissances scientifiques différentes, mais également des niveaux d'expertises différents. L'EA a pour objectif de permettre un alignement entre ces expertises, et utilise pour cela une approche de modélisation par points de vue de différent niveau d'abstraction. Dans chacune de ces vues, chaque concept peut être lié à ceux des autres vues via des liens de cohérence.

La conception de Smart Grids rentre ainsi dans le cadre d'application de la discipline d'EA. Il existe d'ailleurs un cadre d'architecture spécifique aux Smart Grids, appelé *The Smart Grid Architecture Model* (SGAM) [Bruinenberg et al., 2012]. Le SGAM a été mis au point par un groupe d'experts internationals, le *CEN-CENELEC-ETSI Smart Grid Coordination Group*, dans le cadre d'un projet européen de standardisation appelé *Smart Grid Mandate M/490*.

Le SGAM préconise la modélisation d'une architecture de Smart Grids, en classant les différentes entités représentées selon trois dimensions : le domaine, la zone, et le niveau d'interopérabilité (figure 3.7).

3.2.3.2 Approches impliquant l'Ingénierie Dirigée par les Modèles

Le SGAM est une approche de modélisation spécifique au Smart Grid, mais manquant d'une formalisation explicite des langages et méthodes à utiliser pour l'appliquer sur un cas d'étude. Des travaux ont alors été menés pour proposer une approche outillée [Dänekas et al., 2014, Uslar et al., 2019], suivant les principes de MDA :

SGAM Toolbox. *SGAM Toolbox*[8] implémente un métamodèle et un DSML associé définissant de manière formelle les concepts du SGAM, et les reliant aux différents niveaux d'abstraction de MDA – CIM, PIM et PSM. Il en ajoute également un quatrième appelé *Platform Specific Implementation* (PSI) correspondant au code de l'implémentation des composants logiciels d'un Smart Grid (figure 3.8). Cette *toolbox* se présente comme un plugin du logiciel de modélisation

8. www.sgam-toolbox.org

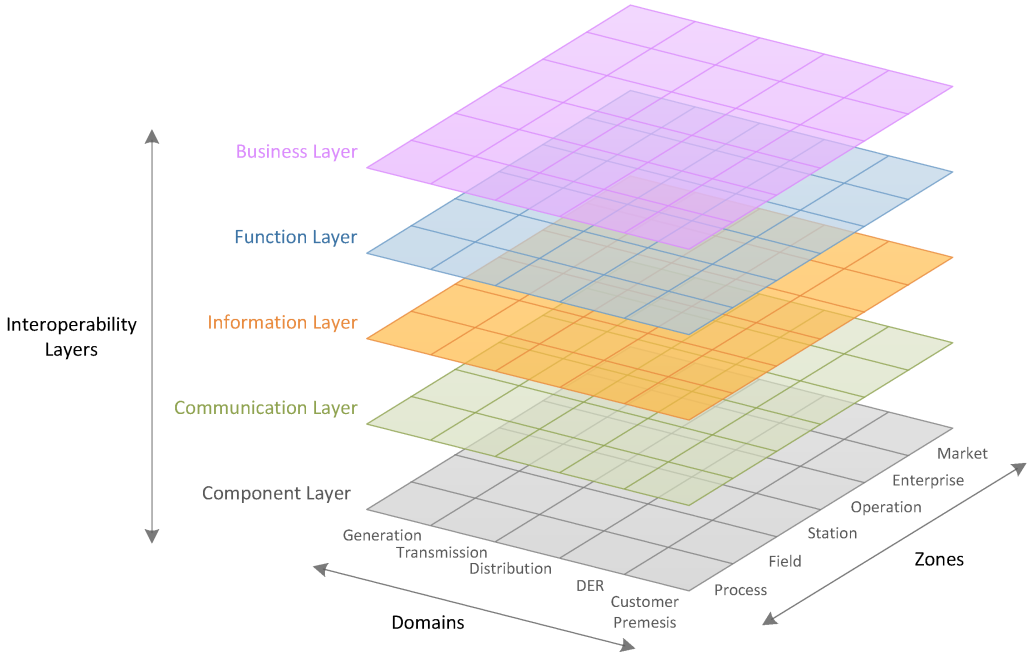

FIGURE 3.7 – *The Smart Grid Architecture Model*, et ses différentes perspectives

UML *Entreprise Architect*[9], et implémente certains mécanismes d'automatisation comme des transformations de modèles [Dänekas et al., 2014].

Power System Automation Language (PSAL). PSAL est un DSML formalisant les concepts du SGAM, associé à une méthodologie de développement rapide d'une fonction informatique de contrôle ou d'automatisation des réseaux électrique [Andrén et al., 2017]. PSAL permet une représentation structurelle des différentes fonctions et composants d'un système de contrôle de Smart Grid, détaillé sur quatre niveaux d'abstraction (figure 3.9). Dans la méthodologie proposée, ce modèle est considéré comme le PIM de l'approche MDA, et construit (manuellement) à partir de l'analyse des cas d'utilisation du système. L'objectif est ensuite d'utiliser un tel modèle pour générer le code d'une application (par exemple en IEC 61499) ou des configurations (en IEC 61850).

L'approche de PSAL nous semble intéressante, parce qu'il repose sur un DSML simplifié, il est ainsi rapide à apprendre, et facile à maintenir et faire évoluer. De plus, il manipule des concepts communs à tous les domaines techniques du Smart Grid – `Application`, `Connection`, `Device` – donc permet de mettre en cohérence à haut niveau les différentes expertises techniques, tout en fournissant un support de référence compréhensible par tous. Enfin, il conserve un niveau d'abstraction élevé, mais permet tout de même la génération d'un modèle d'application logicielle. En revanche, nous avons trouvé dommage le manque de lien entre le concept de `Connection`, central dans l'étude du comportement des télécommunications, et les autres concepts, notamment ceux décrivant les informations échangées entre fonctions du système.

L'une des principale pistes de travail et d'évolution de ces approches basées sur MDA, d'après Uslar

9. https ://sparxsystems.com/products/ea/ , produit de Sparx Systems

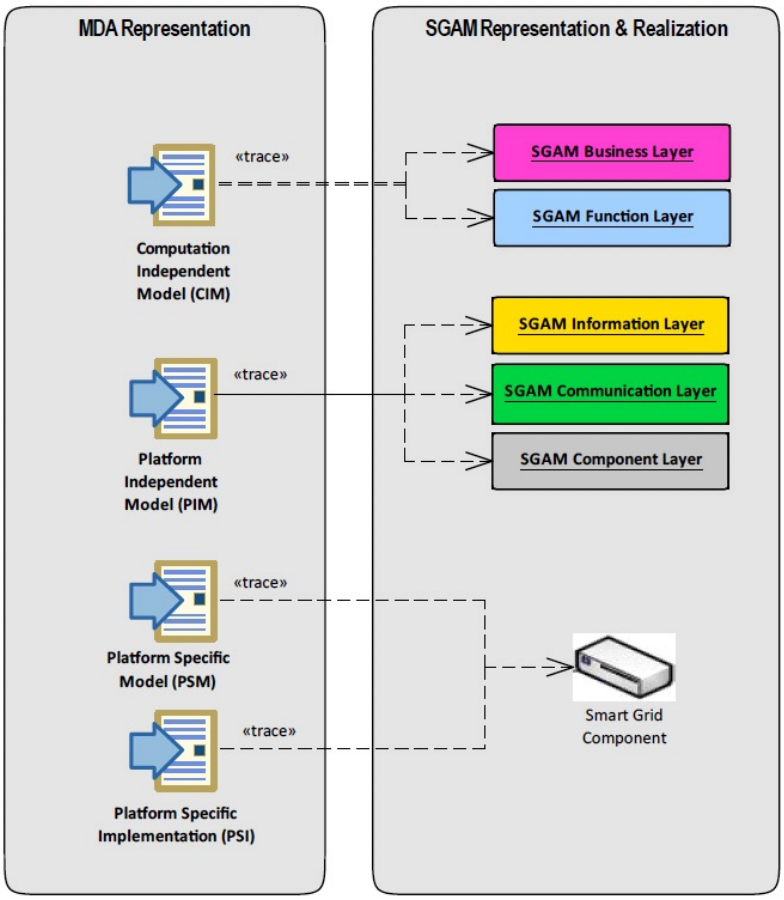

FIGURE 3.8 – Liens entre le *Smart Grid Architecture Model* et les niveaux d'abstraction de MDA dans l'outil *SGAM Toolbox*. Source : [Uslar et al., 2019]

& al. [Uslar et al., 2019], est d'offrir une meilleure intégration avec la démarche de simulation. En effet, les outils s'attachent à fournir des modèles permettant la conception, l'analyse et éventuellement la génération automatique de composants logiciels, mais ne permettent pas la modélisation dynamique du système.

Nous pouvons tout de même évoquer le *framework* MOSAIK [Schutte et al., 2011], qui propose un outil combinant la mise en cohérence de modèles à différents niveaux d'abstraction, et la combinaison de ces différents modèles dans une architecture de *cosimulation* (voir définition 19). MOSAIK implémente des DSL permettant de définir un modèle du scénario de simulation, de la topologie du Smart Grid et des liens entre les modèles de simulation. Puis il génère à partir de ces modèles des fichiers de configuration en XML d'un algorithme de simulation écrit en Python. Néanmoins, Mosaik est compatible avec certains outils uniquement, comme PowerFactory ou MatLab, et l'ajout d'un simulateur nécessite le développement d'un nouvel adapteur spécifique.

Les travaux de thèse de Thomas Paris [Paris, 2019] présentent une approche de simulation de CPS par cosimulation au sein de l'intergiciel MECSYCO. Lui aussi intègre plusieurs DSL permettant la

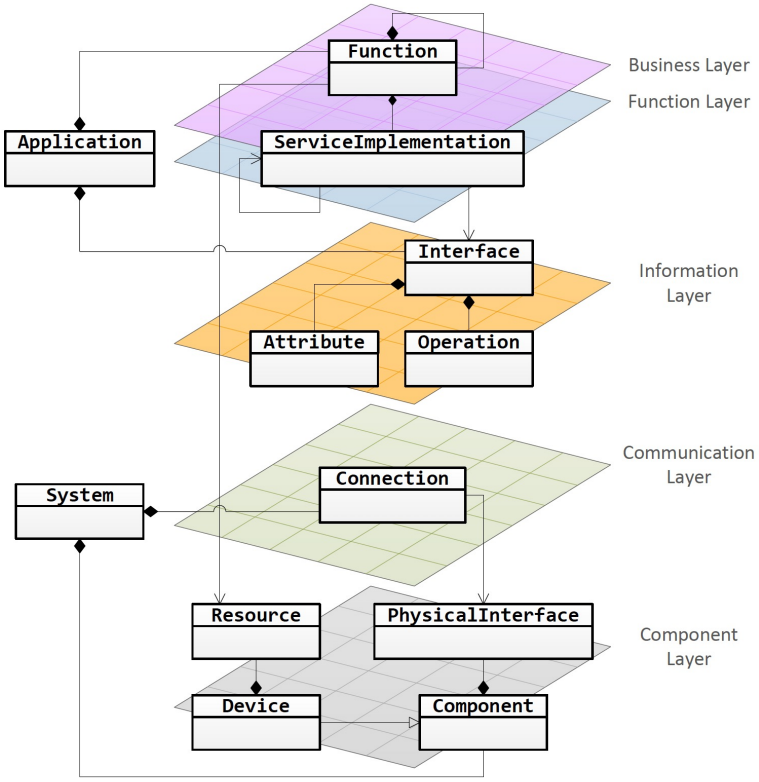

FIGURE 3.9 – Représentation simplifiée en UML des concepts du *Power System Automation Language* (PSAL). Source : [Andrén et al., 2017]

représentation des liens entre les modèles de simulation, afin de configurer l'outil de simulation.

En conclusion, les différentes approches de conception de système nous montrent l'importance des modèles représentant l'architecture du système à différents niveaux d'abstraction, mais ne se concentrent pas sur l'intégration de la simulation pour vérifier la validité de la solution conçue. Certaines approches de simulation de systèmes cyber-physiques et Smart Grid proposent bien l'utilisation de modèles abstraits pour cadrer l'exécution des modèles hétérogènes de simulation, mais représentent principalement l'architecture de la plateforme de simulation et non celle du système cible.

3.3 Approches de simulation de modèles hétérogènes

Dans la section précédente, nous avons présenté différentes pratiques et différents outils permettant la mise en cohérence des modèles de conception d'un système. Ces approches se focalisent sur une **cohérence verticale**, c'est-à-dire entre des modèles de différents niveaux d'abstraction. Néanmoins, la simulation des Smart Grids, bien que s'inscrivant dans une démarche de conception industrielle, présente également la problématique de **cohérence horizontale** au niveau des modèles dynamiques de simulation. Il est en effet nécessaire de pouvoir simuler ensemble des modèles hétérogènes, représen-

tant les comportements des différents domaines du Smart Grid. Dans cette section nous traitons des différentes façons de prendre en compte des modèles dynamiques hétérogènes dans des approches de simulation.

3.3.1 Formalismes hybrides

Il existe des outils permettant de modéliser et simuler des comportements hétérogènes au sein d'un même langage. Nous ne parlerons pas des langages de programmation généraux, comme Python, ou C, qui permettent de simuler n'importe quel type de comportement, mais sont trop complexes à utiliser pour des experts de domaine non-informatique et vont à l'encontre des principes de la modélisation. D'autres outils, comme le logiciel MatLab, ou le langage Modelica, sont qualifiés d'"hybrides", c'est-à-dire qu'ils proposent un formalisme de modélisation qui permet la représentation et la simulation de comportements continus (sous forme d'équations mathématiques) et discrets (par exemple sous forme d'une séquence d'opérations exécutées à la suite).

L'avantage de cette approche est la simplicité d'intégration des différents modèles ou comportements hétérogènes : ils peuvent tous être directement rassemblés au sein d'un unique modèle, simulé par un unique simulateur. En revanche, la mise à l'échelle du système, combinée à l'intrication de tous les comportements différents au sein d'un même modèle rendent difficiles les opérations de maintenance et d'évolution des modèles. De plus, les performances des outils de simulation hybrides sont rarement équivalentes dans tous les domaines représentés, aux langages et simulateurs spécifiques de ces mêmes domaines. Ainsi, Modelica est très performant sur la modélisation et simulation des comportements continus, mais ne propose qu'une faible expressivité pour représenter les comportements manipulant des évènements et des informations complexes.

En pratique, cette approche est ainsi utilisée pour des systèmes de petite taille ou des preuves de concept [Elsheikh et al., 2013], et sont réalisés par un nombre restreint de collaborateurs.

3.3.2 Transformation de modèles

La traduction de modèles est une approche basée sur l'utilisation de plusieurs formalismes pour l'étape de modélisation. Les modèles hétérogènes sont ensuite traduits dans un même formalisme par transformation de modèles, compatible avec l'unique outil de modélisation choisi. ATOM 3 est un outil de simulation proposant une approche de *multi-modélisation*, dans laquelle chaque formalisme est représenté par un graphe, et la traduction d'un formalisme dans un autre utilise des techniques de réécriture de graphe [de Lara and Vangheluwe, 2002].

L'un des avantages de cette méthode est que l'ajout d'un nouveau formalisme à prendre en compte ne modifie pas l'existant, car il suffit uniquement de définir la transformation du métamodèle du nouveau langage, vers le métamodèle du langage de simulation. Néanmoins, dans le cas d'une différence

importante du paradigme de modélisation, ou dans le cas de métamodèles particulièrement larges, la définition de la transformation peut se révéler complexe et demander un effort important, au risque d'introduire des imprécisions ou des pertes d'information entre les modèles d'origine et ceux générés. Enfin, la transformation de modèles ne s'intéresse pas à la réunion du modèle produit avec les autres modèles transformés, qui doit se faire manuellement.

3.3.3 Composition de modèles

Les approches de composition de modèles assemblent des modèles hétérogènes décrits par des formalismes différents. Elles consistent en l'adaptation des Modèles de Calcul (MOC) (voir sous-section 2.1.4) des modèles suivant des règles d'interaction.

Ptolemy II par exemple est un environnement de modélisation et simulation de modèles hétérogènes, dans lequel un grand nombre de MoC sont définis. Le système est modélisé par un ensemble d'acteurs atomiques ou composites en interaction, et des composants particuliers appelés *Directeurs* déterminent selon le MoC la dynamique d'exécution des acteurs et les adaptations à réaliser sur les communications entre eux. La combinaison de plusieurs MoC est réalisée par la définition d'une sémantique abstraite pour chacun, et des adaptations sémantiques nécessaires entre les MoC.

ModHel'X est un autre exemple d'environnement de modélisation et simulation de modèles hétérogènes par composition de modèle. Dans cet outil, les modèles sont organisés en "blocs" dont la dynamique interne n'est pas visible. Seul les informations au niveau de leurs ports d'interface sont prises en compte, et des "blocs d'interface" permettent au modeleur de définir comment est réalisée l'adaptation entre les MoC des modèles hétérogènes.

L'avantage de ces approches est qu'elles apportent une intégration formelle entre plusieurs modèles hétérogènes, tout en préservant la modularité des modèles. De nombreux MoC sont compatibles avec les outils présentés Ptolemy II et ModHel'X, néanmoins ce ne sont pas des solutions d'intégration génériques : il est compliqué d'intégrer des modèles développés par d'autres environnements

3.3.4 Cosimulation

Une pratique de plus en plus rencontrée en simulation de systèmes industriels est la **cosimulation** [Gomes et al., 2018b]. Dans cette approche, le modèle dynamique global du système étudié est décomposé en plusieurs sous-modèles hétérogènes, chacun simulé par un outil adapté. La cosimulation consiste à synchroniser et coupler par interface différents simulateurs pour permettre des échanges de données entre eux au cours de leurs simulations.

Cette pratique favorise la séparation des responsabilités entre les différentes équipes techniques, chargées de développer leurs modèles dans leur propre domaine d'expertise. La courbe d'adaptation de ces équipes est donc diminuée, car celles-ci sont encouragées à utiliser leur propres techniques et

outils de modélisation et de simulation. De plus, les outils de référence dans chaque domaine possèdent souvent de plus grandes performances et une meilleure précision que les outils de modélisation et simulation polyvalents.

La contrepartie est qu'il faut composer une simulation répartie en plusieurs logiciels de simulation, souvent conçus pour être utilisés individuellement. Des incompatibilités d'interface, de plateforme d'exécution (e.g. linux ou windows), ou de vitesse d'exécution par exemple peuvent compliquer la mise en place de l'architecture de cosimulation. Des outils comme *Functional Mockup Interface* (FMI) ou *High Level Architecture* (HLA) proposent des normes standards d'interconnexion de logiciels de simulation, afin de simplifier cette mise en place.

3.3.5 Le cas du *Software* et *Hardware-in-the-loop*

Dans une cosimulation, les différents modèles simulés sont censées se substituer au système réel dans l'analyse du comportement de celui-ci. Le couplage par interfaces permet d'interconnecter des modèles hétérogènes, aux technologies et aux niveaux de détail potentiellement différents. En poussant ce raisonnement, il peut être envisagé de constituer une cosimulation en interconnectant directement des composants du système réel avec des modèles de simulation. Ce type particulier de cosimulation est appelé *Hardware-in-the-loop* (HIL) ou *Software-in-the-loop* (SIL), selon la nature (matérielle ou logicielle) du composant réel intégré.

Comme pour une cosimulation "classique", il est toujours nécessaire de développer une interface et un protocole d'échange de données entre les différents composants de simulation. Les travaux de [Yang et al., 2013] par exemple utilisent des sockets de communication TCP et UDP pour interconnecter un modèle de centrale électrique dans l'outil MATLAB avec un équipement de contrôle du réseau, utilisant le protocole d'échange de données IEC 61499.

Ces possibilités rendent la cosimulation utilisable et réutilisable à différentes étapes du processus de conception d'un système, depuis les premières phases d'analyse avec des modèles de haut niveau, jusqu'aux tests des composants physiques ou logiciels développés. En revanche, HIL et SIL imposent généralement la contrainte d'un déroulement en temps réel, pouvant poser des difficultés techniques de synchronisation, et une restriction des paramètres et conditions d'étude du système.

3.3.6 Bilan

Nous listons dans cette section plusieurs outils et techniques permettant la simulation de systèmes hétérogènes. Le choix de la solution à retenir doit s'effectuer en fonction des critères de choix établis dans l'introduction (chapitre 1) et détaillés dans le chapitre 2. On évalue pour chacun de ces critères la pertinence de chaque solution, que l'on résume dans le tableau 3.1 ($\surd$ signifie que le critère est validé, $\approx$ signifie que le critère est partiellement validé, et $\cdot$ signifie que le critère n'est pas validé) :

1. *la capacité à optimiser les coûts et l'effort d'utilisation.* L'utilisation d'un formalisme hybride est simple et rapide à mettre en place, un unique langage est à apprendre et un seul outil de simulation est à utiliser. La transformation de modèles, la composition de modèles et la cosimulation permettent de choisir les langages à utiliser en fonction des préférences et aptitudes des modeleurs, ce qui simplifie leur tâche, mais demandent un effort initial supplémentaire pour configurer les outils de simulation en fonction des langages à utiliser. Le HIL et SIL sont en revanche généralement coûteux car demandent d'avoir accès à et de mobiliser du matériel ou du logiciel déjà développé, et de mettre en place une interconnexion spécifique.
2. *la capacité à s'intégrer dans une démarche itérative.* Le choix d'un formalisme hybride implique un nombre restreint de modèles, dans lesquels tous les comportements sont représentés, et potentiellement intriqués les uns au autres. Chaque modification demande une relecture de l'ensemble du modèle pour vérifier les parties à réécrire. En revanche, la transformation de modèles, la composition de modèles, la cosimulation et le HIL / SIL répartissent la représentation des différents comportements en plusieurs sous-modèles ou composants, et permettent d'implémenter des changements sans toucher aux modèles non-impactés.
3. *la capacité à faire collaborer plusieurs personnes/équipes aux compétences techniques différentes.* De même que précédemment, l'utilisation d'un formalisme hybride diminue la séparation entre les comportements de différents paradigmes, et donc complique le travail de collaboration. En revanche la transformation de modèles, la composition de modèles, la cosimulation et le HIL / SIL permettent de décomposer la représentation des comportements du système selon le domaine de compétence des collaborateurs.
4. *la capacité à garantir la propriété intellectuelle lors de la collaboration avec des prestataires externes.* L'utilisation d'un langage hybride, la transformation et la composition de modèles imposent que les modèles partagés soient visibles afin de permettre leur exécution commune. La cosimulation et le HIL / SIL permettent une interconnexion par interfaces uniquement, et donc la protection de la propriété industrielle des modèles ou composants développés entre les collaborateurs de la simulation.
5. *la capacité à s'adapter à un changement d'échelle.* L'utilisation d'un formalisme hybride, la transformation de modèles et la composition de modèles ont pour finalité d'être simulés dans un unique outil, exécuté en un seul processus, et qui charge "en mémoire" l'intégralité des modèles. Cela rend le passage à l'échelle plus difficile car demande la mise à disposition d'un calculateur d'autant plus important que le système est large et détaillé. La cosimulation en revanche repose sur l'exécution parallèle de plusieurs modèles, interconnectés via des interfaces et des échanges de données, ce qui permet le déploiement et la parallélisation des calculs entre plusieurs machines connéctées entre elles. Le passage à l'échelle avec HIL et SIL en revanche demande potentiellement la mise à disposition de beaucoup de materiel.

D'après le tableau 3.1, la cosimulation semble la méthode la plus adaptée pour la simulation des Smart Grids, ce que semble confirmer plusieurs travaux comparant les différentes alternatives [Palensky et al., 2017, Gomes et al., 2018c, Guermazi, 2017]. Néanmoins, il reste certaines difficultés à surmonter, principalement liées à la mise en place de l'architecture de simulation, qui interface et pilote l'exécution

Méthode	**Critères**				
	1 (coût)	2 (itération)	3 (collaboration)	4 (PI)	5 (scalabilité)
Formalisme hybride	$\surd$	$\approx$	·	·	·
Transformation de modèles	$\approx$	$\surd$	$\surd$	·	$\approx$
Composition de modèles	$\approx$	$\surd$	$\surd$	·	$\approx$
Cosimulation	$\approx$	$\surd$	$\surd$	$\surd$	$\surd$
HIL / SIL	·	$\surd$	$\approx$	$\surd$	·

TABLE 3.1 – Comparaison des alternatives de simulation de modèles hétérogènes en fonction des critères à valider pour la simulation industrielle de Smart Grids

de plusieurs outils logiciels différents.

3.4 Conclusion du chapitre

L'objectif de ce chapitre était de définir les techniques pouvant mettre en cohérence les différents modèles intervenant dans la modélisation et la simulation de Smart Grids, afin de pouvoir répondre à la **problématique 2** de nos travaux.

Nous constatons que la simulation de Smart Grids nécessite des modèles dédiés à la réflexion et la conception, décrivant le système et son architecture à un certain niveau d'abstraction. Lorsqu'il s'agit de modèles formels, ces modèles peuvent être automatiquement dérivés par des transformations de modèles en des modèles exécutables détaillés, afin de réduire le risque d'erreurs de cohérence entre ces différents modèles. Cette automatisation est particulièrement bénéfique dans le cas de démarches itératives, chaque modification dans les modèles de conception étant rapidement et facilement répercutée dans les modèles dérivés.

Néanmoins, en comparant les approches existantes nous remarquons que :
- les approches utilisant des modèles d'architecture du système les dérivent en modèles de code, représentant le code des parties logicielles du système (MDA, PSAL, SGAM Toolbox),
- les approches ayant pour objectif la validation par la simulation utilisent des modèles de l'architecture de la simulation, qui permet de mettre en cohérence les modèles dynamiques de la simulation.

Ainsi d'un côté les approches manipulant des modèles de conception d'un système n'intègrent pas la modélisation dynamique du système, mais celles centrées sur la simulation ne décrivent pas l'alignement des modèles de simulation avec l'architecture du système. Nous souhaitons dans nos travaux "reconnecter" les deux activités de modélisation et de simulation, et ainsi faire le lien entre les modèles de conception d'un Smart Grid et les modèles de simulation de son comportement.

Nous considérons la cosimulation comme la technique de simulation de modèles hétérogènes la plus adaptée à la simulation des Smart Grids, adressant la **problématique 1**. Néanmoins, la mise en place d'une plateforme de cosimulation nécessite l'interconnexion et la synchronisation des diiférents logiciels

choisis par les modeleurs, en fonction de leurs compétences et des nécessités du cas d'étude.

4

FMI et la cosimulation pour les Smart Grids

Sommaire

4.1 Vocabulaire et définitions de la cosimulation 44

4.2 Technologies et standards de cosimulation 45

4.2.1 Limites sur les interfaces propriétaires 45

4.2.2 *High Level Architecture* 46

4.2.3 *Functional Mockup Interface* 47

4.2.4 Comparaison et limites 49

4.3 Outils compatibles avec FMI pour cosimuler le Smart Grid 50

4.3.1 Simuler le réseau électrique avec FMI 51

4.3.2 Simuler le système d'information avec FMI 51

4.3.2.1 Modélisation et simulation 52

4.3.2.2 Utilisation avec FMI 53

4.3.3 Simuler le réseau de télécommunications avec FMI 53

4.4 Conclusion du chapitre 55

Nous avons défini la cosimulation comme l'approche la plus intéressante pour prendre en compte les différents modèles dynamiques représentant le comportement du Smart Grid, et les exécuter ensemble. Néanmoins les logiciels de simulation sont généralement conçus pour être utilisés individuellement, et ne disposent pas nécessairement d'interfaces compatibles avec les autres logiciels utilisés dans une plateforme de cosimulation. Dans la première section, nous commençons par définir plus en détail la cosimulation et les concepts associés, afin de mieux comprendre son fonctionnement. La seconde section positionne le standard de cosimulation *Functional Mockup Interface*, ainsi que ses avantages et limites par rapport aux autres technologies permettant de rassembler plusieurs simulateurs pour réaliser une cosimulation. Enfin, la dernière section liste et compare les différents outils de modélisation et simulation pouvant être utilisés au sein d'une plateforme de cosimulation FMI pour les Smart Grids.

4.1 Vocabulaire et définitions de la cosimulation

Les travaux de [Gomes et al., 2018c] fournissent un formalisme et un vocabulaire particuliers pour décrire la cosimulation, que nous choisissons de réutiliser. Les définitions suivantes en sont dérivées.

Définition 15 (***Unité de simulation***)**.** L'unité de simulation est l'ensemble constitué d'un modèle dynamique et de son simulateur. Il s'agit d'un composant exécutable prêt-à-l'emploi, calculant la trace d'un modèle à partir de données d'entrée.

Les unités de simulation sont donc des unités de calcul, pouvant disposer d'une interface représentant les données nécessaires et celles produites par leur exécution. Plusieurs unités de simulation peuvent alors être couplées entre elles via des **liens de couplage** entre leurs entrées et sorties.

Définition 16 (***Lien de couplage***)**.** Un lien de couplage relie une sortie d'une unité de simulation à l'entrée d'une autre.

L'ensemble des contraintes de couplage entre plusieurs unités de simulation constitue le **scénario de cosimulation**.

Définition 17 (***Scénario de cosimulation***)**.** Le scénario de cosimulation est l'ensemble des informations nécessaires à l'exécution d'une cosimulation, notamment la date de début et de fin de la simulation, ainsi que les différentes unités de simulation impliquées et les liens de couplage entre elles.

Enfin, un algorithme de pilotage des différentes unités de simulation est nécessaire afin de synchroniser leur exécution et réaliser les échanges de données en fonction du scénario de cosimulation.

Définition 18 (***Orchestrateur***)**.** Algorithme prenant en entrée un scénario de cosimulation, et pilotant le déroulement de l'exécution de plusieurs unités de simulation en synchronisant l'avancement du temps, et réalisant les échanges de données entre leurs entrées et sorties.

Enfin, nous définissons la cosimulation à partir des définitions précédentes :

Définition 19 (***Cosimulation***)**.** La cosimulation est l'exécution d'un scénario de cosimulation par un orchestrateur, permettant de fournir la trace comportementale combinée des modèles dynamiques de chaque unité de simulation.

De manière analogue à l'unité de simulation, nous pouvons définir le concept d'**unité de cosimulation**.

Définition 20 (***Unité de cosimulation***)**.** L'unité de cosimulation est l'ensemble constitué d'un scénario de cosimulation et d'un orchestrateur. Il s'agit d'un composant exécutable prêt-à-l'emploi, pouvant également nécessiter certaines données d'entrée afin de produire la trace comportementale du modèle couplé.

Le schéma de la figure 4.1 illustre ces concepts.

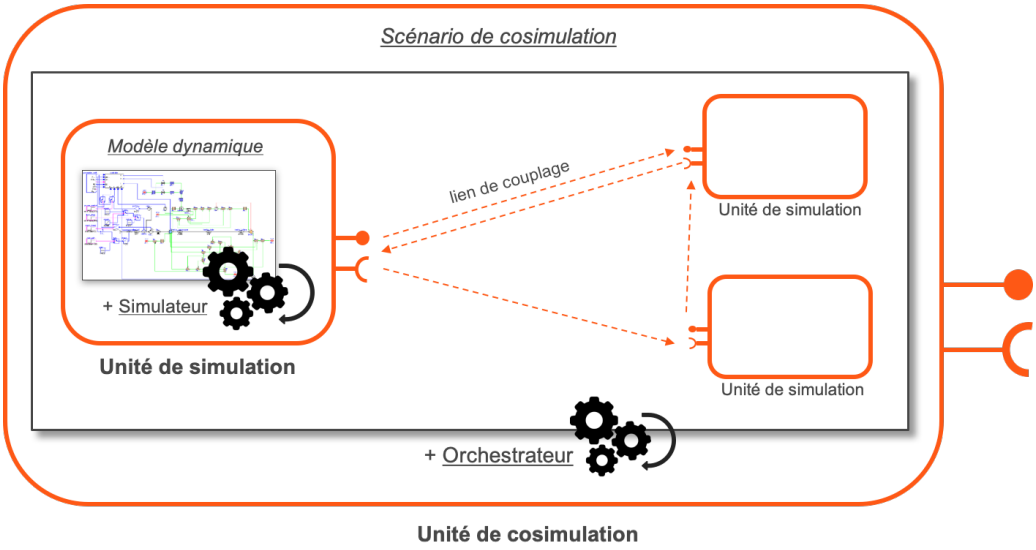

FIGURE 4.1 – Constitution d'une unité de cosimulation

4.2 Technologies et standards de cosimulation

4.2.1 Limites sur les interfaces propriétaires

La mise en place d'une *unité de cosimulation* nécessite d'utiliser des *simulateurs* proposant une interface permettant d'interagir avec leur processus d'exécution. Il est nécessaire en effet d'échanger des données entre ces simulateurs, et de synchroniser leur exécution.

VPNET [Li et al., 2011] est un environnement de cosimulation construit entre le logiciel VTB pour les comportements dynamiques physiques et OPNET pour ceux du réseau de télécommunications. L'échange de données est réalisé via un orchestrateur, qui utilise les sockets windows pour dialoguer avec OPNET, et une interface spécifique de programmation (COM) pour dialoguer avec VTB. Dans l'état de l'art de simulation des Smart Grids, de nombreux autres exemples utilisent ce genre d'interfaces spécifiques [Nutaro, 2011, Mets et al., 2011, Hopkinson et al., 2006, Zhang et al., 2014, Garau et al., 2017].

Le problème est que les interfaces proposées par les éditeurs de logiciel sont des interfaces propriétaires, propres à leur outil. Utiliser ces interfaces directement pour interconnecter ces logiciels entre eux impose de développer des "traducteurs" pour chaque combinaison d'interactions possible. Le nombre de ces "traducteurs" augmente donc rapidement avec le nombre de simulateurs employés – 1 "traducteur" pour 2 outils, 3 "traducteurs" pour 3 outils, 6 pour 4, etc. Le nombre de traducteurs t croît en fonction du nombre de simulateurs s selon la formule $t = \binom{s}{2} = \frac{s!}{(s-2)!\,2!}$.

C'est pourquoi ce genre de travaux se limitent en général à deux ou trois simulateurs au maximum, et n'offrent pas ou peu de réutilisabilité [Gomes et al., 2018c, Shum et al., 2018]. Utiliser un formalisme générique d'interopérabilité entre les logiciels et/ou les modèles, permettrait de couvrir le plus de cas d'étude possibles, et de permettre la réutilisation des modèles entre plusieurs simulations.

4.2.2 *High Level Architecture*

Le *High Level Architecture* (HLA) est une spécification d'architecture logicielle permettant une interopérabilité entre plusieurs composants de simulation [Dahmann and Morse, 1998]. Initié dans la fin des années 90 par le Département de la Défense des États-Unis, il a été déclaré un standard officiel de cosimulation en 2000 par l'IEEE [IEEE, 2001].

Une architecture mise en place avec HLA s'appelle une **fédération**. Les **fédérés** sont des processus qui doivent être mis en interaction, principalement des simulations informatiques, ils peuvent aussi être des interfaces de contrôle en temps réel par un opérateur humain, des processus de visualisation, etc. Ces fédérés implémentent l'**interface d'exécution de HLA** (*runtime interface*) afin d'interagir entre eux, via les services proposés par le **RTI** (*RunTime Infrastructure*). Le RTI peut ainsi être vu comme le point central d'une fédération, permettant la communication, la synchronisation du temps, l'échange d'évènements ou de données entre les fédérés. La figure 4.2 illustre les différents composants d'une fédération, constituée du RTI et des plusieurs processus fédérés en interaction autour (*Data Collector, Simulations, Interface to Live Players*).

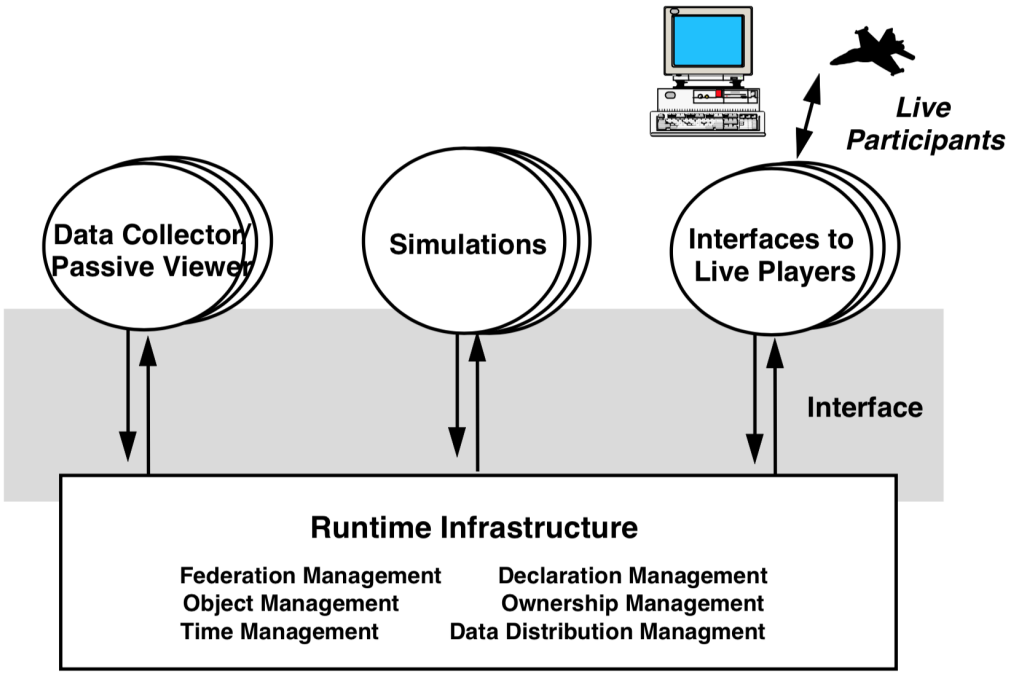

FIGURE 4.2 – Architecture logicielle avec HLA. Source : [Dahmann and Morse, 1998]

Pour réaliser cette interopérabilité, le RTI propose notamment les services suivants :

Declaration Management : Service permettant la gestion des souscriptions et publications. Chaque fédéré peut "publier" (fournir) des données qui seront utilisées par les fédérés y ayant "souscrit".

Object Management : Service permettant la mise à jour et la propagation des données aux fédérés y ayant souscrit.

Ownership Management : Service permettant le changement de propriétaire d'une donnée d'un

fédéré à un autre, afin d'éviter les conflits de mise-à-jour.

Time Management : Service permettant la synchronisation de l'avancée du temps entre les différents fédérés.

Data Distribution Management : Service permettant le routage efficace de la donnée entre fédérés.

De plus, HLA fournit un modèle de documentation d'objets pouvant être partagés (*Object Model Template*), utilisé pour définir les données partagées au sein d'une fédération (Federation Object Model), ou les fédérés pouvant être réutilisés dans d'autres fédération (Simulation Object Model).

HLA est donc un standard de cosimulation permettant l'interopérabilité entre plusieurs logiciels et environnements de simulation. Une fédération et ses fédérés peuvent être vus comme une *unité de cosimulation* intégrant plusieurs *unités de simulation*. Le RTI fonctionne comme l'*orchestrateur* de cette cosimulation, implémentant l'interface d'exécution de HLA pour permettre la réalisation des *liens de couplage* entre les *fédérés*, vus comme les unités de simulation.

4.2.3 *Functional Mockup Interface*

Functional Mockup Interface (FMI) [1] est un standard indépendant d'un outil en particulier, caractérisant une façon de coupler et simuler des modèles dynamiques hétérogènes. Il est développé et maintenu par un groupe d'entreprises et d'organisations, d'abord au sein du projet européen MODELISAR – terminé en 2011 – et aujourd'hui repris en tant que projet de la *Modelica Association* [2]. Comme HLA, il profite de la présence forte de partenaires industriels, tels que Dassault Systèmes, BOSCH, Siemens PLM et autres.

FMI définit deux contextes d'utilisation distincts [Blochwitz et al., 2011] :

***FMI for Model Exchange* :** Les modèles hétérogènes sont fournis sous forme d'un code C, et décrit par les équations algébriques, différentielles et discrètes de sa dynamique. Ils ont pour but d'être utilisés par un environnement de simulation qui calculera la trace comportementale à partir de ces équations.

***FMI for Cosimulation* :** L'objectif est le couplage de plusieurs simulateurs différents dans un environnement de cosimulation.

Dans les deux cas cités ci-dessus, un composant implémentant la norme FMI est appelé *Functional Mockup Unit* (FMU), et plus spécifiquement FMU-ME (FMU for Model Exchange) ou FMU-CS (FMU for Cosimulation) selon le contexte d'utilisation. Une FMU-CS peut donc être vue comme un composant constitué d'un modèle dynamique et de son simulateur associé, tandis qu'une FMU-ME ne contient que le modèle (figure 4.3), et le simulateur est externe et peut être le même entre plusieurs FMU-ME. L'une des autres différences importantes entre ces deux types de FMU est que le format FMU-ME ne peut être partagée qu'en "*white box*" – toutes les règles d'évolution d'état sont visibles – tandis que le

1. https ://fmi-standard.org
2. https ://www.modelica.org

format FMU-CS peut être partagé en tant que "*black box*", communiquant uniquement la structure de son interface [Durling et al., 2017].

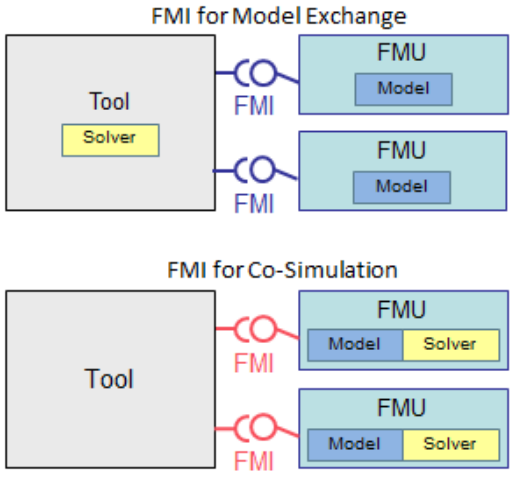

FIGURE 4.3 – Différence entre *FMI for Model Exchange* et *FMI for Cosimulation*. Source : Modelica newsletter 2014-3

Ce chapitre traitant des solutions de cosimulation, le terme de FMI fera implicitement référence à la partie du standard dédié à la cosimulation, *FMI for Cosimulation*, et le terme de FMU à celui de FMU-CS.

Une FMU peut être représentée comme un composant de simulation constitué de **ports** d'entrée et de sortie statiques, qui permettent de fournir ou récupérer l'état de certaines variables de simulation du modèle dynamique intégré. Chaque port est identifié par un type de donnée, correspondant au type de la variable qu'il représente. FMI est compatible avec les types primitifs `Real` (un nombre réel), `Integer` (un nombre entier), `String` (une chaîne de caractère), et `Boolean` (une variable booléenne). Cette expressivité limitée rend son intégration plus simple avec une majorité de simulateurs, qui disposent généralement d'équivalents internes à ces types de données.

Plus concrètement, une FMU est une archive ZIP avec l'extension `.fmu`, contenant :

— Un fichier XML documentant l'utilisation de la FMU, appelé `ModelDescription.xml`. Il contient différents paramètres d'utilisation, et présente les variables exposées en lecture ou écriture du modèle dynamique, qui permettent de déduire les ports d'entrée et de sortie de la FMU. Il est possible de définir notamment un ensemble de propriétés indiquant la compatibilité du composant avec des fonctions avancées et optionnelles du standard, comme le *rollback* permettant de reculer la simulation pour la reprendre à partir d'une date précédente.

— Un ensemble de fonctions C, fournies via leur code source ou une forme compilée (binaire) pour différentes plateformes, implémentant les fonctions définies dans le standard FMI, dont le tableau 4.1 présente les plus importantes. Ces fonctions sont utilisées pour dérouler la simulation du modèle dynamique, et pour accéder en lecture ou écriture aux variables exposées du modèle.

— Un ensemble de ressources pouvant être utilisées par les fonctions de la FMU ou par tout autre outil les utilisant (fichiers de documentation, DLLs et autres bibliothèques partagées, fichier

Nom de la fonction	Description
`fmi2SetXXX`	Fonctions utilisées pour définir la valeur des variables identifiées comme des entrées de la FMU dans le fichier `ModelDescription.xml` associé. `XXX` doit être remplacé par le type de la variable (Real, Boolean, String, Integer).
`fmi2GetXXX`	Fonctions utilisées pour obtenir la valeur des variables identifiées comme des sorties de la FMU dans le fichier `ModelDescription.xml` associé. `XXX` doit être remplacé par le type de la variable (Real, Boolean, String, Integer).
`fmi2DoStep`	Fonction utilisée pour demander à la FMU d'avancer son état interne d'un certain pas de temps.
`fmi2GetFMUState` (optionelle)	Fonction optionelle utilisée pour demander à la FMU de mémoriser son état actuel. Intervient dans l'opération de *rollback*.
`fmi2SetFMUState` (optionnelle)	Fonction optionelle utilisée pour demander à la FMU de revenir dans un état précédemment mémorisé. Intervient dans l'opération de *rollback*.

TABLE 4.1 – Description des principales fonctions C de FMI, implémentées par chaque FMU

d'icône, etc.).

Dans une cosimulation FMI, les échanges entre simulateurs sont restreints à certaines dates, appelées **points de communication**, qui forment un ensemble discret. Entre deux points de communication, les simulateurs évoluent de manière indépendante, et l'ensemble est piloté et synchronisé par un algorithme appelé ***master* FMI de la cosimulation**, qui implémente le **scénario de cosimulation** (voir définition 17). Dans le cadre de FMI, le scénario de cosimulation définit une contrainte de couplage entre deux FMU par un lien d'égalité entre la valeur d'un port de sortie de la première FMU, et celle d'un port d'entrée de la seconde FMU.

4.2.4 Comparaison et limites

HLA et FMI ont tous les deux été définis comme des standards d'interopérabilité logicielle. Le premier explicite une architecture à mettre en place et définit une interface pour interopérer permettant à plusieurs processus de simulation de communiquer au travers d'un bus central : le RTI. FMI en revanche fait le choix de définir une interface et un format d'échange, le FMU, permettant d'importer plusieurs simulations comme des bibliothèques logicielles au sein d'un processus "maître".

En pratique, l'un des principaux critères de choix entre ces deux solutions dépend de la nature de la donnée à échanger entre les simulateurs. Dans une cosimulation FMI, le *master* de cosimulation pilote tous les processus de simulation, et opère les échanges de données aux points de communication qu'il définit indépendamment. Ce comportement échantillonne les signaux échangés, ce qui le rend plus adapté aux signaux continus qu'aux signaux discrets. En revanche, HLA caractérise les données échangées ainsi que les simulateurs selon le paradigme objet. Chaque modification de valeur donne potentiellement lieu à un évènement (publication/souscription), ce qui est adapté à l'échange de signaux

discrets mais beaucoup moins à l'échange de signaux continus.

Si HLA est utilisée pour des cosimulations dites discrètes, et FMI pour des cosimulations continues, ces deux standards présentent des limitations dans le cas de cosimulations hybrides, c'est-à-dire combinant discret et continu [Gomes et al., 2018c]. Or ce dernier type de cosimulation est nécessaire dans le cas de CPS à simuler, tels que les Smart Grids.

Des travaux combinant FMI et HLA [Awais et al., 2013, Garro and Falcone, 2015] ou FMI avec des interfaces spécifiques "Ad-Hoc" [Vaubourg et al., 2015] ont été menés. Ces approches demandent néanmoins des développements spécifiques à chaque cas d'étude, limitant la modularité au sein de la cosimulation, et la réutilisation des modèles développés. D'autres travaux proposent ou utilisent des extensions de la norme FMI pour permettre l'échange de signaux discrets. On peut citer la possibilité d'effectuer des pas de communication de durée nulle [Guermazi, 2017, Cremona et al., 2016], la possibilité d'échanger des signaux sans valeurs [Cremona et al., 2016].

DACCOSIM [3] est un outil proposant une implémentation d'un *master* FMI de cosimulation. Cet outil implémente notamment des fonctionnalités avancées de FMI comme des pas de temps variables intelligents (qui s'adaptent en fonction de la dynamique d'évolution des signaux échangés), des fonctionnalités de détection et d'approximation de la date d'une discontinuité d'un signal, grâce à la fonctionnalité de *rollback*. Tavella [Tavella et al., 2016] propose également dans DACCOSIM une extension de la norme FMI qui permettrait de déterminer avec exactitude la date d'un "évènement" afin de la faire coïncider avec un point de communication.

4.3 Outils compatibles avec FMI pour cosimuler le Smart Grid

Une pratique courante pour simuler les CPS est de considérer la couche du domaine physique, celle du domaine des télécommunications et celle du logiciel [Zhang et al., 2014, Palensky et al., 2017]. Pour un smart grid, le domaine physique sera principalement celui du réseau électrique et de ses comportements électrotechniques, et le logiciel sera l'ensemble des comportements du système d'information (SI).L'utilisation d'outils déjà existants (et maintenus), et compatibles avec la norme FMI permet d'économiser l'effort manuel d'implémentation de l'interface autour du modèle et du simulateur, et l'obligation de maintenir et mettre à jour cette implémentation en fonction des évolutions du standard. On rappelle néanmoins que le contexte industriel et la contrainte de minimisation de l'effort d'adaptation des équipes de modélisation impose de faire certains compromis pour choisir des outils pertinents avec les domaines métiers impliqués. Nous essayons de retenir au maximum des outils standards et conventionnels pouvant convenir à différentes cultures d'entreprise.

3. https ://bitbucket.org/simulage/daccosim

4.3.1 Simuler le réseau électrique avec FMI

Le comportement électrique est généralement modélisé par des équations représentant les lois physiques d'évolution des grandeurs du domaine. FMI étant à l'origine développé pour des systèmes physiques, de nombreux outils existent et proposent une implémentation de la norme. Par exemple, MATLAB SimuLink de MathWorks ou le langage Modélica sont matures, flexibles – via notamment de nombreuses bibliothèques de composants – et couramment utilisés par les experts de la simulation électrotechnique. Ils bénéficient d'une bonne compatibilité avec le standard FMI en permettant l'import et l'export de FMU :

— pour les modèles Simulink, via le plugin FMI Toolbox [4] ;
— pour les modèles Modelica, via les éditeurs de modélisation et simulation Dymola [5] ou OpenModelica [6].

D'autres outils comme DIgISILENT PowerFactory ne proposent que des fonctions d'import (ce qui est souvent le cas car plus faciles à mettre en œuvre que les fonctions d'export), mais les modèles peuvent parfois être convertis dans un format ou langage compatible avec l'un des outils précédents. Le logiciel dispose également d'une API ouverte en python qui permet le développement d'un *wrapper* spécifique pour implémenter l'interface de FMI [Chatzivasileiadis et al., 2016].

On note que MATLAB est un logiciel propriétaire, alors que Modelica est un langage open-source, qui peut être simulé soit via le logiciel propriétaire Dymola, soit le logiciel libre et open-source OpenModelica. De plus, le développement de ce langage ainsi que celui du standard FMI sont dirigés par la même organisation *Modelica Association*, et les outils de modélisation associés implémentent généralement les toutes dernières versions du standard (et même les bêta-fonctionnalités). La transformation du modèle d'un composant en FMU se fait de manière très transparente pour l'utilisateur, car il n'y a que peu ou pas d'adaptation requise à partir d'un modèle simulable pour le rendre transformable [Elsheikh et al., 2013]. Ainsi, l'export d'un modèle en FMU pourra être réalisé directement par le développeur métier du modèle sans connaissances supplémentaires.

4.3.2 Simuler le système d'information avec FMI

La simulation d'un système d'information est une problématique qui soulève de nombreuses difficultés. Selon la définition de l'urbanisme, le SI est un système complexe représentant l'ensemble des moyens mis en place pour traiter l'information, qu'ils soient humains, logiciels ou matériels. Définir l'ensemble du périmètre d'un tel système, et la bonne façon de le simuler, ne fait pas l'objet de cette thèse. Nous redirigeons le lecteur intéressé vers les travaux de thèse de Rachida Seghiri [Seghiri, 2016] qui proposent une approche pour appliquer les principes de l'architecture d'entreprise à la modélisation et la simulation du SI d'un smart grid.

4. https ://fr.mathworks.com/products/connections/product_detail/modelon-fmi-toolbox.html
5. https ://www.3ds.com/products-services/catia/products/dymola/
6. https ://openmodelica.org

Les méthodes de modélisation et de simulation choisies sont du ressort de l'expert concerné par la constitution de la FMU. Nous abordons donc cette question d'un point de vue plus concret, en considérant que le modèle dynamique du SI doit représenter les différents processus de manipulation d'information, généralement réactifs, basés sur des évènements discrets, des algorithmes procéduraux et des transitions instantanées entre états. Notre intérêt est de fournir des outils compatibles avec ce genre de modèle de calcul et le standard FMI.

4.3.2.1 Modélisation et simulation

Les comportements du Smart Grid à modéliser ici sont généralement ceux des applications de pilotage et de contrôle déployées sur le réseau électrique. La modélisation est une pratique courante dans l'ingénierie logicielle, mais vise généralement à la production du code final de l'exécutable, sans passer par l'étape de simulation. Des experts en informatique effectuent à la place des tests unitaires et fonctionnels directement sur le code afin d'analyser et de prédire le comportement de leur application une fois déployée. Or utiliser le logiciel final dans une simulation revient à effectuer des simulations dites *software-in-the-loop* (SIL). Dans le cadre de smart grids possédant de nombreuses applications de contrôle local et décentralisé, l'intérêt de ce genre de solution est très limité car il demande un investissement important.

Le consensus est que les modèles doivent rester simples afin de permettre un gain de temps optimal en privilégiant les tests et modifications sur les modèles plutôt que sur la solution finale. Par exemple, le langage UML apporte une abstraction plus élevée pour représenter le comportement applicatif que les langages de programmation textuels. Il est déjà utilisé pour représenter des processus de traitement de l'information dans les approches d'architecture d'entreprise [Armour et al., 2003]. L'OMG propose un langage standard dérivé (sous-ensemble) de UML nommé *foundational UML* (fUML). Ce langage a la particularité d'avoir une syntaxe complètement exécutable, et peut donc être utilisé pour simuler des comportements réactifs manipulant des évènements discrets. Les travaux de thèse précédemment cités [Seghiri, 2016] proposent d'ailleurs de simuler un système d'information de smart grid à l'aide de modèles exécutables fUML, en se basant sur le cadre d'architecture TOGAF.

En pratique, fUML est très bien adapté à la représentation de la donnée et des échanges entre les composants fonctionnels d'un système. En revanche, la modélisation de processus complexes ou traitant de nombreuses données est possible mais impose de dessiner un diagramme très détaillé. Le langage textuel ALF (*Action Language for fUML*) fut proposé pour tenter de compenser cette limite [Seidewitz and Tatibouet, 2015], mais n'introduit pas de nouvelle sémantique et reste soumis à la même expressivité que le langage graphique. L'outil de modélisation Papyrus fournit au travers de son plugin Moka une implémentation de la sémantique d'exécution de fUML. Il propose également un point d'extension permettant d'interconnecter l'exécution du modèle avec du code Java en utilisant les actions opaques du langage [Guermazi et al., 2015, Seghiri et al., 2016]. Cette méthode peut également être utilisée pour communiquer avec des outils externes.

En effet, bien qu'on ait montré l'intérêt des modèles de simulation par rapport au code applicatif final, un code informatique peut néanmoins être vu comme un modèle, car il reste l'option la plus pratique et rapide pour décrire un comportement algorithmique complexe. Les langages comme Java, C/C++ ou python sont donc à considérer lors de la réalisation d'un modèle du système d'information du Smart Grid.

4.3.2.2 Utilisation avec FMI

Le choix de l'outil et du langage à utiliser pour la modélisation des comportements applicatifs dépend des besoins du cas d'étude à simuler. Papyrus et Moka implémentent une fonctionnalité d'export d'un modèle fUML en FMU, à condition d'implémenter un profil spécifique permettant de définir l'interface de la FMU. En revanche, le fonctionnement en "boîte-noire" *plug and play* de la FMU complexifie les possibilités d'interconnexion avec des outils externes.

Pour des besoins algorithmiques plus complexes, il est possible de ré-implémenter l'interface C fournie par le standard FMI avec des langages compatibles (C, C++, etc.) [Chatzivasileiadis et al., 2016], ou d'utiliser certaines bibliothèques déjà développées dans certains langages :

— FMU SDK par QTronic [7] et ses différents *forks* dans d'autres langages : bibliothèque en C implémentant les interfaces de FMI et offrant un mécanisme d'extension et des scripts de compilation. Requiert néanmoins beaucoup d'opérations manuelles, telles que l'écriture du fichier XML de description de l'interface FMU, et la compression au format FMU.

— JavaFMI par SIANI : *framework* écrit en Java constitué de plusieurs outils permettant d'importer ou exporter des FMU à partir de code écrit en Java [Galtier et al., 2017a]. Les deux outils principaux sont listés ci-dessous :

1. fmi-framework : bibliothèque en java implémentant l'interface de FMI et permettant d'implémenter le comportement de simulation via un mécanisme d'extension ;
2. fmi-builder : exécutable permettant de convertir automatiquement un JAR implémentant fmi-framework en FMU (via une commande ou une interface graphique).

4.3.3 Simuler le réseau de télécommunications avec FMI

De même que pour le domaine électrotechnique, la simulation est une pratique couramment utilisée dans le domaine des télécommunications. On trouve dans la littérature de nombreuses références [Chatzivasileiadis et al., 2016, Mets et al., 2011, Couto and Pierce, 2017] à plusieurs logiciels dont les principaux sont NS-2 / NS-3, OMNeT++, OPNET, Qualnet, J-Sim, etc.

Une rapide recherche sur le site de l'IEEE par exemple donne la fréquence de citation de ces différents logiciels, comme illustré dans la figure 4.4.

7. https ://www.synopsys.com/verification/virtual-prototyping/virtual-ecu/fmu-sdk.html

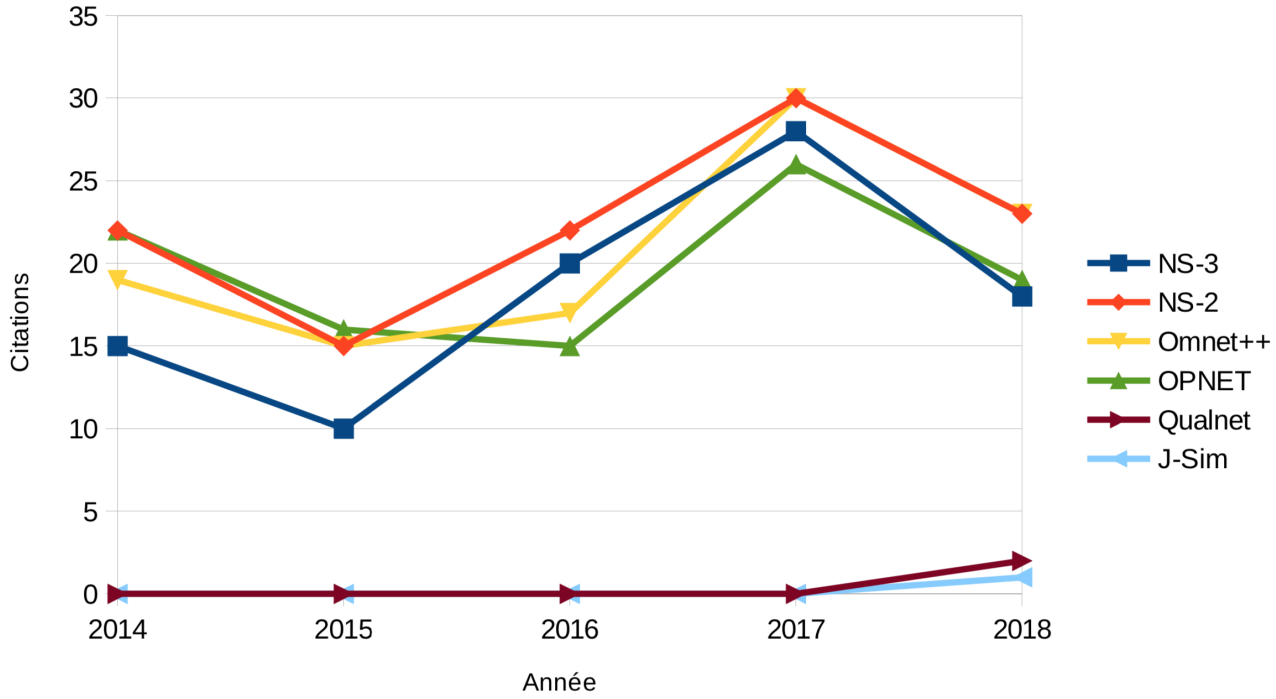

FIGURE 4.4 – Fréquence de citation de quelques uns des principaux logiciels de simulation télécoms parmi les publications de l'IEEE

Malheureusement, aucun d'entre eux ne supporte nativement l'import ou l'export de modèles au format FMU. Cela peut s'expliquer par le fait que les modèles de télécommunications ne manipulent que des évènements discrets, correspondant aux émissions / réceptions de paquets numériques. Il n'y a que peu d'intérêt à maintenir une fonctionnalité d'export vers un format qui n'offre qu'une faible compatibilité avec ce genre de modèles.

Dans un scénario de cosimulation FMI, la FMU d'un modèle de télécommunications aura sans doute le rôle du *Discrete Microstep Delay* décrit par Broman & Al. dans ses travaux [Broman et al., 2015]. L'intégration de ce genre de comportements dans un scénario de cosimulation représente pour les auteurs l'un des principaux verrous dans les outils et standards actuels, comme la norme FMI, pour pouvoir simuler des CPS.

L'environnement de cosimulation de CPS VirGIL [Chatzivasileiadis et al., 2016] utilise le logiciel OMNeT++ pour simuler les transmissions de données via le réseau de télécommunications. Ces travaux montrent notamment qu'il est possible de développer un adaptateur pour rendre le logiciel OMNeT++ compatible avec le standard FMI.

4.4 Conclusion du chapitre

Les standards permettent de résoudre plusieurs difficultés techniques liées aux interactions entre les différents outils de simulation intervenant dans la cosimulation, mais facilitent également la modularité et les collaborations entre plusieurs modeleurs. Au contraire des interfaces propriétaires, les interfaces des outils et la façon dont les données peuvent être échangées entre eux sont normalisés, et conduisent les éditeurs de logiciels à implémenter eux-même la compatibilité de leurs outils avec ces standards.

Néanmoins toutes les difficultés ne peuvent pas être aujourd'hui résolues uniquement avec ces standards, principalement dans le cadres de modèles hybrides, mélangeant à la fois des paradigmes discrets et continus. Différentes méthodes sont employées dans l'état de l'art pour permettre la cosimulation hybride, mais présentent différentes limites, dont l'une des plus importantes est de nécessiter la modification des normes de cosimulation, entraînant la perte de certains avantages de l'utilisation d'un standard.

Deuxième partie

Contribution : Un Environnement de Cosimulation de Smart Grid

5

Mise en place du cadre des travaux : une démarche de simulation d'un Smart Grid avec FMI

Sommaire

5.1 Choix de se limiter au standard FMI 60
5.2 Formalisation d'une démarche pour cosimuler un Smart Grid 60
5.2.1 Positionnement dans le processus de conception 60
5.2.2 Les acteurs 61
5.2.3 Les étapes de la démarche 62
5.2.3.1 Phase 1 : Modéliser la solution 62
5.2.3.2 Phase 2 : Configurer la cosimulation FMI 64
5.2.3.3 Phase 3 : Exécuter et analyser les résultats 64
5.2.4 Contraintes et difficultés dans l'exécution de la démarche 65
5.3 Gérer les échanges discrets avec FMI 66
5.3.1 Problème et définitions 66
5.3.2 Solution proposée 67
5.3.2.1 Composant d'encodage 68
5.3.2.2 Composant de décodage 69
5.3.2.3 Enchaînement des composants 69
5.3.3 Discussion 71
5.4 Conclusion du chapitre 72

La partie précédente nous a permis de déterminer que la cosimulation valide différentes contraintes posées par le contexte industriel dans les études de simulation de Smart Grid. Si cette technologie commence à faire ses preuves sur des systèmes industriels multi-domaines, les travaux de recherche se focalisent généralement sur les contraintes de l'interconnexion de simulateurs spécifiques. Il y a un manque de résultats quant à une méthodologie permettant d'arriver à la construction des modèles de simulation à partir d'un système à concevoir. La première section de ce chapitre explique les raisons de notre choix de proposer une démarche autour de l'utilisation du standard FMI pour permettre

la cosimulation. La deuxième section formalise la démarche dans laquelle s'inscrit la validation du comportement d'un système par la cosimulation FMI, en identifiant notamment les acteurs impliqués et les étapes à réaliser pour atteindre cet objectif. Puis, la troisième section propose une convention d'utilisation du standard FMI permettant l'échange de signaux discrets entre unités de simulation FMU, nécessaire dans les cosimulations de CPS.

5.1 Choix de se limiter au standard FMI

Comme il a été montré dans la section 4.2, FMI est adapté à l'échange de signaux discrets, tandis que HLA est adapté à l'échange d'évènements discrets. Les deux standards présentent d'importantes limitations pour la simulation de systèmes cyber-physiques (CPS), mélangeant discret et continu, et les approches utilisant des interfaces spécifiques, ou détournant un standard existant limitent les possibilités de collaboration entre entreprises, et portent atteinte au **critère 3 de collaboration**.

Nous pouvons ainsi positionner FMI par rapport à HLA selon certains des critères données en introduction (sous-section 1.3.1) :

— Parce qu'il est naturellement adapté à des comportements continus, FMI est déjà utilisé dans le secteur énergétique pour la simulation des comportements électriques [Elsheikh et al., 2013]), secteur dans lequel sont menées les études sur les Smart Grids (meilleur selon le **critère 1 de coût**).

— FMI contrairement à HLA dispose d'un format permettant facilement l'échange de composants simulables qui garantissent la propriété intellectuelle de leur propriétaires grâce au format FMU (meilleur selon le **critère 4 de propriété intellectuelle**).

— Les deux standards permettent la collaboration et l'itération (**critères 3 et 4**) de la même manière.

De plus, les travaux autour du développement de DACCOSIM ont montré qu'il était possible de déployer une cosimulation sur un groupe de calculateurs, ou "cluster" [Galtier et al., 2017b], permettant ainsi la simulation de modèles de grande taille (adéquation avec le **critère 5 de scalabilité**).

Pour ces différentes raisons nous choisissons de présenter dans notre contribution une démarche de simulation de Smart Grids basée sur et compatible pleinement avec FMI.

5.2 Formalisation d'une démarche pour cosimuler un Smart Grid

5.2.1 Positionnement dans le processus de conception

La simulation permet d'estimer le fonctionnement d'un système réel sans y avoir accès, souvent parce qu'il n'existe pas encore. Le but du processus de conception d'un système non-existant est de définir la solution *optimale* répondant à un besoin, en fonction de certains objectifs et contraintes. Si le but de

la simulation n'est pas directement de fournir la solution optimale, elle s'avère très utile dans les cas où une résolution analytique n'est pas possible. Elle s'inscrit alors dans un cycle itératif, en permettant de tester et de comparer des propositions successives, jusqu'à obtenir une solution satisfaisante (mais pas forcément optimale) au problème identifié.

Nous présentons dans cette section une démarche en plusieurs phases formalisant cette approche itérative. Ces phases relativement classiques reprennent celles déjà identifiées dans certains travaux [Quesnel, 2006, Galán et al., 2009], mais nous les détaillons ici en explicitant les étapes et le rôle des acteurs dans le contexte d'une cosimulation avec FMI. Nous établirons dans le chapitre 8 la façon dont nos autres contributions s'inscrivent dans cette démarche et supportent les tâches des différents acteurs.

La première étape de la première phase consiste à choisir et expliciter le système devant être simulé. Il est nécessaire d'avoir déjà identifié les attendus de la solution, et les critères sur lesquels l'efficacité de la solution à simuler sera jugée. Nous considérons dans la présentation de cette démarche que la simulation est utilisée pour valider le comportement d'un système non-existant, ce qui correspond au cas le plus courant. Le cas où le système existe déjà et ne sera pas modifié présente quelques différences, principalement au niveau de l'objectif de la simulation. La majeure partie de notre démarche pourra s'appliquer également dans ce cas, mais les itérations ne viseront pas à faire évoluer le système vers une meilleure solution. Elles auront pour but de tester différents scénarios et possibilités de configuration du système existant.

L'objectif de notre démarche est de guider les différents acteurs impliqués dans le processus de conception et de simulation d'un système dans la mise en place d'une **unité de cosimulation FMI**, puis dans le développement des itérations successives de cette unité jusqu'à obtenir le modèle d'une solution satisfaisante.

5.2.2 Les acteurs

La modélisation d'un Smart Grid pour la simulation requiert des connaissances dans chacun des domaines techniques impliqués dans un tel système, ainsi que dans les méthodes et outils de simulation associés à ce domaine. Le choix de la cosimulation impose des contraintes supplémentaires de couplage entre les simulateurs, et fait appel à des connaissances spécifiques en architecture logicielle et parfois matérielle. L'utilisation d'un standard de cosimulation tel que HLA ou FMI permet la réutilisation d'outils génériques – lorsqu'ils existent – et simplifie les actions et compétences mises en œuvre pour configurer et exécuter une cosimulation.

Nous identifions plusieurs profils d'expertise pouvant être impliqués dans un processus de cosimulation :

Le porteur du besoin. Profil identifiant la ou les personnes en charge de transcrire le besoin attendu aux autres acteurs du processus. Il fournit les spécifications du système à modéliser et simuler.

Les architectes de la solution. Profils identifiant la ou les personnes qui sauront concevoir et proposer une solution pouvant répondre aux spécifications attendues. Ces profils peuvent être spécialisés pour chaque domaine technique étudié.

Les experts en modélisation et simulation (M&S). Profils identifiant la ou les personnes capables de produire les différents modèles attendus de la cosimulation, implémentant le comportement de la solution retenue.

Le chargé de cosimulation. Profil identifiant la ou les personnes en charge de mettre en place et configurer la plateforme de cosimulation à partir des modèles de simulation fournis. Il s'agit également du référent sur la technologie ou le standard utilisé, FMI dans notre cas.

Plusieurs profils peuvent être incarnés par la même personne, de même que les responsabilités d'un profil peuvent être portées par plusieurs personnes. Par exemple, il est courant qu'un expert en modélisation ait les connaissances nécessaires pour concevoir lui-même la solution qu'il souhaite implémenter en fonction des spécifications. De même, la responsabilité d'adapter un modèle de simulation en une unité de simulation au format compatible avec la plateforme de cosimulation peut parfois être également portée par l'expert en modélisation et simulation plutôt que l'expert en cosimulation. C'est d'autant plus vrai lorsque il s'agit d'une pratique courante dans son domaine, ou lorsque les outils de simulation implémentent déjà cette fonctionnalité.

Néanmoins, l'intérêt de notre démarche est de limiter le besoin d'un acteur multi-profils sachant "tout faire", contrainte en accord avec le cadre d'application industriel. L'un des objectifs identifiés est donc bien de faciliter la coordination du travail produit par les différents acteurs.

5.2.3 Les étapes de la démarche

La figure 5.1 présente les différentes *phases* du cycle itératif de simulation d'un Smart Grid par cosimulation.

Chaque phase est décrite dans la suite de cette partie par un ensemble d'étapes, correspondant aux actions à entreprendre lors du premier déroulement.

5.2.3.1 Phase 1 : Modéliser la solution

La phase *"modéliser la solution"* produit une implémentation du comportement d'une solution répondant au besoin identifié. À l'issue de cette phase, les acteurs de la démarche disposent de tous les modèles de simulation représentant la dynamique du comportement du Smart Grid à valider. Les interfaces composées des entrées et sorties de chaque modèles sont identifiées, ainsi que les contraintes de couplage entre ces interfaces. Cette phase implique les étapes suivantes :

1. Définir une solution. Les *architectes de la solution* définissent une cible du système pouvant répondre aux besoins et aux contraintes exprimés par le *porteur du besoin*.

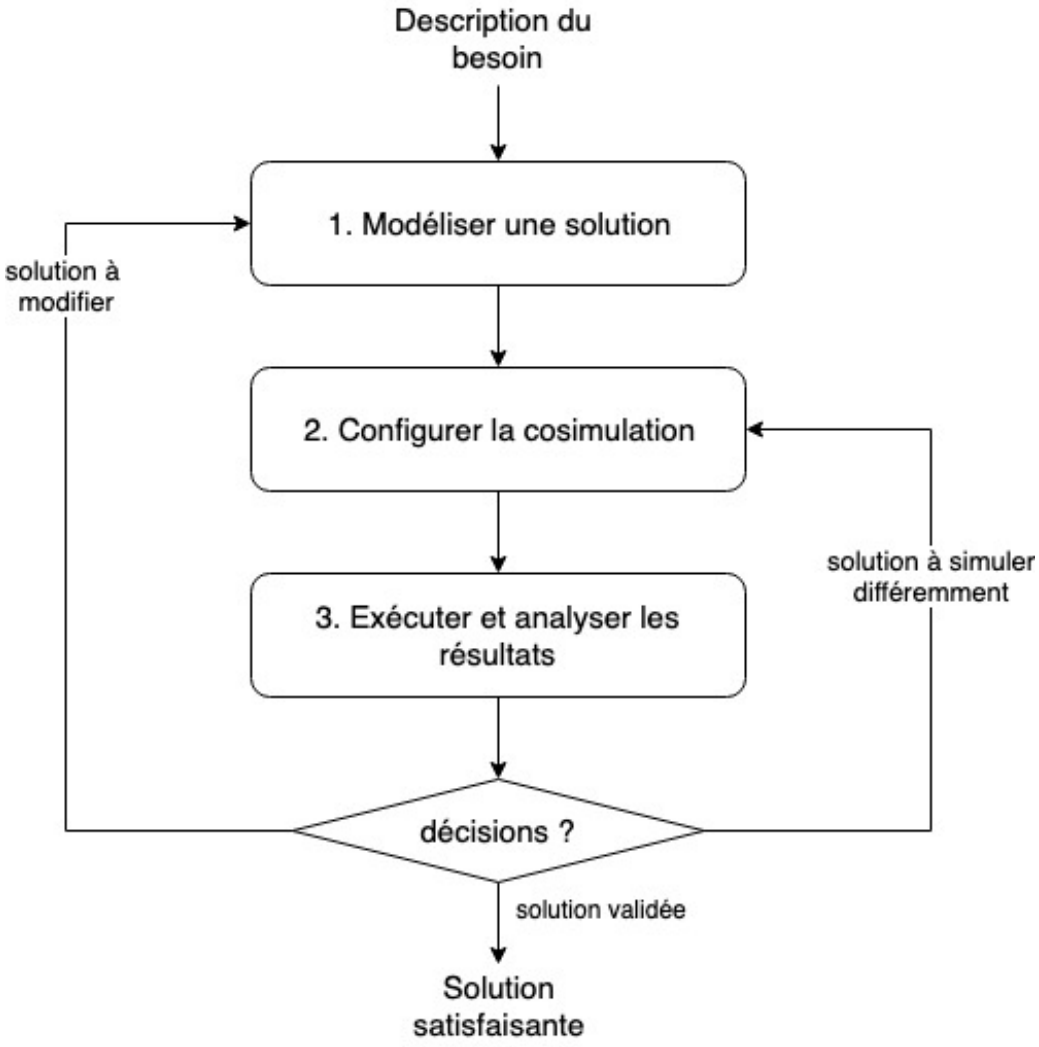

FIGURE 5.1 – Enchaînement des phases de cosimulation d'un Smart Grid

2. Répartir les responsabilités. Les *experts en M&S* identifient et choisissent les différents comportements de la solution à simuler, puis les attribuent aux modèles de simulation qui interviendront dans la cosimulation. Chaque expert a la responsabilité de son modèle et des outils qu'il souhaite employer.

3. Définir les interconnexions et interfaces. Les *experts en M&S* s'entendent ensemble sur les signaux de données devant être échangés entre les modèles au cours de la simulation. À l'issue de cette étape, chaque *expert en M&S* possède la spécification de l'interface qu'il doit implémenter dans son modèle.

4. Produire les modèles de simulation. En fonction des comportements qui lui sont attribués, chaque *expert en M&S* produit un modèle de simulation en respectant l'interface qui lui est imposée.

5.2.3.2 Phase 2 : Configurer la cosimulation FMI

La phase *"configurer la cosimulation FMI"* construit l'unité de cosimulation à partir des différents modèles de simulation produits lors de la phase précédente. À l'issue de cette phase, toutes les unités de simulation sont fournies au format FMU, et l'algorithme *master* de cosimulation est prêt à être exécuté afin de produire des résultats.

1. Adapter les modèles en FMU. Chaque modèle de simulation est adapté en FMU, soit par l'*expert M&S* qui l'a développé s'il possède cette compétence, soit par le *chargé de cosimulation*.
2. Implémenter le scénario de cosimulation dans le *master FMI*. Le *chargé de cosimulation* crée ou paramètre l'algorithme *master* de cosimulation FMI avec les liens vers les FMU et les connexions entre les ports de sortie des unes vers les ports d'entrée des autres, en fonction de ce qui a été défini dans la phase 1. Il configure également les paramètres d'exécution comme la date de début et de fin, la stratégie d'évolution du pas de temps, et les conditions initiales.

5.2.3.3 Phase 3 : Exécuter et analyser les résultats

La phase *"exécuter et analyser les résultats"* détermine les actions à entreprendre concernant la solution cible définie à la phase 1.

1. Évaluer les résultats de cosimulation. Le *chargé de cosimulation* exécute la cosimulation et vérifie que celle-ci s'est déroulée sans erreurs. Il est important d'être critique sur la validité des résultats. Des incohérences entre les modèles de simulation, ou une mauvaise configuration du *master* de cosimulation, peuvent ne pas causer d'erreur d'exécution, mais fournir des résultats faussés. Les *experts en M&S* sont les mieux placés pour relever ces défauts, car ce sont ceux qui connaissent le mieux leurs modèles. Si les résultats sont acceptés, les *experts en M&S*, les *architectes de la simulation*, ainsi que le *porteur du besoin* évaluent ensemble les performances de la solution proposée.
2. Prendre des décisions d'itération. Selon l'adéquation des performances de la solution modélisée avec les besoins et contraintes du système attendu, plusieurs possibilités sont à considérer. Les cas dans lesquels il est nécessaire d'effectuer un nouveau cycle de la démarche, en reprenant à partir de la phase 1 ou 2, sont :
   — Des erreurs, incohérences ou autres imprécisions ont été détectées dans l'étape précédente. Il est nécessaire de revenir en phase 1 ou 2 pour effectuer les modifications nécessaires pour traiter ces défauts.
   — Le modèle de la solution ne fournit pas des résultats satisfaisant par rapport aux attendus du système final. Il est nécessaire de modifier la solution, et les modèles de simulation de la phase 1.
   — Le modèle de la solution fournit des résultats satisfaisants pour les conditions choisies (paramètres d'entrée, conditions initiales, durée de la simulation, événements exceptionnels, etc.). Il reste néanmoins des conditions à tester avant de valider définitivement la solution, soit par

l'ajout de détails supplémentaires sur les modèles de simulation (retour en phase 1), soit par la modification des conditions de simulation (retour en phase 2).

Le schéma de la figure 5.2 illustre les différents cas possibles d'itération.

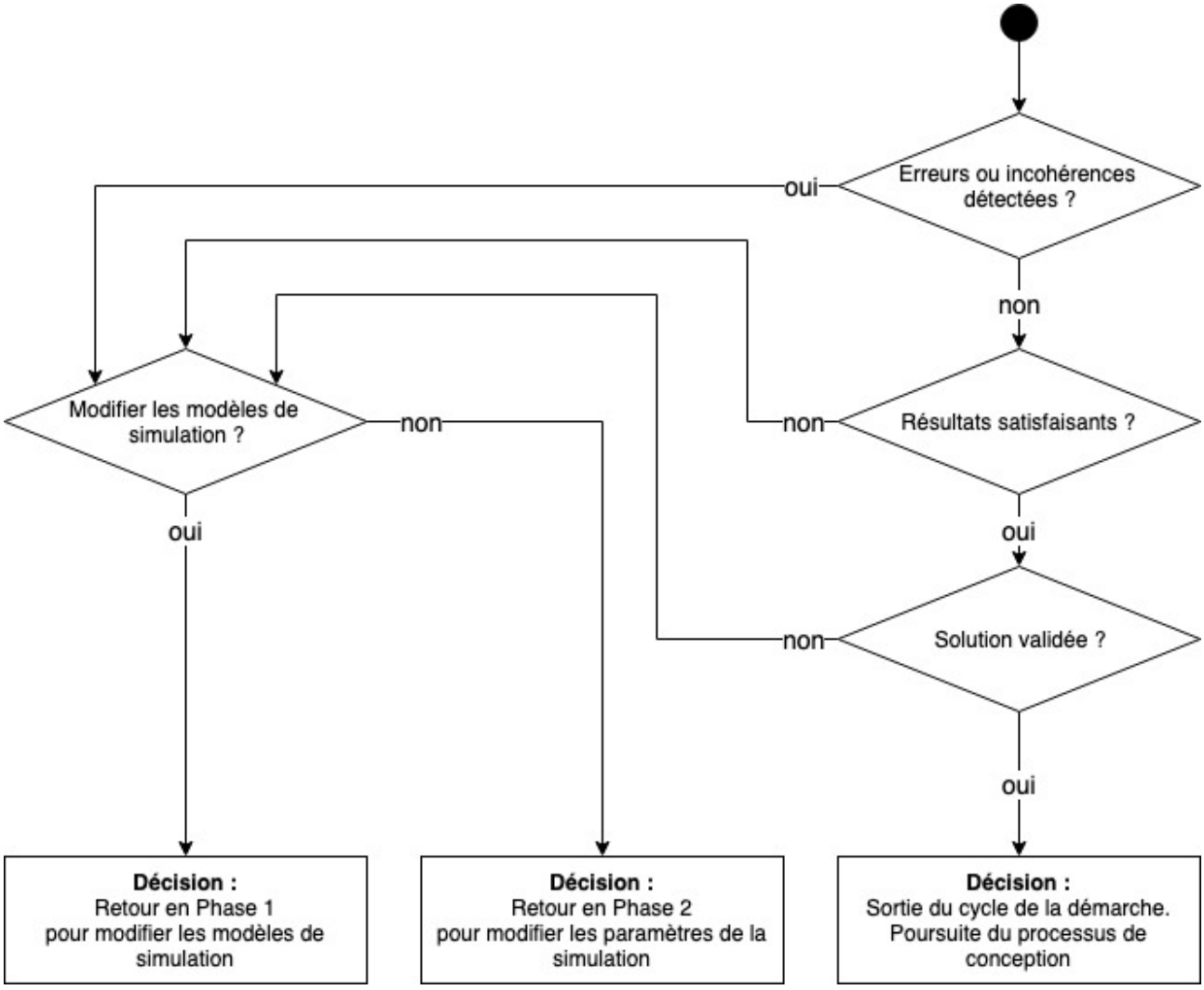

FIGURE 5.2 – Décisions d'itération en fonction de l'analyse des résultats de cosimulation

5.2.4 Contraintes et difficultés dans l'exécution de la démarche

Les étapes sont à réaliser idéalement les unes après les autres dans chaque phase. En pratique, plusieurs retours et échanges sont parfois nécessaires pour lever des ambiguïtés, par exemple :

— Les *experts en M&S* doivent entretenir un certain dialogue, entre eux afin de garantir la cohérence des modèles les uns avec les autres, puis avec le porteur du besoin et l'architecte de la solution afin de garantir la cohérence avec la solution ciblée.

— le *chargé de cosimulation* doit entretenir un dialogue avec les *experts en M&S*, afin de choisir convenablement les conditions initiales et les paramètres de la simulation.

— À chaque modification sur un modèle de simulation – par exemple lors d'une nouvelle itération

du cycle – les impacts sur les autres modèles doivent être évalués et implémentés (modification du design global, modification d'interfaces, etc.)

Ces échanges, s'ils ne sont pas cadrés formellement, peuvent être source de mauvaise interprétation, d'erreurs d'implémentation, et de perte de temps.

Par ailleurs, l'une des principales contraintes techniques de cette démarche est le choix de la norme FMI pour cadrer l'interconnexion entre les modèles de simulation. Bien que l'utilisation de ce standard ne se fasse que dans la seconde phase, elle a un impact dès le début de la démarche. Ainsi, les modèles de simulation développés dans la phase 1 doivent pouvoir être transformés au format FMU. Cela impacte le choix des outils de modélisation et de simulation. De même, les interfaces des modèles et les échanges identifiés doivent pouvoir être transposés en ports et connexions FMI, sachant que :

— seuls cinq types de donnée primitifs sont supportés par FMI (Integer, Real, String, Boolean, Enumeration) ;
— les connexions FMI sont statiques : elles ne supportent pas le changement de type, de destination, et l'absence de valeur échangée au cours de la cosimulation ;
— les signaux échangés via une connexion FMI seront discrétisés par le fonctionnement par pas-de-temps non-nuls de l'algorithme du *master* FMI.

5.3 Gérer les échanges discrets avec FMI

FMI est conçu pour interconnecter des unités de simulation produisant et consommant des signaux temporels continus. Comme évoqué précédemment (section 5.1), cette technique de cosimulation présente de nombreux avantages dans un contexte industriel, notamment parce qu'elle permet de conserver la propriété intellectuelle des modèles impliqués. Néanmoins, pour des simulations impliquant des processus de traitement de l'information et des réseaux de communication, qui reposent sur des événements discrets, il est indispensable de pouvoir échanger également des signaux discrets.

Cette limitation de FMI est reconnue, et certains travaux proposent des modifications de la norme FMI pour atteindre cet objectif, comme présenté dans la sous-section 4.2.4. Nous préférons privilégier la compatibilité avec les outils existants et la conformité avec le standard afin de faciliter la collaboration. Nous présentons ainsi comment exprimer des signaux discrets à partir de signaux continus.

5.3.1 Problème et définitions

On définit un **signal** comme un vecteur d'information évoluant au cours du temps. On peut le représenter par une fonction définie sur un sous-ensemble de $\mathbb{R}^+$. Un **signal en temps-continu**, ou **signal continu**, est défini sur un intervalle continu de $\mathbb{R}^+$. Un **signal en temps-discret**, ou **signal discret**, est défini uniquement sur un ensemble discret de $\mathbb{R}^+$, et chacun des points de cet ensemble est un **évènement** de ce signal. On considère dans la suite de cette partie que tous les signaux prennent

des valeurs pouvant être portées par un signal FMI, donc appartenant à un ensemble $\mathcal{V}$ qui peut être l'un des ensembles suivants : $\mathbb{R}$; $\mathbb{Z}$; $\mathbb{S}$ l'ensemble des chaînes de caractères ; ou $\mathbb{B} = \{\textbf{true}, \textbf{false}\}$ l'ensemble des valeurs booléennes.

Un signal FMI, donc un signal entrant ou sortant d'une FMU, n'est considéré par le *master* de cosimulation qu'à certains **points de communication**. Plusieurs stratégies peuvent être mises en place pour définir ces points de communication, simples comme le choix d'un pas de temps constant, ou plus élaborées comme le choix d'un pas de temps qui s'adapte en fonction des valeurs prises par des signaux FMI lors des points de communication précédents. Certaines peuvent même analyser la valeur d'un signal et décider de retourner la simulation à l'état d'un point précédent (fonctionnalité de *rollback*), et choisir un nouveau pas de temps, mais il y a toujours une indépendance entre le choix d'un nouveau point de communication et l'évolution future d'un signal.

Soit x un signal discret sur $\mathcal{E}$, un ensemble discret représentant les dates des évènements discrets, tel que :

$$x : \mathcal{E} \to \mathcal{V} \quad \text{où} \quad \mathcal{E} \text{ est inclus dans l'intervalle de simulation.}$$

Si l'on veut utiliser x en tant que signal échangé par FMI, l'ensemble des points de communication de la cosimulation $\mathcal{C}$ étant indépendant de l'ensemble des évènements $\mathcal{E}$, il y a une grande probabilité que :

— $\mathcal{E} \cap \mathcal{C} \subset \mathcal{C}$: le signal n'est pas défini pour certains points de communication, ce qui n'est pas représentable avec un signal FMI (pas d'élément "valeur indéfinie" dans $\mathcal{V}$). (**pb. 1**)

— $\mathcal{E} \cap \mathcal{C} \subset \mathcal{E}$: certains évènements du signal d'origine sont manqués par le *master* et les autres FMUs. (**pb. 2**)

La Figure 5.3 illustre ces cas de figure.

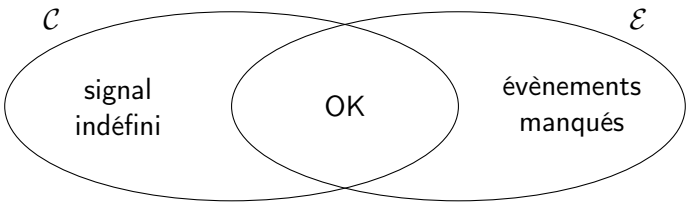

FIGURE 5.3 – Utilisation d'un signal discret en tant que signal FMI

5.3.2 Solution proposée

Afin d'être sûrs que le signal FMI échangé est défini sur l'ensemble des points de communication et éviter **pb. 1**, nous devons produire un **signal continu** sur un intervalle compris entre le premier et le dernier point de communication, c'est-à-dire sur l'intervalle de simulation.

On propose un **composant d'encodage** transformant un signal discret en deux signaux continus couplés pouvant être utilisés en tant que signaux FMI. Le premier signal continu, que l'on nomme

signal d'information, conserve la valeur du dernier évènement apparu jusqu'au prochain évènement. Le second signal continu, appelé **signal de synchronisation** est nécessaire pour exprimer le fait qu'un évènement est bien apparu, sans quoi il n'est pas possible de détecter les suites d'évènements portant une même valeur. En effet, le signal d'information est suffisant dans le cas où deux évènements consécutifs ont une valeur différente, car il est possible de détecter l'instant du changement pour en déduire l'instant d'apparition du nouvel évènement. Mais lorsque le signal est constant, il n'est pas possible de distinguer si un évènement est apparu ou pas.

Suivant ce principe, on propose également un **composant de décodage** pour effectuer l'opération inverse et obtenir un signal discret à partir de deux signaux continus qui ont été échangés par le *master* de cosimulation.

Dans la suite, on notera :
— t_0 et t_f les dates de début et de fin de la simulation, et I_{simul} l'intervalle de cosimulation, tel que $I_{simul} = [t_0\,;t_f] \subset \mathbb{R}^+$. On peut remarquer que t_0 et t_f correspondent respectivement au premier et au dernier point de communication de la cosimulation.
— $x[.]$ au lieu de $x(.)$ si x est un signal discret.

5.3.2.1 Composant d'encodage

Entrée : x un signal discret.
Sortie : u_{info} et u_{sync} deux signaux continus, correspondant respectivement au signal d'information et au signal de synchronisation définis ci-dessus.

Soit $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ un ensemble discret ordonné inclus dans I_{simul}. On pose $i_{\mathcal{E}}(t)$ l'indice du dernier évènement de $\mathcal{E}$ apparu avant ou à la date t :

$$i_{\mathcal{E}}(t) = \max\{i \mid e_i \leq t\} \quad \text{défini sur} \quad [e_1\,;t_f]$$

Les signaux de sortie du composant sont représentés par les définitions suivantes :

$$\forall t \in [e_1\,;t_f] \quad \begin{cases} u_{info}(t) = x[e_{i_{\mathcal{E}}(t)}] \\ u_{sync}(t) = i_{\mathcal{E}}(t) \end{cases} \tag{5.1}$$

Lorsque $t < e_1$, aucun évènement n'est encore apparu. On pose $u_{sync}(t) = 0$. $u_{info}(t)$ peut avoir n'importe quelle valeur, celle-ci n'est pas censée être prise en compte, et sera ignorée par le composant de décodage. Pour plus de sens, on suggère de choisir une valeur initiale correspondant à celle du dernier évènement qui aurait pu se produire avant le début de la simulation. Mais si ce n'est pas possible ou pertinent, la valeur par défaut de l'ensemble image $\mathcal{V}$ peut être une alternative : 0 dans le cas de $\mathbb{R}$ ou

$\mathbb{Z}$, une chaîne vide dans le cas de $\mathbb{S}$, ou la valeur **false** dans le cas de $\mathbb{B}$

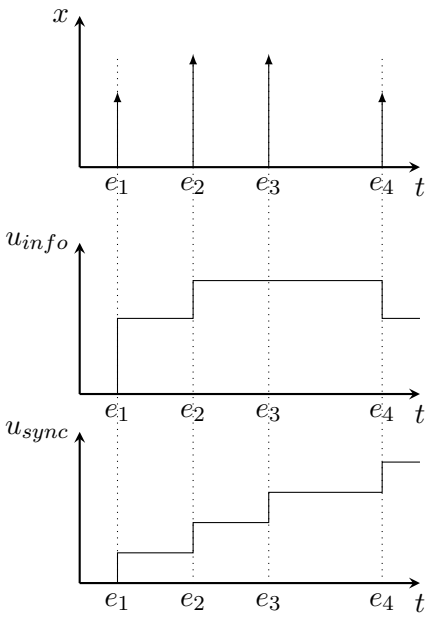

FIGURE 5.4 – Encodage d'un signal x discret en deux signaux u_{info} et u_{sync} continus

5.3.2.2 Composant de décodage

Entrée : v_{info} et v_{sync}, deux signaux continus correspondant respectivement au signal d'information et au signal de synchronisation.
Sortie : y un signal discret.

On définit l'ensemble discret $\mathcal{F} = \{f_0, f_1, \ldots, f_p\} \subset I_{simul}$ comme l'ensemble des points correspondant aux fronts montants de v_{sync}. On peut écrire les éléments de $\mathcal{F}$ comme les éléments d'une suite croissante :

$$\begin{cases} f_0 = \min\{t \mid v_{sync}(t) > 0\} \\ f_k = \min\{t \mid v_{sync}(t) - v_{sync}(f_{k-1}) > 0\} \end{cases} \quad t \text{ variant entre } t_0 \text{ et } t_f$$

On définit le signal y produit par le composant de décodage comme un signal discret sur $\mathcal{F}$, représenté par :

$$\forall f \in \mathcal{F} \quad y[f] = v_{info}(f) \tag{5.2}$$

5.3.2.3 Enchaînement des composants

On a présenté un composant encodant un signal discret en deux signaux continus, puis un composant décodant suivant la même stratégie deux signaux continus en un signal discret.

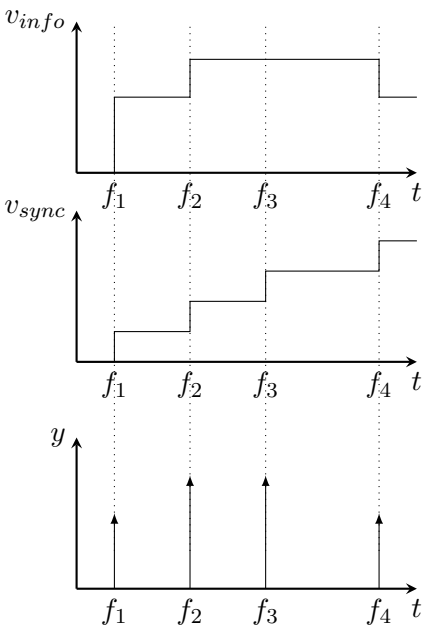

FIGURE 5.5 – Décodage de deux signaux v_{info} et v_{sync} continus en un signal y discret

Néanmoins, tout signal échangé à l'aide de FMI est discrétisé par le *master* de cosimulation sur $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ l'ensemble des points de communication. Parce que FMI est construit pour des échanges continus, la majorité des outils de simulation et d'export de FMU implémentent des techniques de reconstruction du signal d'entrée discrétisé en un signal continu, par interpolation, avant d'être transmis au modèle intégré. FMI supporte cette fonctionnalité en offrant la possibilité d'échanger des signaux entre des FMU avec leurs dérivées à différents ordres. Dans le cas des signaux de sortie de notre composant d'encodage, aucune dérivée n'est échangée, donc les FMU de destination considèreront ces signaux comme constants entre les points de communication. On parle d'interpolation échelon.

u_{info} et u_{sync} en sortie du composant d'encodage sont des signaux en escalier, continus sur I_{simul}. La Figure 5.6 illustre la différence entre le signal u_{info} et u_{sync} sortant d'un modèle et le signal correspondant v_{info} et v_{sync} entrant dans un autre modèle après avoir été échangé par FMI.

On peut formaliser la relation entre u_{info} et u_{sync}, et v_{info} et v_{sync} par :

$$\begin{cases} v_{info}(t) = u_{info}(c_{i_{\mathcal{C}}(t)}) \\ v_{sync}(t) = u_{sync}(c_{i_{\mathcal{C}}(t)}) \end{cases} \quad \text{où} \quad i_{\mathcal{C}}(t) = \max\{i \mid c_i \leq t\}$$

On déduit de l'équation précédente et des équations (5.1) et (5.2) la relation suivante entre y et x :

$$\forall f \in \mathcal{F} \quad y[f] = v_{info}(f) = u_{info}(c_{i_{\mathcal{C}}(f)}) = x[e_{i_{\mathcal{E}}(c_{i_{\mathcal{C}}(f)})}] \tag{5.3}$$

où $c_{i_{\mathcal{C}}(f)}$ est la date du dernier point de communication avant ou à la date f, et $e_{i_{\mathcal{E}}(c_{i_{\mathcal{C}}(f)})}$ est la date du dernier évènement avant ou à la date de ce point de communication.

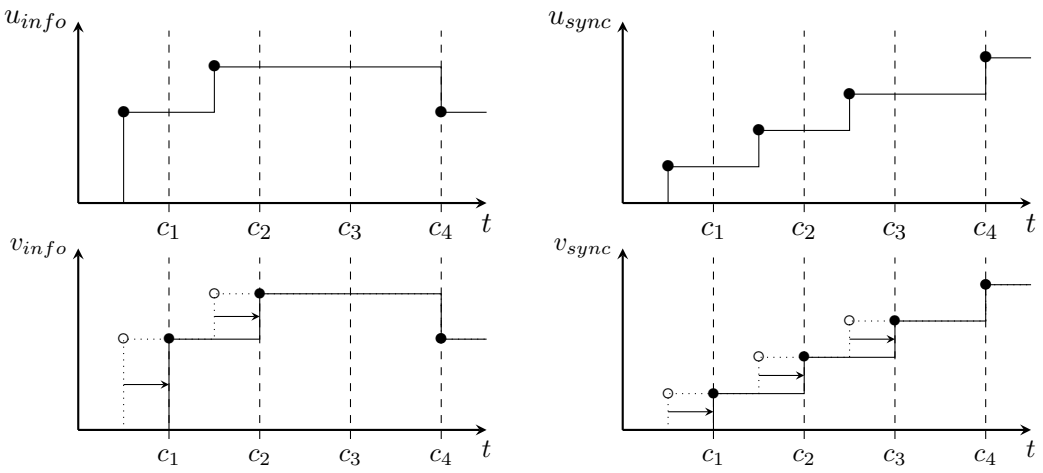

FIGURE 5.6 – Différence entre un signal encodé produit par un modèle et le signal reçu par un autre modèle après échange FMI

Or $\mathcal{F}$ étant construit à partir des fronts montants de v_{sync}, on constate que ceux-ci correspondent tous à un point de communication (comme illustré Figure 5.6). C'est-à-dire que :

$$\forall f \in \mathcal{F}\ ,\ \exists k \text{ tel que} \quad \begin{aligned} f &= c_k \\ c_{i_{\mathcal{C}}(f)} &= c_k \end{aligned}$$

Ainsi on peut simplifier l'équation (5.3) par :

$$y[f] = y[c_k] = x[e_{i_{\mathcal{E}}(c_k)}] \tag{5.4}$$

5.3.3 Discussion

A partir d'un signal discret x produit par un modèle, on sait obtenir un signal discret y, ressemblant à x, dans un autre modèle après échange par FMI. Grâce à nos composants d'encodage et de décodage, on utilise des signaux continus pour l'échange ce qui assure qu'une valeur est définie pour chaque point de communication, résolvant **pb. 1**.

D'après l'égalité (5.3), un évènement reçu à c_k a la valeur d'un évènement émis à $e_{i_{\mathcal{E}}(c_k)}$, qui est la date du dernier évènement émis avant c_k. On a donc

$$c_{k-1} < e_{i_{\mathcal{E}}(c_k)} \leq c_k$$

Ainsi notre solution répond à **pb. 2** en assurant que si un évènement émis n'est pas synchronisé avec un point de communication, il pourra quand même être détecté et reçu, mais retardé de $c_k - e_{i_{\mathcal{E}}(c_k)}$. Ce retard est une limite inhérente au fonctionnement de la version actuelle 2.0 du standard FMI, mais peut être réduit grâce à certaines stratégies intelligentes de choix du pas de temps de simulation.

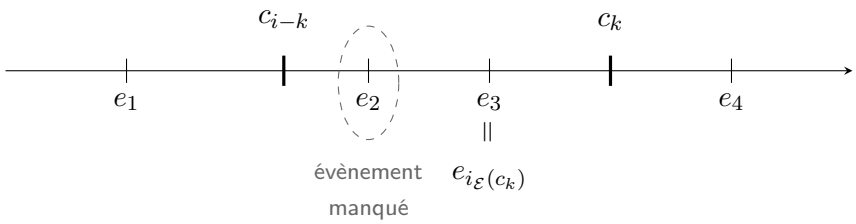

FIGURE 5.7 – Les évènements trop proches sont manqués

Malheureusement, **pb. 2** n'est pas entièrement résolu car les évènements émis entre c_{k-1} et $e_{i_\mathcal{E}(c_k)}$ sont tout de même manqués, comme le montre la Figure 5.7. Notre stratégie ne permet d'exprimer qu'un seul évènement à chaque point de communication, c'est un choix arbitraire de ne considérer que le dernier émis depuis le précédent point de communication. Néanmoins, u_{sync} et v_{sync} portent toujours l'information du nombre d'évènements émis depuis le début de la cosimulation. Ainsi et vérifiant la différence entre $v_{sync}(c_{k-1})$ et $v_{sync}(c_k)$, on peut détecter le nombre d'évènements qui ont eu lieu entre c_{k-1} et c_k. Un *master* de cosimulation suffisamment intelligent et disposant de la fonctionnalité optionnelle *rollback* du standard FMI (retour en arrière) peut décider de rejouer le dernier pas de cosimulation avec une durée plus faible si nécessaire, et ainsi détecter les évènements manqués.

Malgré ces avantages, l'utilisation de ces composants dans les modèles exportés en FMU requièrent des changements potentiellement importants de ces modèles, ainsi que la création de nouvelles connexions FMI dédiées à la synchronisation des FMU (signaux u_{sync} et v_{sync}). Ce type de connaissance est également spécifique aux problématiques de cosimulation, et peuvent ne pas être du ressort et de la responsabilité des équipes de modélisation.

5.4 Conclusion du chapitre

Dans ce chapitre a été proposée une formalisation des différentes étapes nécessaires pour définir et valider la conception d'un système grâce à la cosimulation FMI. Ces étapes sont regroupées au sein d'une démarche en trois phases, et font intervenir un certain nombre de profils de compétence. Cela nous a permis d'identifier un certain nombre de difficultés, d'abord dues à la potentielle multiplicité des acteurs physiques, puis au choix du standard FMI.

Le nombre de profils de compétence nécessaires implique que personne ne peut manuellement valider l'ensemble des modèles de simulation développés. Cela peut causer des incompatibilités entre ces modèles, et une incohérence avec l'objectif de conception initial. Ce risque est accru à chaque nouvelle itération de la démarche, dans laquelle les modèles peuvent être à nouveau modifiés, et les impacts sur les autres modèles difficiles à prévoir.

Notre démarche de cosimulation repose sur le standard FMI pour piloter la simulation des modèles dynamiques, et réaliser les échanges de signaux entre elles. L'une des difficultés due à l'utilisation du

standard FMI est le manque de support natif des échanges de signaux discrets, indispensables dans le cas des simulations de CPS comme les Smart Grids. C'est pourquoi nous avons proposé ici une méthode d'encodage et décodage de ces signaux discrets afin de permettre leur échange via le standard FMI. Néanmoins, le caractère conventionnel de cette méthode implique qu'elle puisse ne pas être respectée, nécessitant une certaine rigueur de vérification et une entente préalable entre les acteurs de la modélisation. On peut également ajouter une seconde difficulté liée à ce standard, qui est l'obligation de produire des unités de simulation au format spécifique *FMU*, format pas toujours supporté par les outils de modélisation et simulation existants. Nous n'avons par exemple trouvé aucun outil pertinent pour simuler le réseau télécoms, qui soit compatible avec FMI.

Le prochain chapitre présentera certains outils que nous avons développés afin de traiter ces difficultés et diminuer l'effort des collaborateurs de notre démarche de simulation de Smart Grid. Ces outils sont fondés sur les principes de l'ingénierie dirigée par les modèles (voir section 3.1), et automatisent certaines opérations grâce à différents générateurs de fichiers.

6

Modéliser et exécuter un scénario de cosimulation de Smart Grid

Sommaire

6.1 fmi4omnetpp : Cosimuler le réseau de télécommunications avec OMNeT++ et FMI . . . 76
6.1.1 Rôle du simulateur télécoms au sein de la simulation du Smart Grid . . . 76
6.1.2 Étendre le fonctionnement d'OMNeT++ . . . 77
6.1.3 Principe et structure de fmi4omnetpp . . . 79
6.2 CosiML : un DSL pour la cosimulation de CPS . . . 81
6.2.1 Objectif de CosiML . . . 81
6.2.2 Métamodèle de CosiML . . . 82
6.3 Génération des artefacts de cosimulation FMI . . . 86
6.3.1 Processus . . . 86
6.3.2 Générateurs d'adaptateur . . . 87
6.3.2.1 Générateur d'adaptateur pour OMNeT++ . . . 89
6.3.2.2 Générateur d'adaptateur pour Java . . . 90
6.3.2.3 Générateur d'apatateur CSV . . . 91
6.3.3 Choix de DACCOSIM en tant que *master* de cosimulation FMI . . . 92
6.3.3.1 Générateur de configuration . . . 92
6.3.4 Générateur de l'exécutable de la cosimulation . . . 93
6.4 Conclusion du chapitre . . . 94

Les outils présentés dans ce chapitre ont pour but de permettre la réalisation de la cosimulation, avec FMI, d'un Smart Grid. Ceux-ci adressent directement certaines des difficultés techniques identifiées au chapitre précédent. La première section présente une extension du logiciel OMNeT++ , un outil de modélisation et simulation du réseau de télécommunications. Cette extension apporte la compatibilité avec le standard FMI 2.0, et permet donc d'évaluer l'impact du réseau télécoms dans la cosimulation FMI d'un Smart Grid. La seconde section présente le langage CosiML, un DSL que nous avons développé dans le but de construire un modèle représentant le scénario de cosimulation attendu. Enfin, la dernière section présente différents outils permettant de générer les artefacts de la cosimulation à partir d'un modèle conforme à ce langage CosiML.

6.1 fmi4omnetpp : Cosimuler le réseau de télécommunications avec OMNeT++ et FMI

6.1.1 Rôle du simulateur télécoms au sein de la simulation du Smart Grid

Plusieurs travaux autour de la simulation de CPS évaluent l'impact du réseau télécoms dans le comportement du système [Yang et al., 2013, Chatzivasileiadis et al., 2016, Cremona et al., 2016]. Le modèle de télécommunications, lorsqu'il est distinct du modèle de comportement applicatif métier, ne produit pas de donnée importante pour les modèles des autres domaines étudiés. Dans le cas de la simulation du smart grid, le modèle du réseau électrique partage des relevés de mesure au modèle du SI et attend en retour des valeurs de consignes et autres ordres de pilotage. Le modèle télécoms simule leur transmission, en appliquant des délais, éventuellement des pertes, mais sans impacter par son existence les interfaces structurelles des autres modèles (la définition de leurs entrées et sorties). Dans le scénario de cosimulation, l'unité de simulation télécoms est reliée aux autres unités de façon à "intercepter" les signaux échangés.

Dans notre approche avec FMI, chaque échange dont nous souhaitons simuler la transmission donne lieu à plusieurs signaux FMI :

— un signal sortant de la FMU émettrice vers la FMU télécoms,

— un signal sortant de la FMU télécoms vers la FMU destinataire

La FMU émettrice et la FMU destinataire peuvent parfois être la même FMU.

Par ailleurs, nous rappelons que le type des ports des FMU, ainsi que les liens entre ports d'entrée et de sortie, sont statiques, et que les ports d'entrée FMI ne peuvent accepter plus d'une connexion. Ainsi, la transformation d'une FMU télécoms dépend fortement du scénario de cosimulation à mettre en place, car un couple de ports d'entrée et de sortie devra être prévu pour chaque échange possible, identifié de manière unique par une donnée, un émetteur et un destinataire. La figure 6.1 montre ces différents cas de figure, ainsi que l'influence de la simulation de la transmission sur les signaux échangés.

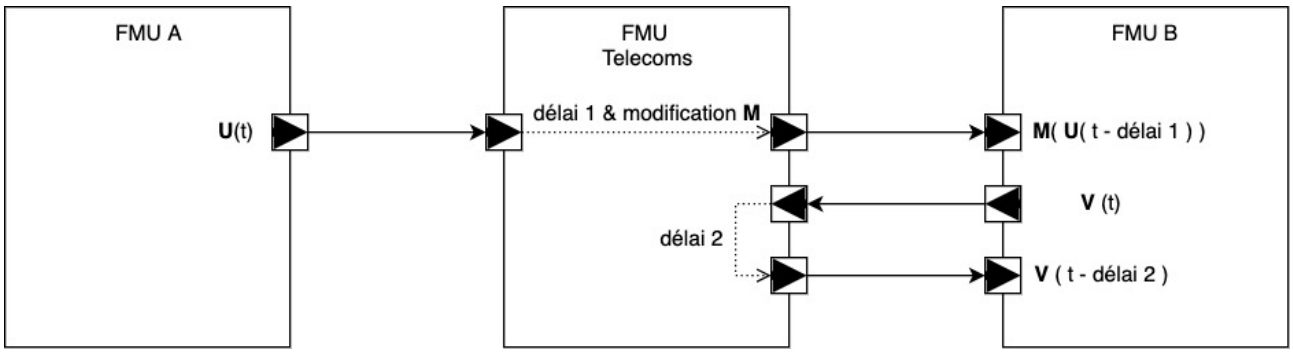

FIGURE 6.1 – Ajout d'une FMU télécoms pour évaluer l'impact de la transmission dans la simulation du système

Nous notons qu'une stratégie différente peut être imaginée, dans laquelle l'unité de simulation té-

lécoms transmet aux autres unités des données spécifiques à son domaine, comme un délai, un taux d'échec, un débit calculé, etc. Ces données sont alors utilisées par les autres unités de simulation dans leurs calculs comportementaux. Nous n'avons pas suivi cette voie, car nous constatons que cela ne respecte pas les pratiques habituelles des équipes de modélisation. Les comportements de transmission correspondent plus à des effets de contraintes qu'à de réels comportements métiers, et les modèles autres que le modèle de télécommunication (modèles comportementaux physiques et applicatifs) ne manipulent pas ce genre d'information.

Comme indiqué dans la section 4.3, les outils de modélisation et de simulation du réseau télécoms les plus utilisés dans le monde industriel n'implémentent pas la possibilité de fournir des unités de simulation au format FMU. Nous avons choisi de développer un outil, fmi4omnetpp, afin d'étendre le fonctionnement du simulateur OMNeT++ et de pouvoir proposer cette fonctionnalité.

6.1.2 Étendre le fonctionnement d'OMNeT++

Un modèle OMNeT++ est constitué de modules communiquant des messages entre eux. Les *modules simples* sont des classes C++ qui étendent les classes du cœur de simulation d'OMNeT++ . Le logiciel propose également un langage textuel spécifique, appelé *Network Description* (NED), afin de représenter l'architecture de ce réseau de modules. Il permet de déclarer les modules simples utilisés, de les connecter entre eux et de les assembler dans des *modules composés*. Un réseau de télécommunication (*Network*) peut être représenté par un réseau de modules, comme illustré dans la figure 6.2 [1].

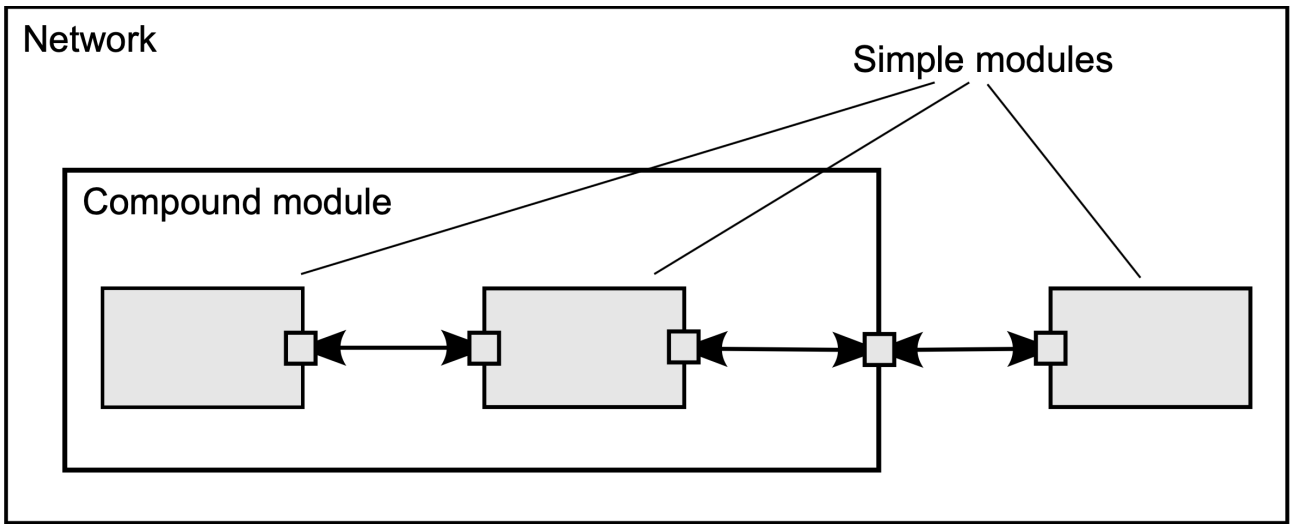

FIGURE 6.2 – Modules simples et composés (extrait de la documentation)

Au cours d'une simulation, les modules s'échangent des messages via des liens de connexion, directement à leur destinataire ou en suivant un chemin particulier. Les messages contiennent des structures de données arbitraires et complexes, décrites dans des fichiers `.msg`. La création et l'envoi de message

1. extrait du manuel en ligne d'OMNeT++ : https ://doc.omnetpp.org/omnetpp/manual/

peuvent être décidés par les algorithmes des modules simples, ou prévus à l'avance dans un fichier de configuration `.ini`.

Ainsi, un modèle OMNeT++ est composé de :
— fichiers `.ned` textuels présentant la topologie et structure du réseau de modules ;
— fichiers `.msg` permettant de définir les types de messages échangés ;
— fichiers sources des modules simples déclarés dans les fichiers NED. Ils sont écrits en C++ et fournis avec les extensions `.c`/`.h`.

Par ailleurs, OMNeT++ fonctionne autour de la notion d'évènement, comme l'envoi ou la réception de message. Lorsque des évènements futurs sont prévus, ils sont insérés dans la pile de futurs évènements **FES** (*Futur Event Set*) selon leur date d'apparition. L'évolution du temps et le déroulement de la simulation se fait de manière discrète en exécutant les évènements de la FES dans leur ordre d'apparition.

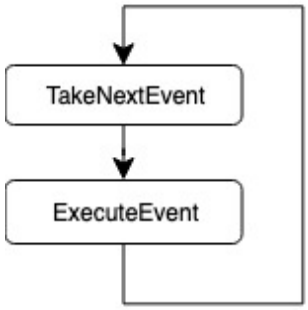

FIGURE 6.3 – Boucle principale d'exécution de OMNeT++

La figure 6.3 illustre les fonctions appelées dans la boucle principale de déroulement de la simulation d'un modèle OMNeT++ :

TakeNextEvent : Cette fonction récupère depuis la FES le prochain évènement devant avoir lieu.

ExecuteEvent : Cette fonction exécute les actions associées à l'évènement en cours, avance la date actuelle à celle de l'évènement, et le supprime. Par exemple, l'exécution d'un évènement "envoi de message" crée un ou plusieurs évènements "réception de message" à une date ultérieure calculée, et les ajoute à la FES.

OMNeT++ dispose de mécanismes d'extension et d'encapsulation, ainsi que d'une documentation développeur très complète. Les principales difficultés liées à l'adaptation d'une unité de simulation OMNeT++ sont les suivantes :

1. Adapter le cycle d'exécution classique de OMNeT++ à celui d'une FMU. L'exécution d'une FMU est dirigée par l'algorithme du *master* de cosimulation et ses appels aux différentes fonctions de l'interface (`fmi2Instanciate`, `fmi2DoStep`, etc.)
2. Adapter la notion FMI d'évolution continue du temps à celle d'évolution discrète, par évènement, de OMNeT++ .

3. Lier la structure de l'interface de la FMU aux évènements d'envoi et de réception de messages dans le modèle.

6.1.3 Principe et structure de fmi4omnetpp

fmi4omnetpp est un outil constitué de deux parties :

1. **Une extension** du cœur de simulation de OMNeT++ , fournissant une nouvelle interface de contrôle de la simulation qui implémente le standard FMI 2.0.
2. **Un outil de transformation automatique** d'un modèle OMNeT++ au format FMU. Cet outil facilite la compilation et la génération des fichiers nécessaires à la production d'une FMU, à partir du modèle réalisé et de la définition des interfaces souhaitées.

Le *FMU SDK* de QTronic GMBH [2] est un outil de développement de FMU à partir d'un code source écrit en langage C. OMNeT++ étant développé en C++, compatible avec C, nous choisissons d'adapter et d'intégrer ce SDK dans `fmi4omnetpp`.

Le déroulement de la simulation d'une FMU est principalement piloté par les appels successifs du *master* aux fonctions `fmi2SetXXX`, puis à la fonction `fmi2DoStep`, et aux fonctions `fmi2GetXXX` (définies dans le tableau 4.1). Occasionnellement, les fonctions `fmi2GetFMUState` et `fmi2SetFMUState` peuvent être appelées pour effectuer un *rollback* de la simulation. Notre extension étend et intègre le cœur de simulation de OMNeT++ afin d'implémenter ces fonctions.

La figure 6.4 présente le processus d'exécution de la simulation suite à l'appel à la fonction `fmi2DoStep` de la FMU. Celui-ci reprend le processus et les fonctions de OMNeT++ utilisées dans son exécution classique (hors FMU) et présentées à la figure 6.3, mais la boucle principale d'exécution ne traite que les évènements ayant lieu pendant le pas de temps simulé. De nouvelles fonctions ont été ajoutées pour faire le lien avec l'interface de la FMU :

CreateFMIMessages : Cette fonction analyse les valeurs des ports d'entrée de la FMU pour en déduire les messages à envoyer via le réseau télécoms Elle crée dans la pile d'évènements, la FES, les évènements "envoi de message" associés.

HandleFMIMessage : Cette fonction est appelée si l'évènement en cours est lié à un message FMI. S'il s'agit d'un évènement "réception de message", alors les ports de sortie associés de la FMU sont mis à jour.

La manière dont sont traduites les valeurs des signaux FMI entrants en évènement interne – et vice-versa – est un cas typique d'échange discret via signaux FMI. Nous nous attendons donc à ce que chaque message modélisé devant être transmis par le réseau de télécommunication respecte notre convention établie en section 5.3, et soit porté par un signal discret encodé sur deux signaux FMI (continus), nommés `<signalName>` et `<signalName>_sync`.

Par souci de flexibilité et de compatibilité avec des FMU ne respectant pas notre convention, nous

2. FMU Software Development Kit, https ://github.com/qtronic/fmusdk

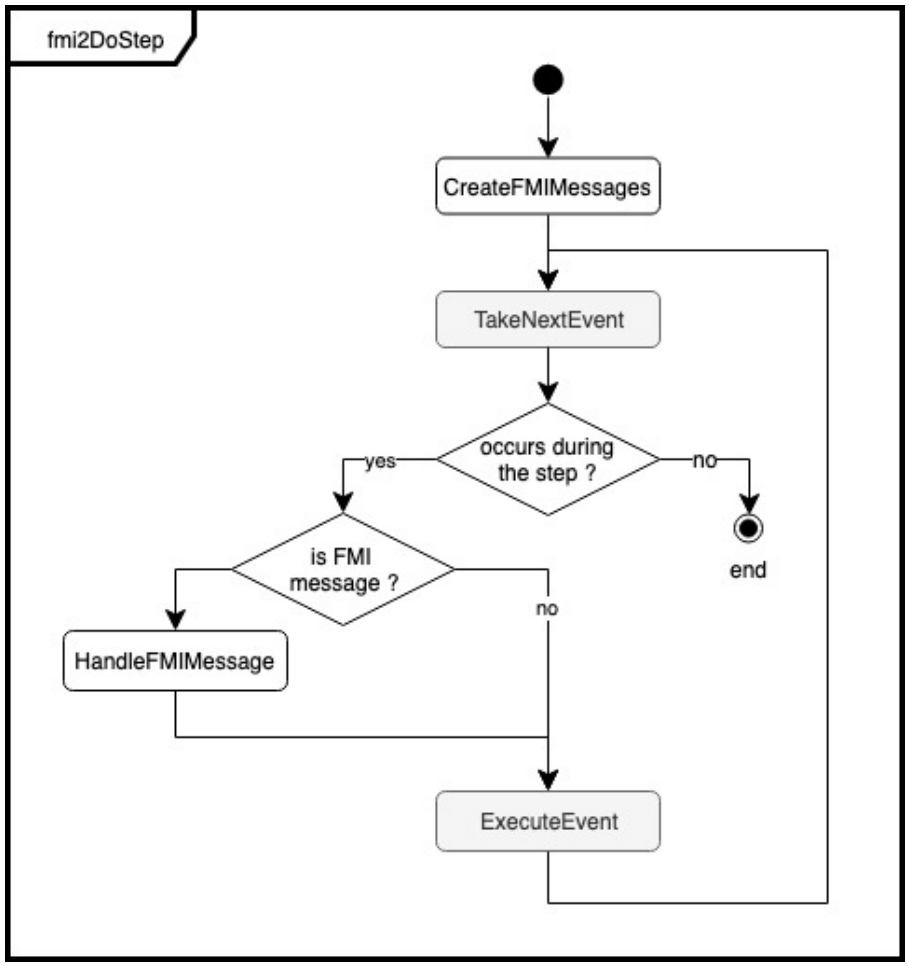

FIGURE 6.4 – Boucle principale d'exécution de OMNeT++

avons tout de même implémenté un fonctionnement alternatif dans le cas où il n'y a pas de signal de synchronisation en entrée (et donc pas de port `<signalName>_sync`). Nous créons alors un nouvel évènement à chaque appel de la fonction `fmi2DoStep`, ce qui correspond à l'échantillonnage d'un signal continu sur la durée de chaque pas de temps.

Enfin, notre outil de transformation automatique peut générer une FMU à partir du modèle réalisé avec notre extension, et d'un fichier JSON de correspondance des messages externes avec les modules internes du modèle OMNeT++ . Il est important de relever que les messages FMI "externes" devant transiter par le réseau de télécommunications modélisé doivent être définis dans notre modèle. C'est de cette façon que le modeleur définit les entrées et sorties de son modèle, et c'est cette information qui est utilisée par notre outil de transformation pour définir la structure des ports d'entrée et sortie de la FMU. Notre outil effectue ainsi les actions suivantes :

1. une lecture des fichiers du modèle afin d'identifier les messages FMI devant transiter sur le réseau, et leur correspondance avec les modules internes ;
2. une génération des fichiers nécessaires à la constitution de l'archive au format `.fmu` (voir sous-section 4.2.3), et du script de compilation des sources du modèle ;

3. l'exécution du script de compilation du modèle, et la compression du dossier des sources au format `.fmu`.

6.2 CosiML : un DSL pour la cosimulation de CPS

6.2.1 Objectif de CosiML

CosiML (*Cosimulation Modeling Language*) est un DSL que nous avons développé afin de représenter l'architecture de la cosimulation à déployer puis exécuter. Il permet de représenter :

— les différentes unités de simulation attendues, avec la description de leur interface et l'emplacement de leurs modèles de simulation ;
— le scénario de cosimulation, c'est-à-dire les connexions présentes entre les sorties et les entrées des unités de simulation ;
— les propriétés nécessaires de configuration de l'exécution, comme la date de début et de fin de la simulation.

Notre langage est construit pour représenter des cosimulations utilisant la technologie FMI, mais nous nous appuyons sur des notions et du vocabulaire génériques afin qu'il puisse s'appliquer à d'autres technologies au fonctionnement similaire. Les différents choix et parti-pris supposent notamment que :

— le langage ne peut pas exprimer des interfaces dynamiques ou des liaisons dynamiques entre unités, c'est-à-dire qui évoluent au cours de la simulation ;
— les interfaces des unités de simulation ont des types de donnée simples (*Real, Integer, Boolean, String*) fixés.

Nous rappelons que l'une des principales difficultés de la simulation de modèles couplés, et notamment la cosimulation, est liée à la complexité de l'architecture logicielle à déployer. En effet, une fois que nous disposons de tous les modèles de simulation représentant notre système, il faut les transformer en FMU en utilisant les outils spécifiques à leur exécution, puis rassembler ces artéfacts au sein d'une même plateforme disposant d'un *master* de cosimulation FMI, et enfin configurer ce *master* et les connexions entre les FMU. Ces opérations sont aujourd'hui principalement manuelles, et requièrent un certain effort de compréhension des modèles de la part du responsable de la cosimulation.

L'intérêt de centraliser les données nécessaires à la cosimulation au sein d'un unique modèle productif est de ne définir cette information qu'une seule fois, de la valider et de s'en servir comme référence lorsque c'est nécessaire. Ainsi, nous souhaitons qu'un modèle conforme à notre langage CosiML puisse être utilisé pour générer et configurer automatiquement l'architecture de la cosimulation :

— **Par un ensemble de contraintes sémantiques** nous assurons la cohérence du couplage entre les interfaces de modèles de simulation hétérogènes, et validons la façon dont ce couplage pourra être simulé sur une architecture de cosimulation respectant la norme FMI.
— **Par le développement de générateurs spécifiques**, nous permettons la génération automatique d'un "exécutable" de la cosimulation.

Les contraintes sémantiques seront exprimées directement sur la syntaxe abstraite de CosiML, au sein

de son métamodèle à l'aide notamment du langage de contraintes OCL. Les générateurs développés jouent eux aussi un rôle dans la sémantique d'exécution du langage, en traduisant les éléments du modèle dans des artéfacts de simulation qui pourront être exécutés par un cosimulateur. Ils peuvent appartenir à l'une des trois catégories suivantes :

1. **Générateur d'adaptateur** : Les modèles de simulation peuvent être fournis avec différents formats, parfois non compatibles avec la technologie de cosimulation. Ces modèles requiert une *transformation* au format compatible, dans notre cas **FMU**. Les *générateurs d'adaptateur* se chargent de créer les adaptateurs nécessaires et les scripts de transformation pour chaque modèle de simulation. En particulier, des modèles manipulant des évènements discrets peuvent être adaptés avec l'intégration des composants de décodage et d'encodage présentés dans la section 5.3 au chapitre précédent. Nous choisissons de gérer de la même manière le cas particulier où le modèle fourni est juste un fichier de données datées et indépendantes, qui doivent être transmises au cours de la cosimulation aux autres unités de simulation.
2. **Générateur de configuration** : Le *master* de cosimulation FMI doit être configuré, notamment avec le scénario de cosimulation et les paramètres de simulation. Ces fichiers de configuration sont générés automatiquement par le *générateur de configuration*.
3. **Générateur de l'exécutable** : Le script de lancement de la cosimulation spécifique à la plateforme d'exécution est généré, afin de minimiser l'action de l'utilisateur pour obtenir les résultats de simulation. Ce script, créé par le *générateur de l'exécutable*, se charge également d'appeler les scripts de transformation créés par les générateurs d'adaptateur pour chaque modèle de simulation.

Ainsi, nous distinguons à l'utilisation une première action de mise en place des artefacts de simulation, avec génération des adaptateurs de modèle et des scripts de configuration et de lancement de la cosimulation. Une seconde action est nécessaire pour exécuter la cosimulation et obtenir les résultats. Nous choisissons de séparer ces deux actions car elles nécessitent chacune des outils différents, et donc peuvent être exécutées sur deux plateformes différentes. Par exemple, les outils de modélisation et de génération des artefacts de simulation requièrent certainement une interface graphique, mais pas forcément une grande puissance de calcul, à l'inverse de l'exécution de la cosimulation, qui peut être déployée sur un cluster de simulation par exemple.

6.2.2 Métamodèle de CosiML

CosiML est un langage manipulant des notions correspondant au vocabulaire générique de la cosimulation présenté dans la section 4.1. La figure 6.5 illustre dans un diagramme Ecore une partie de ce métamodèle, contenant les notions principales de ce langage et les liens entre elles.

Les descriptions des différents éléments sont données ci-dessous :

CosimulationModel : L'élément racine du modèle, il contient les paramètres de la cosimulation, qui sont la date de début (`starttime`), de fin (`stoptime`) et la durée du pas de temps (`stepsize`)

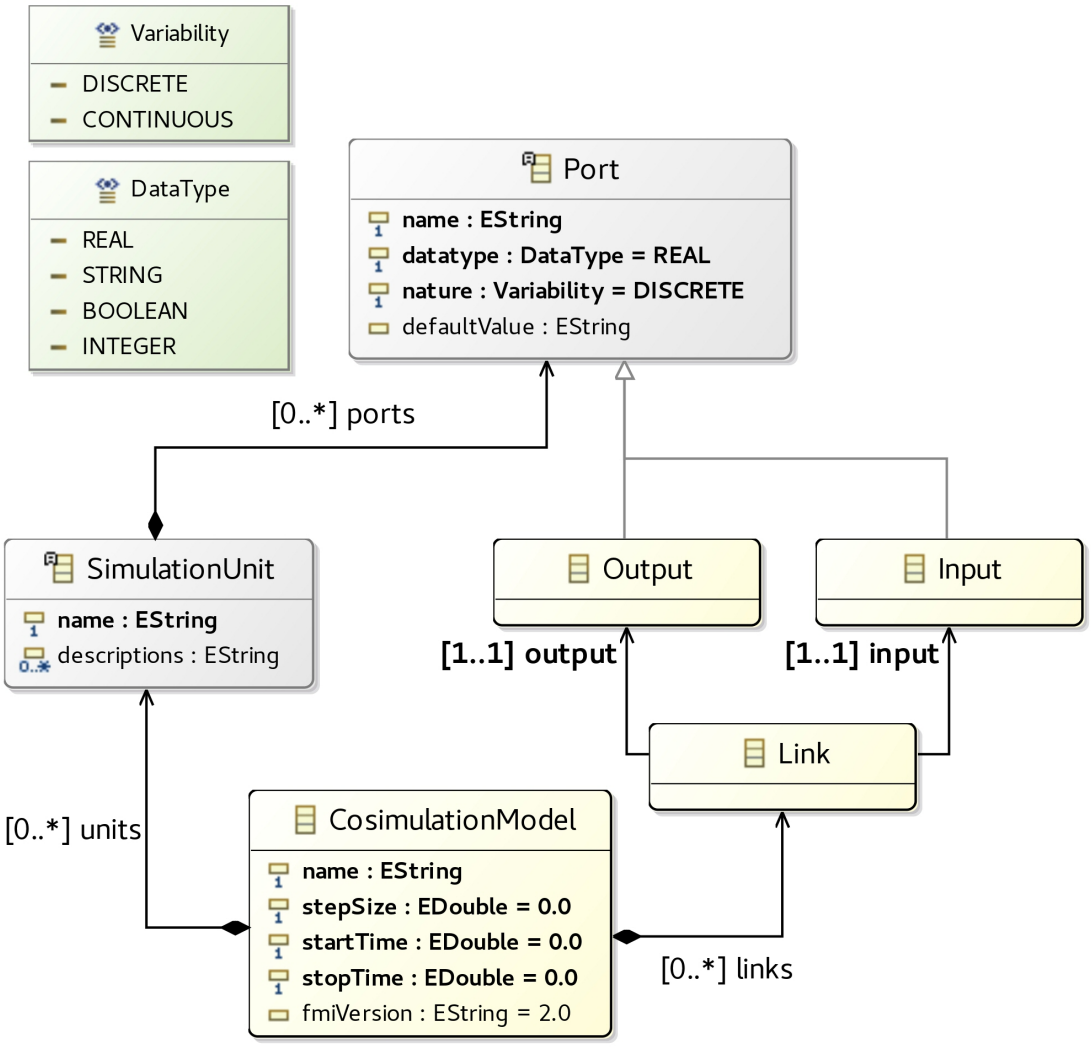

FIGURE 6.5 – Métamodèle simplifié de CosiML

de la simulation. L'élément contient les unités de simulation (`units`) et leurs interconnexions (`links`). Il s'agit des éléments nécessaires pour établir un *scénario de cosimulation* (définition 17).

SimulationUnit : Cet élément représente une unité de simulation impliquée dans la cosimulation. Ses attributs sont le nom de l'unité (`name`), unique, et une description textuelle (`description`). Son interface structurelle est définie par sa liste de ports (`ports`).

Port : Cet élément représente un port d'une unité de simulation. Il possède un nom unique (`name`) ; un type de donnée (`datatype`) qui correspond à l'un des quatre type autorisés (*Real, Integer, Boolean, String*, voir énumération `DataType`) ; une nature (`nature`) pouvant être soit discrète soit continue (voir énumération `Variability`) ; et une valeur par défaut optionnelle (`defaultvalue`).

Input : Élément dérivé de `Port`, représentant un port d'entrée.

Output : Élément dérivé de `Port`, représentant un port de sortie.

Link : Représente une liaison d'échange de donnée entre un port de sortie (`input`) et un port d'entrée (`output`). Il peut y avoir autant de ports d'entrée que voulu reliés à un port de sortie

par des liaisons, mais pas plus d'un port de sortie ne peut être relié à un port d'entrée.

Datatype & Variability : Énumérations utilisées pour définir un ensemble de valeurs que peuvent prendre certains attributs. Elles sont utilisées ici respectivement pour les attributs `datatype` et `nature` de l'élément `Port`.

Les éléments grisés (avec un nom en italique) sont des éléments *abstraits*, c'est-à-dire qu'ils ne peuvent pas être instanciés dans un modèle CosiML. L'élément `SimulationUnit` ne peut donc être instancié que par l'un de ses éléments dérivés, présentés dans la figure 6.6. Pour permettre l'exécution du modèle, un élément `SimulationUnit` doit faire référence à l'emplacement du modèle à simuler dans la cosimulation.

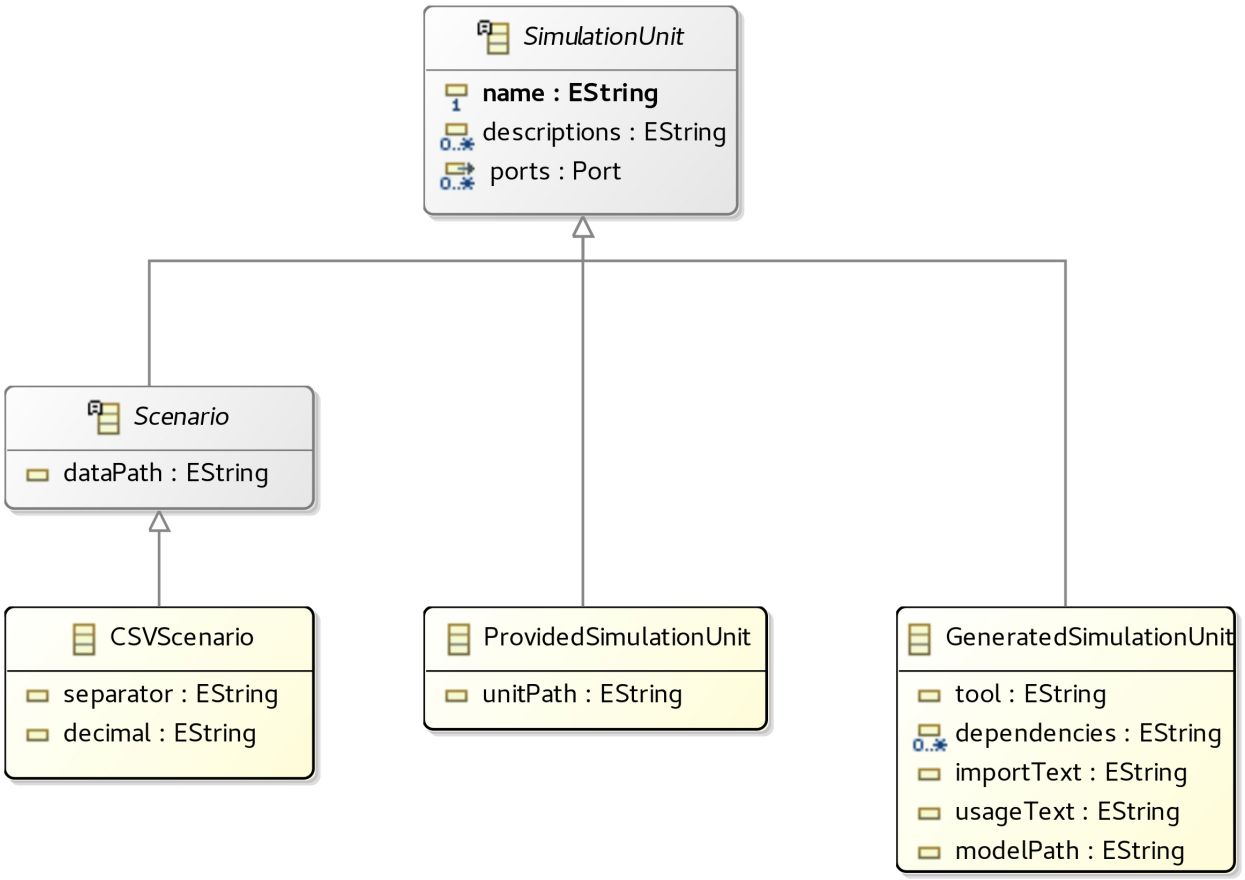

FIGURE 6.6 – Détail des spécialisations de l'élément SimulationUnit

Chacun de ces types dérivés permet de faire référence à un type particulier de modèle, tel que décrit ci-dessous :

ProvidedSimulationUnit : Un type particulier de `SimulationUnit`, qui fait référence à un modèle de simulation déjà disponible au format requis pour la cosimulation, via son attribut `unitPath`. Ce modèle de simulation ne nécessitera donc aucune adaptation particulière pour l'exécution de la cosimulation. Dans notre cas, ce format est le format FMU.

GeneratedSimulationUnit : Un type particulier de `SimulationUnit`, faisant référence à un modèle de simulation non disponible au format requis pour la cosimulation, disponible à l'empla-

cement `modelPath`. C'est le rôle des différents *générateurs d'adaptateur* de CosiML de définir comment ce modèle de simulation doit être adapté au bon format, en s'aidant de l'attribut `tool` permettant d'expliciter l'outil qui a été utilisé pour construire le modèle (comme un compilateur Java ou C++, ou un outil de modélisation comme OMNeT++). Notre but est de rester suffisamment générique pour éviter des modifications du métamodèle lorsque l'on veut supporter un nouvel outil à adapter. Nous proposons 3 attributs supplémentaires pour définir comment le modèle de simulation peut être lié à son adaptateur généré :

`importText` **:** spécifie la façon d'importer le modèle de simulation dans son langage de modélisation. Par exemple, un modèle écrit en Java devra être importé avec :

```
importText = import package.Classname;
```

tandis qu'un modèle écrit en C++ :

```
importText = #include "filename.h"
```

`usageText` **:** spécifie la façon d'appeler l'objet permettant d'interagir avec le modèle de simulation dans l'adaptateur généré (l'objet faisant office de "point d'entrée" du modèle de simulation). Par exemple, pour un modèle en Java ou en C++ :

```
usageText = Classname,
```

dependencies : spécifie une liste des fichiers et ressources à "embarquer" dans l'unité de simulation, dont pourrait dépendre le modèle de simulation.

Les valeurs que peuvent prendre ces attributs dépendent grandement de l'outil utilisé, et des conventions choisies lors du développement du *générateur d'adaptateur* associé.

DataSourceUnit : Un type particulier de `SimulationUnit`, qui ne fait que "produire" de la donnée. Généralement utilisé en tant que source indépendante de données, il fait référence à un fichier de données temporelles via son attribut `dataPath`. Le format des données attendu dépend de l'élément dérivé que nous choisissons d'instancier. Pour le moment uniquement le CSV est supporté (via l'élément `CSVSourceUnit`), mais nous considérons que de futures versions du langage (et de la sémantique associée) pourraient supporter plus de formats. Nous appelons également *générateur d'adaptateur* un générateur créant les fichiers nécessaires pour transformer ce fichier de données en FMU. Nous avons choisi de créer un élément distinct afin de mettre en valeur la distinction entre les unités de simulation constituées à partir de modèles de simulation du système (représentées par des `ProvidedSimulationUnit`s et des `GeneratedSimulationUnit`s), et les unités constituées à partir de fichiers de données statiques (représentées par des `DataSourceUnit`), utilisées généralement pour paramétrer les autres à partir de données d'environnement, ou exprimer l'apparition d'évènements externes indépendants du système. Néanmoins, pour des besoins de débuggage ou de résultats rapides, cet élément donne également un moyen simple d'intégrer des comportements indépendants dont la trace comportementale est déjà connue.

CSVSourceUnit : Un type particulier de `DataSourceUnit`, faisant référence à un fichier de données au format CSV. Ses attributs `separator` et `decimal` permettent de définir les caractères utilisés en tant que séparateur de donnée et marqueur décimal.

Un certain nombre de contraintes doivent être vérifiées pour assurer la validité d'un modèle, représentées par des contraintes OCL associées au métamodèle Ecore :

— Chaque `SimulationUnit` a un nom unique et *normalisé*[3] dans un même `CosimulationModel`.
— Chaque `Output` a un nom unique et *normalisé* dans une même `SimulationUnit`. De même pour chaque `Input`.
— Chaque `Output` ne peut être relié au maximum qu'à un `Input` via un `Link`. Un `Input` peut être relié à autant d'`Output`s que voulu.
— Chaque `Link` doit relier un `Input` et un `Output` ayant le même `datatype`, et appartenant à deux `SimulationUnit`s différentes.
— Aucun `DataSourceUnit` ne peut contenir d'élément `Input` (voir description de `DataSourceUnit`).
— La date de début `startTime` est inférieure à la date de fin de la simulation `stopTime`.

Nous laissons la possibilité au niveau du métamodèle de connecter une sortie discrète à une entrée continue, et vice-versa. En effet, c'est un cas qui se présente fréquemment dans l'étude de systèmes cyber-complexes. Nous choisissons de laisser le rôle d'interprétation de ce genre de connexions aux outils d'exécution du modèle, comme les générateurs de texte présentés dans la section suivante.

6.3 Génération des artefacts de cosimulation FMI

6.3.1 Processus

Nous avons développé CosiML afin de pouvoir représenter un scénario de cosimulation, mais nous souhaitons l'exploiter avec le standard FMI de cosimulation. À cette fin, nous développons des outils permettant d'exploiter CosiML dans ce contexte, en automatisant la génération des artefacts de simulation compatible avec FMI.

La Figure 6.7 illustre le processus de génération de ces artefacts de simulation.

À l'issue de l'étape de génération, un dossier contenant tous les fichiers générés, et le script principal de lancement de la simulation, est obtenu. Ces différents fichiers sont créés par les générateurs présentés au début du chapitre :

1. Le scénario et les paramètres de la cosimulation sont sauvegardés dans un fichier **configuration de cosimulation**. Les chemins vers les différentes FMU à cosimuler sont notamment référencées dans ce fichier.
2. Pour chaque modèle de simulation à adapter, c'est-à-dire pour chaque élément `GeneratedSimulationUnit` ou `DataSourceUnit`, un *générateur d'adaptateur* doit créer les éléments nécessaires à la transformation du modèle en FMU compatible avec les scénarios de cosimulation. Un **adaptateur** (qui peut être sous la forme d'un ou plusieurs fichiers, selon l'outil de modélisation utilisé) peut être généré, souvent dans le langage utilisé par le modèle (sauf dans le cas

3. normalisé au sens du type XML `normalizedString` : pas de tabulations, de retours à la ligne. Contrainte imposée par le standard FMI

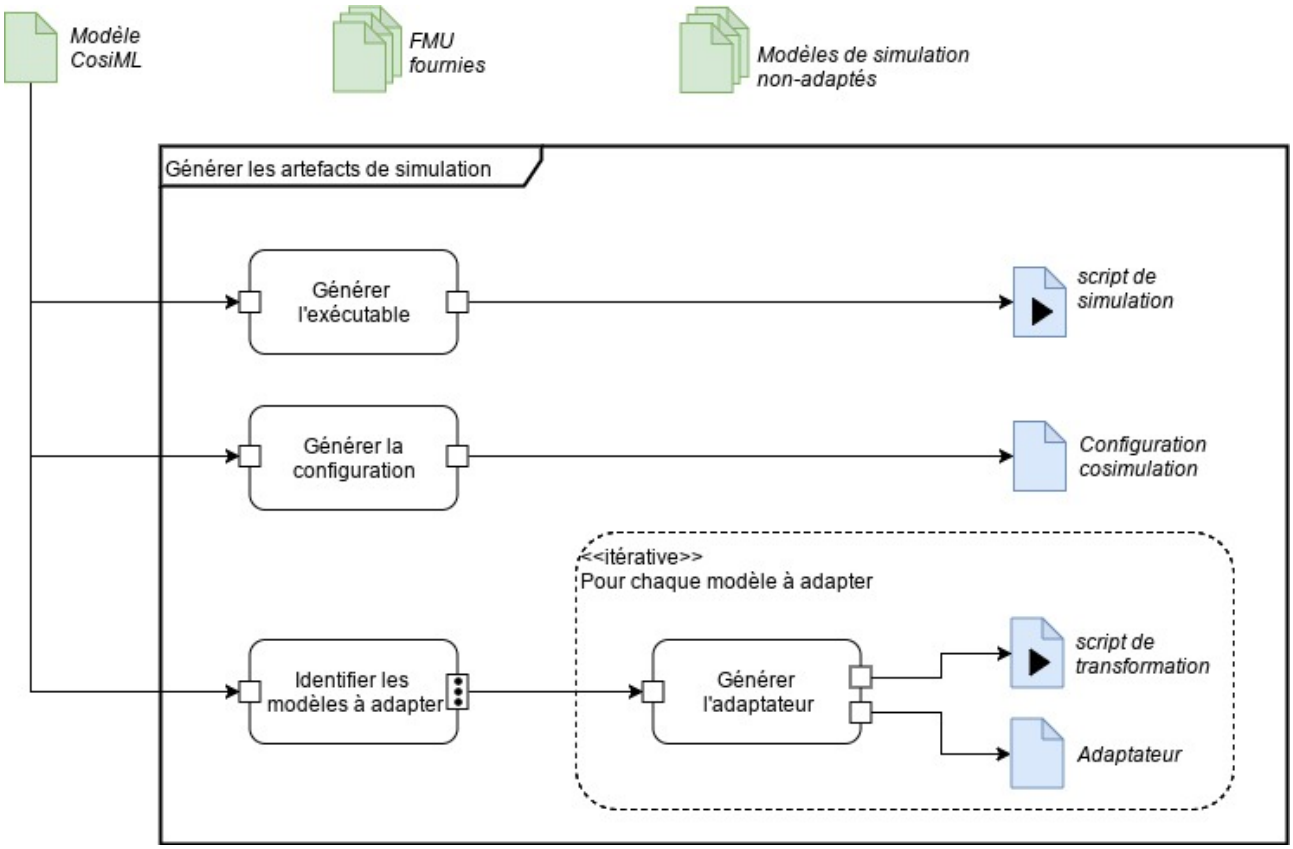

FIGURE 6.7 – Processus de génération des artefacts de simulation

particuler d'un `DataSourceUnit`), afin de faire le lien entre l'interface du modèle et l'interface FMI. Un **script de transformation** est également généré, permettant de réaliser les étapes de transformation (compilation, compression, etc.) en FMU.

3. Enfin, Le **script de simulation** généré permet d'obtenir les résultats de la compilation. La figure 6.8 explicite le processus d'exécution de ce script. Dans un premier temps, il identifie les scripts de transformation et les exécute afin de créer les FMU manquantes, puis charge le fichier de *configuration de cosimulation* dans le *master* de cosimulation et l'exécute pour obtenir les résultats.

6.3.2 Générateurs d'adaptateur

Nous faisons le choix de la version 2.0 du standard FMI comme technologie de cosimulation, nos modèles de simulation doivent donc être fournis à la plateforme de cosimulation au Format FMU (et plus précisément FMU-CS embarquant le modèle et son moteur de simulation). La section 4.3 référence, dans les différents domaines du Smart Grid, un certain nombre d'outils de simulation pouvant être utilisés avec le standard FMI, auxquels il faut ajouter le logiciel OMNeT++ avec notre extension fmi4omnetpp. Afin d'être opérationnels sur des cas d'études de simulation de Smart Grid, nous choisissons d'assurer une compatibilité accrue avec les outils suivants au minimum :

— Dymola ou OpenModelica et leur langage Modelica , pour la simulation du réseau électrique ;

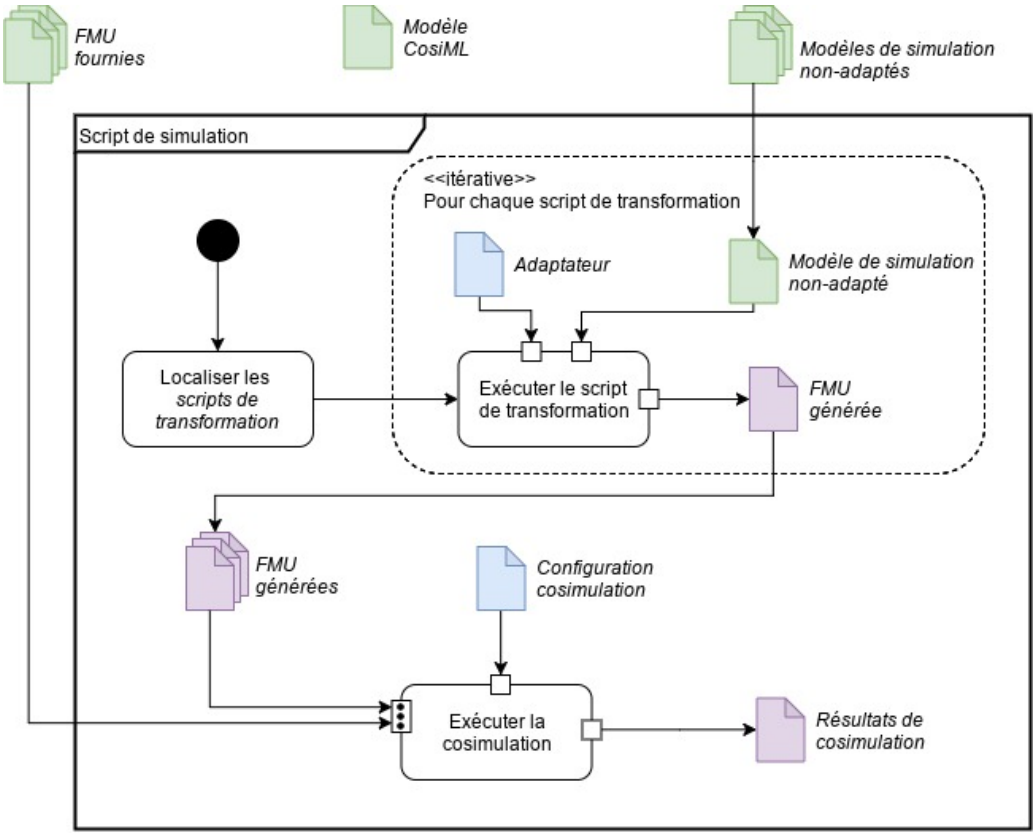

FIGURE 6.8 – Processus du *script de simulation* permettant d'obtenir les résultats de simulation

— Java avec les outils de JavaFMI, pour modéliser des processus de traitement de l'information, ainsi que des algorithmes plus complexes de manipulation de données ;
— OMNeT++ et fmi4omnetpp, pour la modélisation des transmissions sur un réseau de télécommunications ;

Chacun de ces outils implémente un processus d'exportation d'un modèle au format FMU. Mais cela demande parfois une phase d'adaptation, entraînant des modifications importantes au modèle afin qu'il soit compatible avec le processus d'exportation. L'automatisation de cette tâche permettrait d'augmenter la simplicité d'utilisation, de diminuer le coût de formation nécessaire et le risque de bugs et d'erreurs liés à ce processus d'export en FMU.

Le développement d'un *générateur d'adaptateur* est particulièrement indiqué dans les cas où :
— les modifications demandées reposent plus sur une connaissance de l'outil d'exportation et du fonctionnement de la norme FMI plutôt que sur les connaissances métiers nécessaires à l'élaboration du modèle de simulation ;
— elles sont systématiques et demandent souvent les mêmes actions pour chaque modèle d'un même langage ou formalisme ;

Nous chercherons également à créer un *générateur d'adaptateur* chaque fois que nous voudrons adapter automatiquement un nouveau format de fichier de données datées en FMU (rôle de l'élément `DataSourceUnit` de CosiML).

Le langage Modelica et le standard FMI sont développés de manière à ce que la transformation d'un modèle au format FMU soit la plus simple possible pour l'équipe de modélisation. La définition des entrées et sorties d'un modèle Modelica utilise la même expressivité que la définition de l'interface d'une FMU. Ainsi, nous ne considérons pas que ce soit le meilleur candidat au développement d'un générateur d'adaptateur.

En revanche, dans le cas de Java et OMNeT++ , le processus d'exportation en FMU n'est généralement pas une compétence maîtrisée par les utilisateurs. Nous avons choisi de développer :

— un générateur d'adaptateur pour OMNeT++ , utilisant notre extension du logiciel, développée pour permettre l'export FMU,
— un générateur d'adaptateur pour Java,
— un générateur d'adaptateur pour un fichier de données CSV.

De plus, chaque générateur peut être configuré avec un fichier de propriétés faisant le lien entre un modèle CosiML et la plateforme de déploiement de la simulation. Ce fichier de propriété prend la forme d'un fichier d'extension .properties [4] placé à côté du modèle CosiML. Les générateurs sont implémentés en utilisant le plugin Acceleo de l'environnement Eclipse EMF.

6.3.2.1 Générateur d'adaptateur pour OMNeT++

Ce générateur effectue une transformation de modèle depuis un élément `GeneratedSimulationUnit` d'un modèle CosiML, vers du texte. L'attribut `tool` de l'élément doit faire référence à l'outil "OMNeT++ ", utilisé pour développer le modèle de simulation.

Propriétés Le générateur requiert certaines propriétés, principalement sur l'emplacement des bibliothèques requises à la compilation du modèle OMNeT++ :

1. Le nom utilisé dans le modèle CosiML pour faire référence à l'outil compilateur Java.
   default : `gen-omnet.toolname = omnetpp`
2. L'emplacement de l'extension fmi4omnetpp permettant de transformer un modèle en FMU.
   default : `gen-omnet.fmi4omnetpp = fmi4omnetpp/`
3. Le chemin d'installation de OMNeT++ , s'il n'est pas présent dans le `PATH` du système d'exécution de la cosimulation.
   default : `gen-omnet.omnetpp = omnetpp/`
4. L'emplacement de la librairie standard C++ Boost, requise par l'extension fmi4omnetpp, si celle-ci ne se trouve pas dans les emplacements standards du système.
   default : `gen-omnet.boost-lib = boost/`

4. .properties est une extension de fichier principalement utilisée en java pour stocker les paramètres de configuration d'un logiciel. Nous l'utilisons car nous sommes dans un environnement Eclipse EMF basé sur le langage Java

6.3.2.2 Générateur d'adaptateur pour Java

Ce générateur effectue une transformation de modèle depuis un élément `GeneratedSimulationUnit` d'un modèle CosiML, vers du texte. L'attribut `tool` de l'élément doit faire référence à l'outil "Java", utilisé pour développer le modèle de simulation.

Propriétés La transformation requiert certaines propriétés, qui peuvent être modifiées dans le fichier .properties accolé au modèle CosiML :

1. le nom utilisé dans le modèle CosiML pour faire référence à l'outil compilateur Java.
   default : `gen-java.toolname = java`
2. l'emplacement de l'outil fmu-builder.jar de JavaFMI, permettant de transformer en FMU un modèle Java implémentant la librairie fmu-framework.
   default : `gen-java.fmu-builder = fmu-builder.jar`
3. l'emplacement de la librairie fmu-framework de JavaFMI
   default : `gen-java.fmu-framework = fmu-framework.jar`
4. les conventions de nommage de l'interface non-adaptée du modèle de simulation, en fonction du nom du port correspondant (la variable `{0}` représente le nom du port du `GeneratedSimulationUnit` considéré) :
   — le nom de la fonction permettant de lire la valeur d'une variable correspondant à un port de sortie continu.
   default : `gen-java.get-continuous-funcName = get_{0}`
   — le nom de la fonction permettant de détecter un évènement correspondant à un port de sortie discret.
   default : `gen-java.get-discrete-funcName = getEvent_{0}`
   — le nom de la fonction permettant de donner la valeur d'une variable correspondant à un port d'entrée continu
   default : `gen-java.set-continuous-funcName = set_{0}`
   — le nom de la fonction permettant de créer un évènement correspondant à la valeur d'un port d'entrée discret
   default : `gen-java.set-discrete-funcName = setEvent_{0}`

Nous retenons la solution de JavaFMI pour effectuer la transformation d'un modèle Java en FMU. Néanmoins, nous considérons que le créateur du modèle n'a pas forcément de connaissances du fonctionnement de cette solution, et qu'il ne l'utilise pas dans son modèle. Le rôle du générateur est donc de créer un fichier Java implémentant le fmu-framework de JavaFMI, et faisant l'adaptation de l'interface du modèle en une interface compatible avec la transformation en FMU. Cette transformation devant être réalisée par l'outil fmu-builder, un *script de transformation* sera créé pour automatiser l'appel à cet outil avec les paramètres du modèle.

De plus, comme rappelé précédemment nous préférons favoriser la généricité du métamodèle de

CosiML plutôt que de le surcharger avec des règles spécifiques aux outils supportés. Ainsi, nous choisissons d'utiliser une convention de nommage pour l'interface du modèle fourni, convention qui peut être adaptée en modifiant la valeur des propriétés

`gen-java.get-continuous-funcName`
`gen-java.get-discrete-funcName`
`gen-java.set-continuous-funcName`
`gen-java.set-discrete-funcName`

du fichier de propriétés.

Règles de transformation La transformation depuis un élément `GeneratedSimulationUnit` vers un *adaptateur* est définie en Acceleo. Nous listons ci-dessous les éléments remarquables de cette transformation :

1. Un `GeneratedSimulationUnit` entraîne la création d'une classe Java nommée à partir de son attribut `name`.
2. Le modèle de simulation fourni est importé dans cette classe avec `importText`, et nous accédons à ses fonctions d'interface non-adaptée avec `usageText`.
3. Un `Port (P1)` de nature `CONTINUOUS` entraîne la création d'un port FMI nommé `(P1.name)`.
4. Un `Port (P2)` de nature `DISCRETE` entraîne la création de deux ports FMI `(P2.name)` et `(P2.name.sync)` en accord avec l'utilisation des composants d'encodage et décodage de signaux discrets via FMI (voir section 5.3).

Nous utilisons également le plugin Acceleo pour effectuer la transformation d'un élément `GeneratedSimulationUnit` vers un *script de transformation.*

6.3.2.3 Générateur d'apatateur CSV

Ce générateur effectue une transformation de modèle depuis un élément `CSVSourceUnit` d'un modèle CosiML, vers du texte.

Nous choisissons de générer un code Java, que nous savons déjà transformer en FMU grâce aux outils de JavaFMI. Ce code généré importe le fmu-framework de JavaFMI, charge en mémoire les données du fichier CSV, et met à jour les ports de sortie de la FMU au fur et à mesure de la simulation avec les valeurs contenues dans ce fichier.

La transformation requiert certaines propriétés, qui peuvent être modifiée dans le fichier .properties accolé au modèle CosiML :

1. l'emplacement de l'outil fmu-builder.jar de JavaFMI, permettant de transformer en FMU un modèle Java implémentant la librairie fmu-framework.
   default : `gen-csv.fmu-builder = fmu-builder.jar`

2. l'emplacement de la librairie fmu-framework de JavaFMI
   default : `gen-csv.fmu-framework = fmu-framework.jar`

Ici encore, nous utilisons une simple convention de nommage pour assurer une bonne correspondance entre les données et les ports de la FMU :

1. La première ligne du fichier CSV donne un nom à chaque colonne de données. Toutes les colonnes dont le nom correspond au nom d'un port de la FMU à créer seront utilisées comme valeur pour ce même port.
2. La première colonne liste les dates auxquelles les valeurs de la ligne doivent être appliquées aux ports correspondants. Son nom n'importe pas.

C'est la nature du port indiqué dans le modèle CosiML qui déterminera si chaque ligne du fichier CSV correspond à un évènement d'un signal discret, ou au point d'un signal continu. Dans le cas d'une nature discrète, la fonction d'encodage associée sera ajoutée au code généré.

6.3.3 Choix de DACCOSIM en tant que *master* de cosimulation FMI

Le choix du *master* de cosimulation FMI définit le format du fichier *configuration de la cosimulation* et les fonctionnalités disponibles.

DACCOSIM est un logiciel uniquement dédié à être un *master* de cosimulation FMI, contrairement à de nombreux autres qui proposent un *master* en tant que fonctionnalité supplémentaire de leur moteur de simulation. Ce faisant, de nombreuses fonctionnalités sont offertes par DACCOSIM [Évora Gómez et al., 2019] – par rapport à un *master* basique :

— une stratégie de pas de temps intelligent, utilisant la fonctionnalité optionnelle de *rollback*, pour approcher un point de communication au plus proche de la date d'un point de discontinuité. Cette fonctionnalité permet d'améliorer la précision de simulation de FMUs échangeant des signaux encodés, pour lesquels chaque discontinuité correspond à un évènement du signal discret d'origine ;
— un DSL permettant de définir un scénario et les paramètres d'une cosimulation de manière textuelle plutôt que graphique. Cela nous permet d'automatiser le paramétrage de l'outil en générant automatiquement le modèle conforme à ce DSL ;
— une fonctionnalité d'exécution distribuée de la cosimulation, optimisant le temps de simulation. Cela rend l'outil d'autant plus intéressant pour des cas d'étude industriels à grand échelle – comme les Smart Grids – en permettant d'utiliser un large nombre de FMUs ;

Nous implémentons donc un générateur de configuration et un générateur de l'exécutable compatibles avec l'utilisation de ce logiciel.

6.3.3.1 Générateur de configuration

Nous générons un fichier textuel dans le DSL développé par l'équipe de DACCOSIM. Les notions manipulées par le modèle CosiML et le DSL sont très proches, la principale difficulté est dans le traite-

ment de la nature discrète d'un échange de données. Nous considérons que les modèles de simulation, et donc les unités de simulation FMU, utilisent notre stratégie d'encodage et de décodage des signaux discrets. C'est-à-dire qu'un `Port` discret correspondra à deux ports FMI dans le scénario de cosimulation FMI de DACCOSIM : un port d'information, et un port de synchronisation. De même, un `Link` entre deux `Port`s discrets correspondra à deux liens FMI, entre les deux ports d'information puis entre les deux ports de synchronisation. Pour le traitement des liens hybrides, c'est-à-dire entre deux ports de nature différente, nous choisissons d'autoriser un fonctionnement dégradé ou de l'interdire, selon le cas :

Cas discret vers continu : Dans le cas où un `Output` discret est lié à un `Input` continu, nous choisissons de ne créer qu'un lien entre le port FMI d'information de sortie, et l'unique port FMI d'entrée. Il s'agit d'un mode *dégradé*, dans lequel la valeur du signal discret original est toujours transmise au modèle de destination. En revanche, sans le signal de synchronisation nous perdons la "détection" des évènements consécutifs de même valeur. Ce mode dégradé s'applique notamment aux cas où seuls les changements de valeur du signal sont importants pour le modèle de destination.

Cas continu vers discret : Nous choisissons d'interdire le cas où un `Output` continu est lié à un `Input` discret.

Nous utilisons le plugin Acceleo de Eclipse EMF pour définir les règles de transformation d'un modèle conforme à CosiML vers un fichier de configuration DACCOSIM.

6.3.4 Générateur de l'exécutable de la cosimulation

À ce stade, de nombreux fichiers ont pu être générés par les générateurs précédents. Les unités de simulation sont fournies soit au format FMU, soit avec un script exécutable de transformation vers le format FMU. Pour exécuter la simulation du scénario de cosimulation, chacun de ces scripts de transformation doit être exécuté, puis le logiciel DACCOSIM peut être exécuté avec en paramètre le fichier de configuration correspondant. Ces actions ne demandent pas d'effort particulier, mais peuvent être automatisées afin d'accélérer le processus d'exécution de la cosimulation.

Ce générateur ne demande aucune configuration particulière (pas de propriétés nécessaires dans un fichier .properties). Néanmoins, il fait le lien avec les fichiers générés par l'ensemble des autres générateurs développés ici. Ceux-ci respectent une certaine convention de nommage et d'emplacement de fichiers. Tout nouveau générateur développé pour étendre l'utilisation du langage CosiML doit utiliser cette même convention s'il veut être compatible avec le générateur de l'exécutable de la cosimulation.

6.4 Conclusion du chapitre

Ce chapitre propose plusieurs outils permettant de réaliser la cosimulation d'un Smart Grid en utilisant et respectant le standard FMI. L'une des premières contraintes étant que nous ne disposions pas de logiciel de simulation d'un réseau de télécommunications compatible avec FMI, nous avons développé une extension du logiciel OMNeT++ . Cette extension nous permet d'interconnecter un modèle de télécommunication avec la cosimulation FMI, ce que nous ne pouvions pas faire jusqu'à présent. Les deux autres domaines principaux du Smart Grid, le domaine électrotechnique et le domaine de l'information, disposent déjà d'outils permettant d'effectuer la transformation vers le format FMU.

Nous proposons également de construire un modèle du scénario de cosimulation, permettant d'expliciter les unités de simulation intervenant dans la cosimulation, ainsi que les liens entre elles. Nous avons développé le langage CosiML dans cet objectif, et utilisons des outils de vérification syntaxique pour garantir la faisabilité de son implémentation, en validant notamment que les interfaces des unités de simulation soient cohérentes avec les données devant être échangées entre elles.

Par ailleurs, si nous avons souhaité que CosiML reste indépendant dans son expressivité de la technologie ou du standard utilisé pour la cosimulation, nous l'accompagnons d'outils qui permettent de gagner du temps et des efforts lorsque le standard FMI et certains logiciels sont utilisés. Nous permettons en effet d'automatiser :

— la transformation au format FMU de fichiers de données CSV, de code source en Java, ou de modèles développés avec OMNeT++ ,
— la configuration et l'exécution du logiciel DACCOSIM, qui implémente un *master* de cosimulation FMI.

Les artefacts de simulation sont facilement générés à partir des modèles de simulation et du modèle de la cosimulation, ce qui diminue le risque d'erreurs manuelles et la répétition des tâches dans le cadre d'une stratégie d'amélioration et de modification itérative des modèles.

7

L'architecture fonctionnelle au service de la simulation d'un Smart Grid

Sommaire

7.1 Lier architecture fonctionnelle et simulation du Smart Grid 96

7.1.1 Trois nouveaux DSLs 96

7.1.2 Comportements fonctionnels et comportements de transmission 97

7.2 SGridML : Un DSL pour modéliser l'architecture fonctionnelle de la simulation du Smart Grid 98

7.2.1 Métamodèle de SGridML 98

7.2.2 Syntaxe concrète 100

7.3 Transformation de modèle entre SGridML et CosiML 100

7.3.1 AllocationML et CatalogML : deux DSL pour définir la transformation 100

7.3.2 Génération d'un modèle de cosimulation CosiML 101

7.4 Conclusion du chapitre 103

Nous avons présenté dans le dernier chapitre différents outils permettant de faciliter la simulation d'un Smart Grid à partir d'un ensemble de modèles de simulation hétérogènes. Développer "à la main" un modèle de cosimulation en CosiML demande un travail de préparation important. En effet, cela nécessite d'identifier les modèles qui interviendront dans la simulation, les outils qui seront utilisés pour les développer, et enfin leurs interfaces et les connections entre elles. Notre objectif est de soutenir et faciliter cette réflexion préparatoire à l'aide de modèles productifs pouvant être intégrés dans notre démarche IDM. La première section de ce chapitre présente en quoi modéliser l'architecture fonctionnelle est utile pour la simulation d'un Smart Grid. La seconde section présente le langage SGridML, un DSL que nous avons développé pour permettre la représentation des liens entre les comportements du Smart Grid, en s'inspirant de l'architecture fonctionnelle. Enfin, la dernière section présente les deux DSL créés pour permettre la génération automatique d'un modèle CosiML à partir d'un modèle SGridML.

7.1 Lier architecture fonctionnelle et simulation du Smart Grid

7.1.1 Trois nouveaux DSLs

L'architecture fonctionnelle de système consiste à identifier les *fonctions* d'un système, et la manière dont elles interagissent. Il s'agit d'ailleurs généralement de l'un des premiers modèles développés en conception de système. Une *fonction* d'un système est l'identification d'un comportement ou d'un ensemble de comportements du système, fortement couplés et répondant à un besoin spécifique. Les fonctions sont donc les comportements attendus et voulus du système, ceux à implémenter dans sa réalisation. Un modèle d'architecture fonctionnelle représente à la fois les fonctions du système, leurs interfaces – constituées de points d'entrée et de sortie – et les connections possible entre les points de sortie et ceux d'entrée. Lors du processus de conception, avant de définir les différents composants d'un système, nous établissons un modèle de d'architecture fonctionnelle de ce système. Ensuite, nous pouvons identifier les composants à construire pour implémenter ces fonctions.

Dans notre approche, nous nous inspirons de cette pratique de modélisation fonctionnelle, en l'adaptant au fait que le "système" en cours de réalisation est la plateforme de cosimulation du Smart Grid, et ses "composants" sont les unités de simulation. Ainsi, nous souhaitons identifier dans un *modèle fonctionnel (de la simulation)* les comportements intervenant dans la simulation du système (et les informations échangées entre eux), afin de les attribuer à des unités de simulation. Nous proposons de réunir ces informations au sein de plusieurs modèles, et de nous en servir pour construire le scénario de cosimulation. En suivant les pratiques de l'IDM, nous pouvons générer directement un modèle de ce scénario de cosimulation, conforme à notre langage CosiML présenté au chapitre précédent.

Les avantages de cette génération automatique sont de :
- réduire les risques d'erreurs et d'incohérences entre les conclusions du travail préparatoire et les modèles,
- diminuer le temps de modélisation en réutilisant ce qui a déjà été modélisé,
- diminuer le temps d'itération lors des ajustements et autres corrections décidés suite à l'analyse des résultats de cosimulation.

Nous choisissons de développer pour cet objectif trois langages différents :
- SGridML, le langage permettant de définir les comportements de simulation et les échanges entre eux,
- CatalogML, un langage permettant de référencer et lister un ensemble d'unités de simulation,
- AllocationML, un langage permettant de lier les deux langages précédents, et donc d'associer un comportement de SGridML à l'unité de simulation de CatalogML qui l'implémentera.

Nous faisons remarquer que SGridML est indépendant de la solution de simulation choisie, et peut être utilisé pour différentes approches autres que la cosimulation. Les comportements élémentaires peuvent par exemple correspondre à différents modules d'un unique modèle de simulation, être alloués sur une infrastructure HIL (*Hardware-in-the-loop* ou SIL (*Software-in-the-loop*). En revanche, CatalogML et AllocationML sont construits afin de pouvoir spécifier une transformation complète d'un

modèle SGridML vers un modèle CosiML valide, ce qui fixe le choix de la cosimulation.

7.1.2 Comportements fonctionnels et comportements de transmission

Un **comportement de simulation** représente une opération de calcul d'un simulateur manipulant de la donnée. L'interface de ce comportement représente la donnée manipulée – produite ou consommée – par cette opération.

Les comportements de simulation comprennent ainsi généralement les fonctions du système (celles étudiées par l'analyse fonctionnelle), mais également les comportements "non voulus" : comportements externes influençant le système (évènements, pannes, etc.) ou comportements dégradant les performances tels que les comportements de transmission et autres limites. Néanmoins, certains comportements de transmission ont une particularité par rapports aux autres comportements identifiés : ils ne consomment ou ne produisent pas directement de la donnée utile, ils "conduisent" des données d'un point (depuis un comportement "producteur") à un ou plusieurs autres points (vers un ou des comportements "consommateur"). L'impact de cette transmission peut être important (souvent en termes de retard entre le départ et l'arrivée, parfois sur la valeur de la donnée elle-même), ou peut être négligé. Ce type de comportement se place donc en "interception" d'une ou de plusieurs connections pouvant exister entre des comportements. Son interface dépend directement des données échangées par les connections sur lesquelles il se place.

Ainsi, il est plus naturel et moins redondant de décrire un comportement de transmission par l'identification des connexions qu'elle *réalise* plutôt que par son interface d'entrée et de sortie.

Nous implémentons cette distinction dans notre langage proposé, SGridML, qui permet donc de représenter les comportements simulables. Nous mettons ainsi en évidence deux types de comportements de simulation du système, que nous appelons comme suit :

Comportements fonctionnels Comportements reprenant généralement ceux de l'analyse fonctionnelle d'un système, appellation que nous étendons à tous les comportements produisant ou consommant de la donnée utile au scénario de cosimulation.

Comportement de transmission Comportements transmettant des données en les modifiant éventuellement, mais sans en créer de nouvelles.

7.2 SGridML : Un DSL pour modéliser l'architecture fonctionnelle de la simulation du Smart Grid

7.2.1 Métamodèle de SGridML

Comme pour CosiML, nous disposons d'un métamodèle de SGridML exprimé dans le langage Ecore. Ce langage permet de représenter deux types de comportements : des comportements fonctionnels (de création et modification de données), et des comportements de transmission de données. Cette "donnée" manipulée représente de l'information quelle que soit sa forme : état physique, information numérique, ou fait intangible (par exemple).

La figure 7.1 montre le métamodèle complet de SGridML, dont les éléments sont décrits ci-dessous :

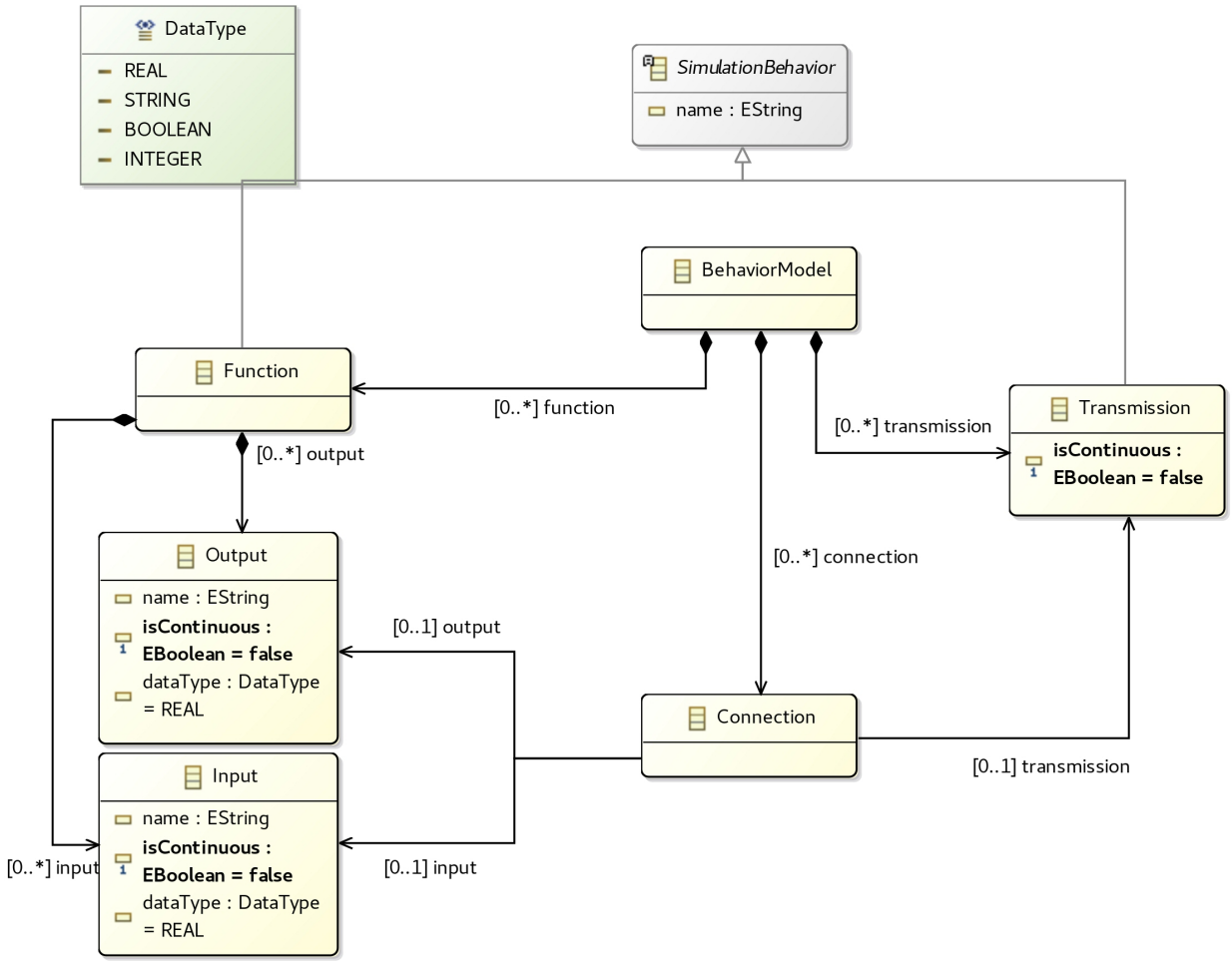

FIGURE 7.1 – Métamodèle Ecore de SGridML

BehaviorModel : Élément racine, utilisé uniquement à des fins de navigation dans le modèle. Il contient une liste des comportements fonctionnels du système (`function`), des connexions existantes entre ces fonctions (`connection`), et des comportements de transmission éventuels de

ces connexions (`transmission`).

SimulationBehavior : Élément représentant une instance d'un comportement élémentaire de simulation, nommé par son attribut `name`.

Function : Élément dérivé de `SimulationBehavior`, représentant un comportement fonctionnel de la simulation du système. Il contient une interface d'entrée (liste `input`), et de sortie (liste `output`). Ces interfaces sont décrites de manière statique, et représentent donc tous les échanges de données possibles, entrant ou sortant de ce comportement au cours de la simulation du système.

Output & Input : Ces éléments représentent un point d'échange de données, part de l'interface d'un comportement fonctionnel. Un `Input` est un point d'entrée d'une donnée, permettant de fournir au comportement qui le contient une donnée nécessaire à sa réalisation. De même, un `Output` est un point de sortie, permettant de partager une donnée produite par le comportement qui le contient, aux autres comportements fonctionnels du modèle. Chacun a pour attribut un nom `name`, le type de la donnée pouvant être échangée `dataType` et un booléen `isContinuous` indiquant si la donnée est de nature continue ou discrète.

Connection : Élément représentant une connexion entre un point de l'interface de sortie d'un comportement fonctionnel du modèle (attribut `output`), et un point de l'interface d'entrée d'un autre (attribut `input`). Cela indique qu'un signal portant l'évolution d'une donnée sera échangé au cours de la simulation du système, entre les deux comportements connectés. Le comportement de transmission de cette connexion peut être spécifié par son attribut optionnel `transmission`. Dans le cas où cet attribut n'est pas spécifié, la transmission est considérée instantanée et parfaite (sans modification de valeur).

Transmission : Élément dérivé de `SimulationBehavior`, représentant un comportement de transmission de la simulation du système. Une donnée transmise peut être altérée, c'est-à-dire retardée, supprimée ou modifiée par ce comportement. Néanmoins, contrairement à un comportement fonctionnel, un comportement de transmission n'a pas d'interface d'entrée ou de sortie explicite. Nous explicitons la nature de ce comportement de transmission avec l'attribut `isContinuous`.

DataType : Énumération représentant les différents types de donnée possibles. Nous pouvons avoir un nombre entier relatif (`INTEGER`), un nombre réel (`REAL`), une chaîne de caractères (`STRING`) ou une valeur booléenne (`BOOLEAN`).

Nous définissons un comportement instancié `SimulationBehavior` comme *élémentaire* par le fait que nous ne souhaitons pas le décomposer en sous-comportements, souvent parce que ceux-ci sont trop fortement couplés et la définition de leurs interconnexions trop complexes.

En connectant ces instances de comportement entre elles, nous obtenons un modèle de tous les échanges de donnée possibles entre les composants qui implémenteront ces instances. Dans notre utilisation de ce langage, un `SimulationBehavior` pourra être alloué à une unité de simulation dans un modèle AllocationML. Une contrainte de cette utilisation est donc qu'un `SimulationBehavior` doit pouvoir être entièrement implémenté dans une unique unité de simulation, sinon cet élément doit être re-décomposé.

7.2.2 Syntaxe concrète

Eclipse EMF fournit un éditeur pour créer ou modifier des modèles au format XMI, conformes à un métamodèle Ecore : le *Sample Reflective Ecore Model Editor*. Nous l'utilisons pour développer des modèles en CosiML, ou développer notre propre *syntaxe concrète*.

7.3 Transformation de modèle entre SGridML et CosiML

7.3.1 AllocationML et CatalogML : deux DSL pour définir la transformation

Le modèle AllocationML nous permet de mettre en cohérence un besoin de simulation représenté dans un modèle SGridML, et l'architecture d'une cosimulation représentée par CosiML. En effet, pour pouvoir réaliser la cosimulation du système modélisé avec SGridML, les unités de simulation doivent respecter une interface minimale, et certaines interconnexions doivent apparaître dans le scénario de cosimulation.

Ces liens de cohérence peuvent être exploités de plusieurs façons, par exemple, en développant un outil de vérification et de détection d'erreurs entre un modèle SGridML et un modèle CosiML. Une deuxième option, qui est celle que nous choisissons de développer, est de générer les éléments nécessaires dans un modèle CosiML pour être cohérent avec un modèle SGridML donné.

Les éléments de CosiML générés seront les `Ports` des `SimulationUnits`, et les `Links` entre eux. Plutôt que de fournir à ce générateur un modèle de CosiML incomplet afin qu'il le remplisse, nous préférons modéliser les éléments non-générables (`CosimulationModel` et les `SimulationUnits`) automatiquement dans des bibliothèques de composants réutilisables, et générer entièrement le modèle de la cosimulation.

CatalogML est un DSL développé dans ce but. Il permet de définir un catalogue d'unités de simulation (`SimulationUnit`) à l'interface (`Port`) non-définie ou incomplète.

Les figures 7.2 et 7.3 montrent respectivement le métamodèle complet de AllocationML et celui de CatalogML.

Les éléments du langage AllocationML sont :

AllocationModel : Élément racine du modèle d'allocation, il contient une liste de *mappings* (`mapping`). Il contient également les paramètres de simulation, qui sont la date de début (`starttime`), de fin (`stoptime`) et la durée du pas de temps entre les points de communication de la cosimulation (`stepsize`).

Mapping : Élément représentant un lien d'allocation entre une instance de comportement (`simulationbehavior`) et une unité de simulation (`simulationunit`). Nous ajoutons un attribut générique appelé `parameters`, afin de permettre à l'utilisateur de paramétrer la façon dont une instance de comportement est reliée à une unité de simulation (par exemple pour détailler com-

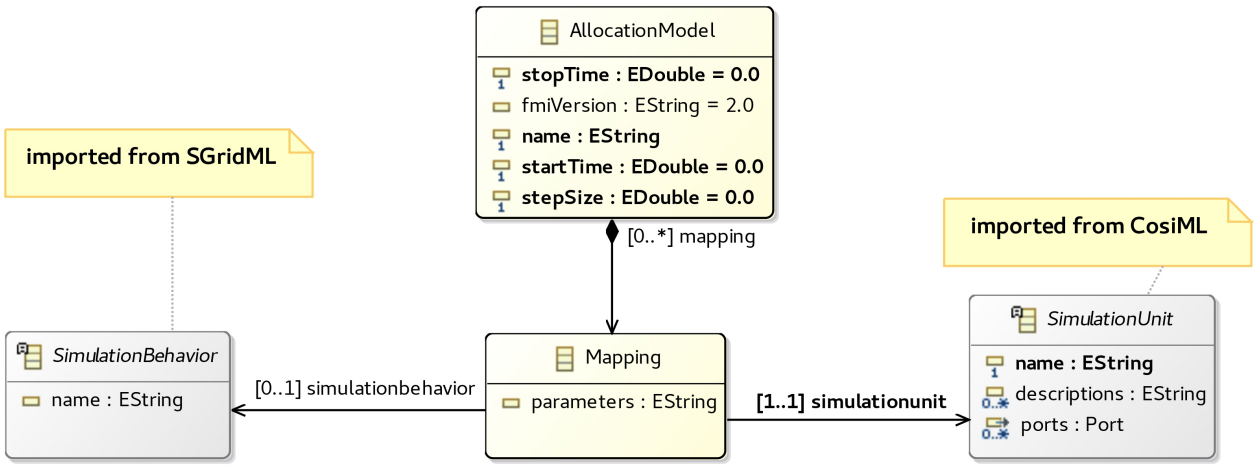

FIGURE 7.2 – Métamodèle de AllocationML

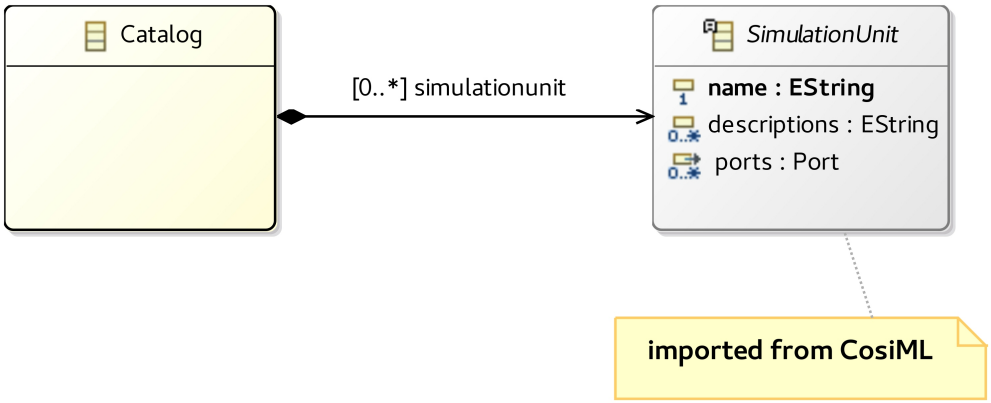

FIGURE 7.3 – Métamodèle de UnitCatalogML

ment lier les données en entrée et sortie du comportement avec le modèle inclus dans l'unité de simulation). C'est à l'utilisateur souhaitant apporter la compatibilité avec un nouvel outil de définir comment ce champ devra être analysé (opération de *parsing*), s'il en a besoin.

Les éléments `SimulationBehavior` et `SimulationUnit` ne font pas partie de AllocationML. Ils sont importés des métamodèles Ecore SGridML et CosiML.

Le langage CatalogML ne contient qu'un seul élement, qui est :

Catalog : Élément racine du modèle, utilisé en tant que conteneur pour une liste d'unités de simulation (`simulationunit`).

7.3.2 Génération d'un modèle de cosimulation CosiML

La génération d'un modèle de cosimulation CosiML requiert un modèle de comportement SGridML, un modèle de catalogue CatalogML et un modèle d'allocation AllocationML. La figure 7.4 illustre les

liens de dépendance entre ces différents modèles.

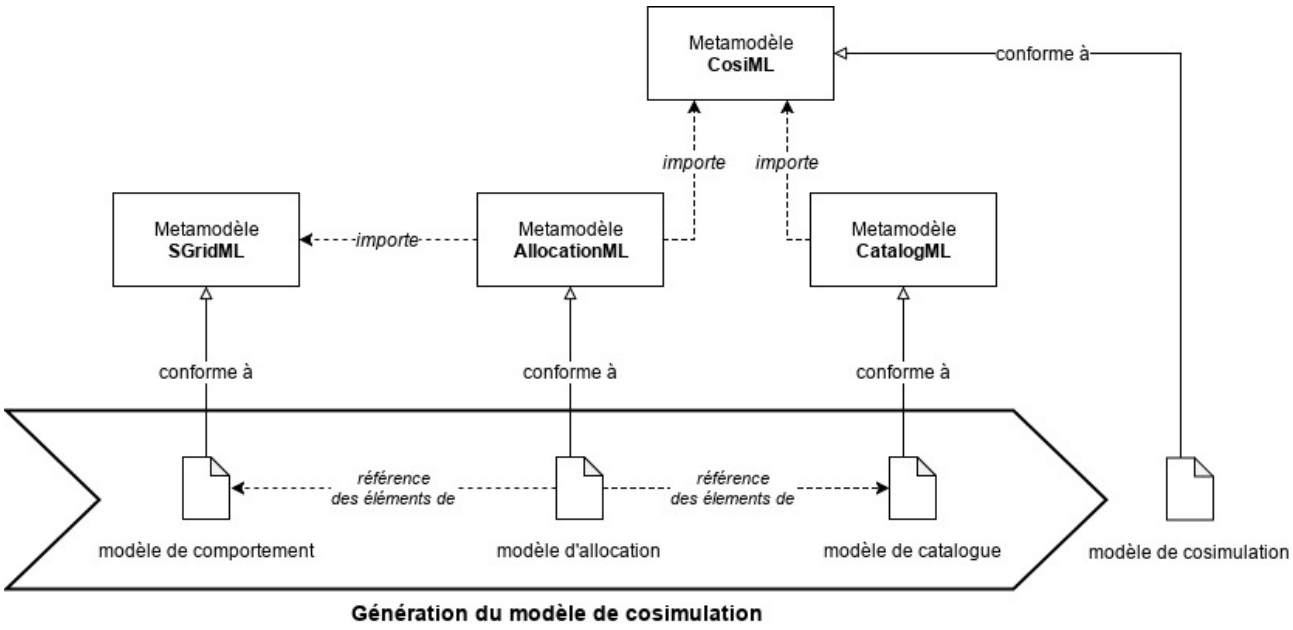

FIGURE 7.4 – Modèles nécessaires à la génération du modèle CosiML, et liens de dépendance

Certaines règles de transformation sont simples : les unités de simulation du modèle de catalogue liées par le modèle d'allocation sont copiées dans le modèle de cosimulation, de même pour la configuration de la cosimulation sélectionnée. Mais les règles de génération de ports et de liens sont plus complexes car elles demandent de nombreux parcours des modèles, et des conditions particulières de génération. Le principe est le suivant. À partir d'un élément `SGridML::Connection` du modèle de comportement :

1. Cette connection est liée à un `SGridML::Input` et un `SGridML::Output`, que nous appelons respectivement In et Out. Pour chacun, nous identifions son élément conteneur `SGridML::Function`, que nous nommons F_{in}, F_{out}. Cette connexion peut être éventuellement liée à un `SGridML::Transmission`, que nous appelons T.
2. Ces `SGridML::SimulationBehaviors` peuvent être alloués sur des unités de simulation. Ainsi, chaque connexion peut concerner entre 0 et trois unités de simulations `CosiML::SimulationUnits`. Nous les appelons A, B et C. Nous considèrons une quatrième pseudo-unité de simulation $\emptyset$ pour signifier qu'un `SGridML::SimulationBehavior` n'a pas été alloué.
3. Selon les possibilités d'allocation de ces trois `SGridML::SimulationBehaviors` sur ces quatre `CosiML::SimulationUnits`, entre zéro et deux `CosiML::Ports` et entre zéro et deux `CosiML::Links` peuvent être générés dans le modèle CosiML.

Dans le cas où T est "alloué" sur $\emptyset$ nous considérons que la connexion a une transmission instantanée. Mais nous décidons d'interdire les cas où F_{in} ou F_{out} ne sont pas alloués (la connexion est ignorée). Pour chaque élément `SGridML::Connection`, le tableau 7.1 montre, pour chaque possibilité d'allocation retenue de F_{in}, F_{out} et T sur A, B, C et $\emptyset$, les éléments à générer dans le modèle de sortie CosiML. Nous utilisons la notation suivante pour les ports et les liens devant être générés :

— $\in A$ signifie que le `SGridML::SimulationBehavior` observé est alloué à A. $\in \emptyset$ signifie qu'il n'est

pas alloué. Nous écrivons $\in \emptyset$ pour T dans les cas où T n'est pas alloué, ou qu'il n'existe pas (la connexion n'est reliée à aucun `SGridML::Transmission`).

— $\overrightarrow{Out}$ et $\overleftarrow{In}$ représentent respectivement un `CosiML::Output` et un `CosiML::Input` transformé à partir de In et Out (respectivement). $\overrightarrow{Out - In}$ représente un `CosiML::Output` transformé à partir du couple (In, Out).

— $A \rightarrow B$ représente un `CosiML::Link` généré entre les ports de sortie générés de A et ceux d'entrée de B.

— Nous ne représentons pas le `CosiML:Port` supplémentaire de synchronisation généré dans le cas d'un signal discret, car ils suivent les même règles de création et de connexion que le `CosiML::Port` d'information.

Éléments d'entrée				**Éléments générés**			
Unités liées à `Connection`	`SimulationBehavior`			`Port`			`Link`
	F_{Out}	F_{In}	T	A	B	C	
A	$\in A$	$\in A$	$\in A$	-	-	-	-
A, $\emptyset$	$\in A$	$\in A$	$\in \emptyset$	-	-	-	-
A, B	$\in A$	$\in B$	$\in B$	$\overrightarrow{Out}$	$\overleftarrow{In}$	-	$A \rightarrow B$
	$\in A$	$\in B$	$\in A$	$\overrightarrow{Out}$	$\overleftarrow{In}$	-	$A \rightarrow B$
	$\in A$	$\in A$	$\in B$	$\overrightarrow{Out}$ $\overleftarrow{In}$	$\overrightarrow{Out - In}$ $\overleftarrow{Out - In}$	-	$A \rightarrow B \rightarrow A$
A, B, $\emptyset$	$\in A$	$\in B$	$\in \emptyset$	$\overrightarrow{Out}$	$\overleftarrow{In}$	-	$A \rightarrow B$
A, B, C	$\in A$	$\in B$	$\in C$	$\overrightarrow{Out}$	$\overleftarrow{In}$	$\overrightarrow{Out - In}$ $\overleftarrow{Out - In}$	$A \rightarrow C \rightarrow B$

TABLE 7.1 – Les différentes possibilités de transformation d'un élément `Connection` vers des éléments `Port` et `Link`

La figure 7.5 montre une extraction d'une composition des modèles d'entrée, correspondant au cas de la dernière ligne du tableau 7.1. La figure 7.6 montre un extrait du modèle de sortie correspondant aux éléments générés dans ce même cas.

Nous implémentons ces règles de transformation dans un script Acceleo.

7.4 Conclusion du chapitre

Nous présentons dans ce chapitre un nouveau DSL, SGridML, créé pour lister les comportements de simulation devant être modélisés pour simuler un Smart Grid. Nous comparons son objectif à celui d'un langage d'architecture fonctionnelle en conception de système, en définissant le système comme étant les artéfacts de simulation (scénario de cosimulation, modèles et unités de simulation). La particularité de ce langage est notamment de distinguer deux types de comportement de simulation : les comportements fonctionnels et les comportements de transmission.

Ce langage est construit pour être réutilisable dans n'importe quelle approche de simulation. Néan-

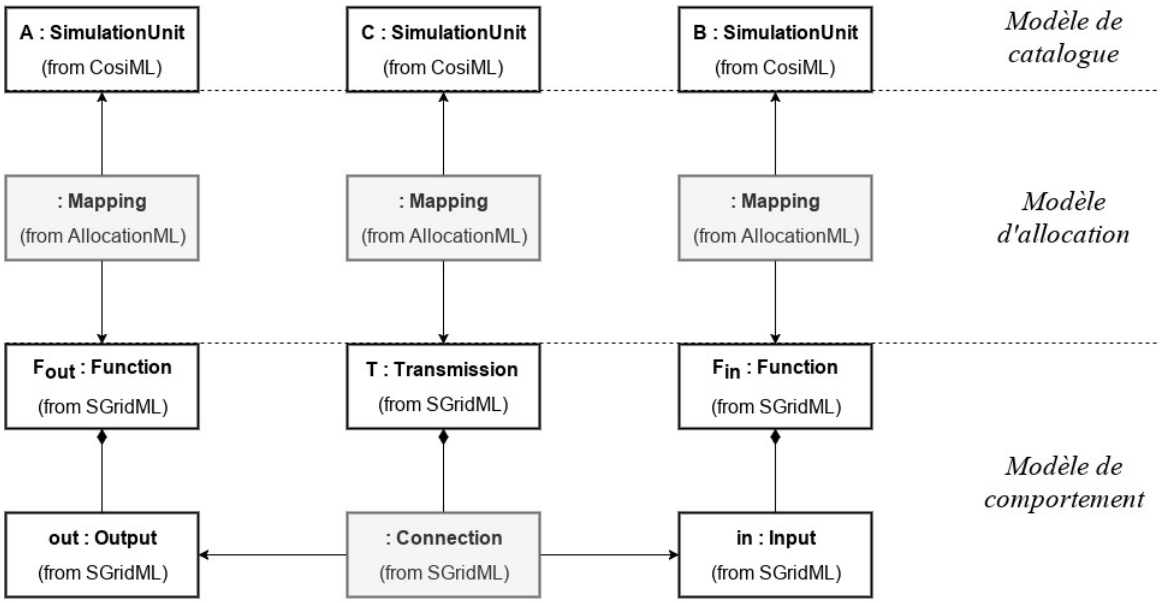

FIGURE 7.5 – Exemple de connexion impliquée dans trois unités de simulation

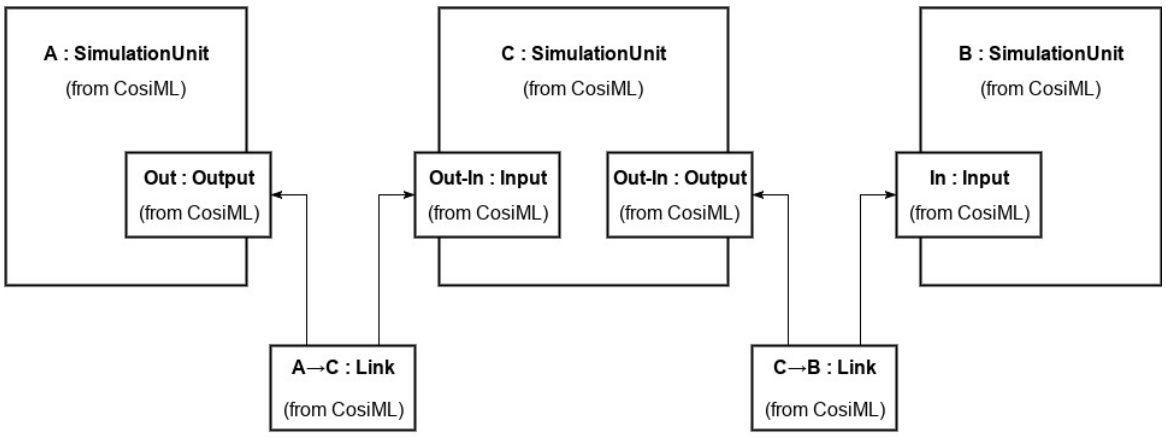

FIGURE 7.6 – Exemple de scénario de cosimulation généré à partir d'une connexion

moins, nous justifions son intérêt dans notre démarche par le développement d'une transformation de modèles, qui permet de générer automatiquement un modèle CosiML. Cette transformation se base sur deux autres langages créés en plus de SGridML : CatalogML et AllocationML. En plus de fournir un support de validation de contraintes de cohérence grâce à des règles sémantiques, l'utilisation de ces langages ainsi que la transformation de modèles apportent un nouveau gain de temps dans l'application de notre démarche de cosimulation et de chacune de ses itérations. Le prochain chapitre formalisera la constitution d'un environnement de simulation de Smart Grid, appelé *Smart Grid Simulation Framework*, composé notamment des outils présentés dans ce chapitre et les précédents, et illustrera son utilisation dans le cadre de notre démarche.

8

Présentation de *Smart Grid Simulation Framework* sur un cas d'étude et observations

Sommaire

8.1 Contexte du cas d'étude 108
8.1.1 Présentation du problème 108
8.1.2 Utiliser le *Smart Grid Simulation Framework* 109
8.2 Déroulement de la démarche 110
8.2.1 Phase 1 : modélisation du système de l'île de Sein 110
8.2.1.1 Modèle de comportement de la simulation 110
8.2.1.2 Modèle de catalogue et d'allocation 112
8.2.1.3 Modèle de cosimulation 113
8.2.1.4 Développement des unités de simulation 115
8.2.2 Phase 2 : Configurer la cosimulation FMI 117
8.2.2.1 Adapter les modèles en FMU 117
8.2.2.2 Implémenter le scénario de cosimulation dans le *master* FMI 118
8.2.3 Phase 3 : Analyser les résultats 119
8.2.3.1 Évaluer les résultats de simulation 119
8.2.3.2 Prendre les décisions d'itération ou valider le modèle de la solution . . 121
8.3 Observations 121
8.4 Conclusion du chapitre 123

Ce chapitre présente l'utilisation du *Smart Grid Simulation Framework*, qui est le nom que nous donnons à l'environnement constitué de l'ensemble de nos outils présentés dans les chapitres 6 et 7 précédents. Notre objectif est de montrer l'application des différentes contributions de nos travaux sur un cas d'étude identifié : celui de l'évolution du réseau électrique de l'Île de Sein. Nous présentons dans la première section la problématique de ce cas d'étude, et rappelons la démarche qui sera utilisée et qui a déjà été présentée au chapitre 5. La deuxième section illustre étape par étape la mise en œuvre de cette démarche, et son utilisation avec le *Smart Grid Simulation Framework*. Puis, la troisième

section généralise les observations qui peuvent être faites sur l'utilisation de notre environnement dans un contexte plus général que le cas d'étude de l'Île de Sein.

8.1 Contexte du cas d'étude

8.1.1 Présentation du problème

Nous choisissons un cas d'étude réel de EDF pour appliquer et tester notre démarche et nos outils de simulation. Le système est le réseau électrique de l'île de Sein, une île Bretonne française. Ce réseau n'est pas relié à la production du continent, il dépend donc uniquement de ses sources de production pour assurer la fourniture électrique de l'île.

Aujourd'hui, le réseau est constitué d'une centrale de production thermique assurant la fourniture principale de l'île. Bien que ce groupe de production soit suffisant pour les besoins de consommation, des équipements de production photovoltaïque (PV) sont installés afin d'augmenter la part de production d'énergie renouvelable (EnR) dans le mix énergétique de l'île. La principale difficulté liée à cette configuration est que nous nous appuyons sur une source d'énergie intermittente et imprévisible (autant que la météo). Pour satisfaire l'équilibre entre la production et la consommation, la source doit parfois être limitée en puissance ou même découplée du réseau, ce qui crée des pertes économiques et une dégradation de l'empreinte carbone.

Pour favoriser l'intégration de cette production intermittente tout en maintenant la qualité de fourniture et la sûreté du système électrique, nous souhaitons mettre en place un système de stockage pilotable, composé d'une batterie et de son interface associée (onduleur, transformateur, etc.). Par la suite, nous nommerons "batterie" l'ensemble de ce système de stockage. De plus, un contrôle intelligent sera implémenté au travers d'un *Energy Management System* (EMS), qui sera couplé au Système de Contrôle et d'Acquisition de Données (SCADA, en anglais) afin de piloter la production et d'optimiser l'utilisation des EnR. Cette étude porte sur l'évaluation de la solution complète de stockage de contrôle intelligent, via ces deux équipements. Cet ensemble permettra notamment :

— d'optimiser l'utilisation de la production PV, et d'éviter de recourir à leur découplage ou à la limitation de sa puissance en cas de faible consommation de l'île ;
— d'éviter de démarrer des groupes supplémentaire de production thermique pour le passage des pointes de consommation ;
— lorsque les conditions le permettront (charge de la batterie, puissance actuelle de consommation et de production PV), une alimentation de l'île sans aucun fonctionnement des groupes diesel.

Nous distinguons alors deux modes de fonctionnement dans le système sous étude :

Mode couplé : Dans le *mode couplé*, la centrale thermique est couplée au réseau électrique, et est la source principale de production. Le système de stockage est piloté en puissance.

Mode ilôté : Dans le *mode ilôté*, la centrale thermique est découplée du réseau électrique. La batterie devient la source principale, et n'est plus pilotée en puissance.

Tous les nœuds du système sont contraints en puissance, que ce soit par des consignes de pilotage ou des contraintes extérieures (niveau d'ensoleillement, besoins en consommation, etc.), excepté la source principale. La source principale, différente pour chacun des deux modes de fonctionnement, doit pouvoir compenser les pertes non-prévisibles du système en émettant ou absorbant de la puissance active et réactive en temps-réel pour éviter l'effondrement du réseau.

Pour simplifier notre exemple, nous restreignons notre étude au pilotage à distance de la production, et ne modélisons pas les fonctions de protection du réseau électrique (régulation de tension ou fréquence, protection de défauts et court-circuits). Nous considérons les fonctions du système suivantes :

— Le réseau électrique tente de s'établir s'il le peut dans un point de fonctionnement en fonction des contraintes appliquées.
— L'EMS surveille l'état du réseau et calcule les consignes de fonctionnement à appliquer aux différents équipements en fonction de leurs contraintes de fonctionnement (charge restante, puissance maximale / minimale acceptée, etc.).
— Le PV produit au maximum de ce qu'il peut produire, ou est limité par une consigne de l'EMS.
— La batterie fonctionne dans le mode choisi par l'EMS et :

1. en *mode couplé* : produit ou consomme la puissance en fonction de la consigne de l'EMS et de sa charge restante ;
2. en *mode ilôté* : produit ou consomme en fonction de la demande du réseau électrique et de sa charge restante.

— La centrale thermique fonctionne avec un groupe de production, qui peut être découplé du réseau sur demande de l'EMS, ou produire selon la demande du réseau électrique et de ses capacités (puissance minimale et maximale).

Notre objectif est de **modéliser l'intégration du stockage et des algorithmes de pilotage intelligent de la production sur le réseau électrique existant**, et d'évaluer l'efficacité et la pertinence des choix de conceptions par la simulation. Impliquant la simulation de modèles hétérogènes, cet exemple est adapté à l'utilisation de notre démarche de cosimulation, présentée au chapitre 5.

8.1.2 Utiliser le *Smart Grid Simulation Framework*

Les langages de modélisation et générateurs présentés dans les chapitres précédents (chapitres 6 et 7) ont été développés pour supporter les étapes de notre démarche de cosimulation (chapitre 5 et figure 5.1). Mis ensemble, ils constituent avec les outils de simulation supportés un environnement complet et extensible permettant de guider et simplifier le travail des acteurs de cette démarche. Nous appelons cet environnement ***Smart Grid Simulation Framework***.

Les domaines techniques étudiés sont ceux classiques du Smart Grid identifiés précédemment, nous avons donc la présence des profils d'expert en modélisation et simulation électrique, télécoms et SI, ainsi qu'un architecte de solution dans chacun de ces domaines. Enfin, un porteur du besoin et un chargé de cosimulation sont également présents.

La présentation du cas d'étude (sous-section précédente) est fournie par le porteur du besoin, ainsi que toutes les spécifications nécessaires à la reconstruction d'un modèle du système existant. La démarche peut donc être mise en œuvre. Le tableau 8.1 synthétise la correspondance entre les étapes de notre démarche et les outils du *Smart Grid Simulation Framework*.

	Étape	**Outil à utiliser**
Phase 1 : Modéliser la solution		
	Définir une solution	Construire un modèle SGridML
	Répartir les responsabilités	Construire un modèle CatalogML et un modèle AllocationML.
	Définir les interfaces et interconnexions	Générer automatiquement le modèle CosiML.
	Produire les modèles de simulation	Construction des modèles de simulation en fonction des modèles déjà réalisés et générés
Phase 2 : Configurer la cosimulation FMI		
	Adapter les modèles en FMU	Mettre à jour le modèle CatalogML avec les informations des modèles de simulation et re-générer le modèle CosiML. Générer les adaptateurs nécessaires à partir de ce modèle CosiML.
	Implémenter le scénario de cosimulation dans le *master FMI*	Générer à partir du modèle de CosiML le script de configuration de DACCOSIM.
Phase 3 : Exécuter et analyser les résultats		
	Évaluer les résultats de cosimulation	Le script d'exécution de la cosimulation est généré automatiquement depuis le modèle CosiML.
	Prendre des décisions d'itération	Cela reste un travail qualitatif et humain, en revanche, l'implémentation des décisions d'itération est facilitée par l'ensemble de nos outils de génération et d'automatisation

TABLE 8.1 – Correspondance entre les étapes de la démarche et les outils de *Smart Grid Simulation Framework*

8.2 Déroulement de la démarche

Cette section présente le déroulement des différentes phases de la démarche présentée au chapitre 5, appliquée au cas d'étude du réseau électrique de l'île de Sein.

8.2.1 Phase 1 : modélisation du système de l'île de Sein

8.2.1.1 Modèle de comportement de la simulation

La première étape est l'identification des comportements de simulation du système électrique (intelligent) de l'île de Sein. Pour rappel, ces comportements correspondent à des opérations de calcul et de manipulation d'information.

Les fonctions du système sont à modéliser, mais également les fonctions de l'environnement qui agissent sur celui-ci. Ainsi, nous identifions les comportements fonctionnels de la simulation suivants :

Fonction 1 : CalculerPointDeFonctionnement — Le système électrique obéit aux lois de la physique et de l'électrotechnique. Nous souhaitons calculer l'état d'équilibre du réseau en fonction des contraintes extérieures, c'est-à-dire l'état physique (tension, intensité, puissance, etc.) de tous les points d'intérêt à visualiser. Cette fonction permet notamment de calculer les pertes de transmission, et la puissance devant être délivrée par la source principale (centrale diesel ou batterie, selon le mode) pour satisfaire les contraintes.

Fonction 2 : CalculerPuissancePV — La puissance fournie par les sources de production photovoltaïques dépend de la luminosité, nous souhaitons calculer comment cette énergie lumineuse est convertie en énergie électrique. Les sous-fonctions de calcul du maximum de production, et d'application des consignes de puissance extérieures sont intégrées afin d'obtenir la puissance réelle injectée sur le réseau électrique.

Fonction 3 : StockerEnergie — Nous avons besoin de modéliser le comportement du système de stockage, afin de calculer le niveau de charge de la batterie, et de calculer la puissance réellement injectée sur le réseau en fonction des consignes de pilotage et des caractéristiques et de l'état de la batterie.

Fonction 4 : CalculerConsignes — Cette fonction se charge de calculer l'ensemble des consignes à appliquer aux équipements du réseau électrique. Il s'agit de consignes de production, en fonction des mesures de la consommation et des possibilités de production. Nous choisissons de ne pas prendre en compte des données de prévisions météo et de consommation.

Fonction 5 & 6 : Illuminer & Consommer — Pour la réalisation des comportements identifiés précédemment, nous avons besoin de données non produites par le système étudié. Il s'agit de données externes, indépendantes, qui sont l'évolution de la consommation et de la luminosité au cours du temps. Nous représentons l'intégration de ces données au comportement du système par deux fonctions, qui ne font que produire des données sans en consommer.

Ces informations sont représentées au sein d'un modèle SGridML, dont le métamodèle est défini à la section 7.2. Pour chacun de ces comportements fonctionnels, représentés par l'élément `Function`, nous identifions dans le tableau 8.2 les interfaces d'entrée et de sortie, modélisés par des éléments `Input`s et `Output`s. Nous spécifions notamment la variabilité des signaux de données, continus ou discrets.

Cette définition des interfaces de chaque comportement fonctionnel nous permet de tracer en parallèle des connexions entre eux. Pour que le modèle soit complet et cohérent, il faut que chaque `Input` instancié soit connecté à un (et un seul) `Output` du modèle. Par ailleurs, certaines de ces connexions représentent des échanges de données dont le temps de transmission peut être non négligeable. En effet, nous considèrons dans notre cas d'étude que les différentes sources de production, ainsi que l'EMS, sont distinctes au niveau géographique, et qu'ils utilisent le réseau de télécommunication pour communiquer entre eux. Ainsi, un comportement de transmission `Nature` nommé `TransmettreTelecom` est instancié dans notre modèle. Sa nature de transmission est discrète (attribut `isContinuous` défini à `false`), et est associée à toutes les connexions entrantes ou sortantes du comportement fonctionnel

`Function`	`Input` et `Output`			
`name`	I/O	`datatype`	`name`	continu ?
CalculerPoint-DeFonctionnement	Output	REAL	diesel_puissanceRequise	Oui
	Output	REAL	batterie_puissanceRequise	Oui
	Input	REAL	batterie_puissance	Oui
	Input	REAL	consommateur_consommation	Oui
	Input	REAL	pv_production	Oui
	Input	BOOLEAN	mode_estCouplé	Oui
CalculerPuissancePV	Output	REAL	production	Oui
	Output	REAL	puissanceMaxDisponible	Oui
	Input	BOOLEAN	estLimité	**Non**
	Input	REAL	puissanceLimite	Oui
	Input	REAL	luminosite	Oui
StockerEnergie	Output	REAL	SOC	Non
	Output	REAL	puissance	Oui
	Input	BOOLEAN	estCouplé	Oui
	Input	REAL	puissanceEms	Oui
	Input	REAL	puissanceReseau	Oui
CalculerConsignes	Output	REAL	pv_puissanceLimite	Non
	Output	BOOLEAN	pv_estLimité	Non
	Output	BOOLEAN	mode_estCouplé	Non
	Output	REAL	batterie_puissanceConsigne	Non
	Input	REAL	batterie_puissance	Non
	Input	REAL	batterie_SOC	Non
	Input	REAL	diesel_puissance	Non
	Input	REAL	pv_puissanceMax	Non
	Input	REAL	pv_puissance	Non
	Input	REAL	consommation_puissance	Non
Illuminer	Output	REAL	luminosite	Oui
Consommer	Output	REAL	puissance	Oui

TABLE 8.2 – `Function`, `Input` et `Output` du modèle SGridML de l'île de Sein

`CalculerConsignes`. Nous considèrons toutes les autres connexions sans comportement de transmission associé.

Cela représente un total de 6 `Function`s, 16 `Connection`s (car 16 `Input`s), et 1 `Nature` à modéliser. La figure 8.1 est une représentation schématique du modèle SGridML ainsi construit, qui est en pratique fourni au format de données XMI.

8.2.1.2 Modèle de catalogue et d'allocation

La seconde étape de la phase de modélisation de la solution est la répartition des tâches entre les différents experts en M&S, chargé de développer les modèles de la cosimulation. Pour cela, le langage CatalogML nous permet de définir une liste d'unités de simulation, et le langage AllocationML d'y

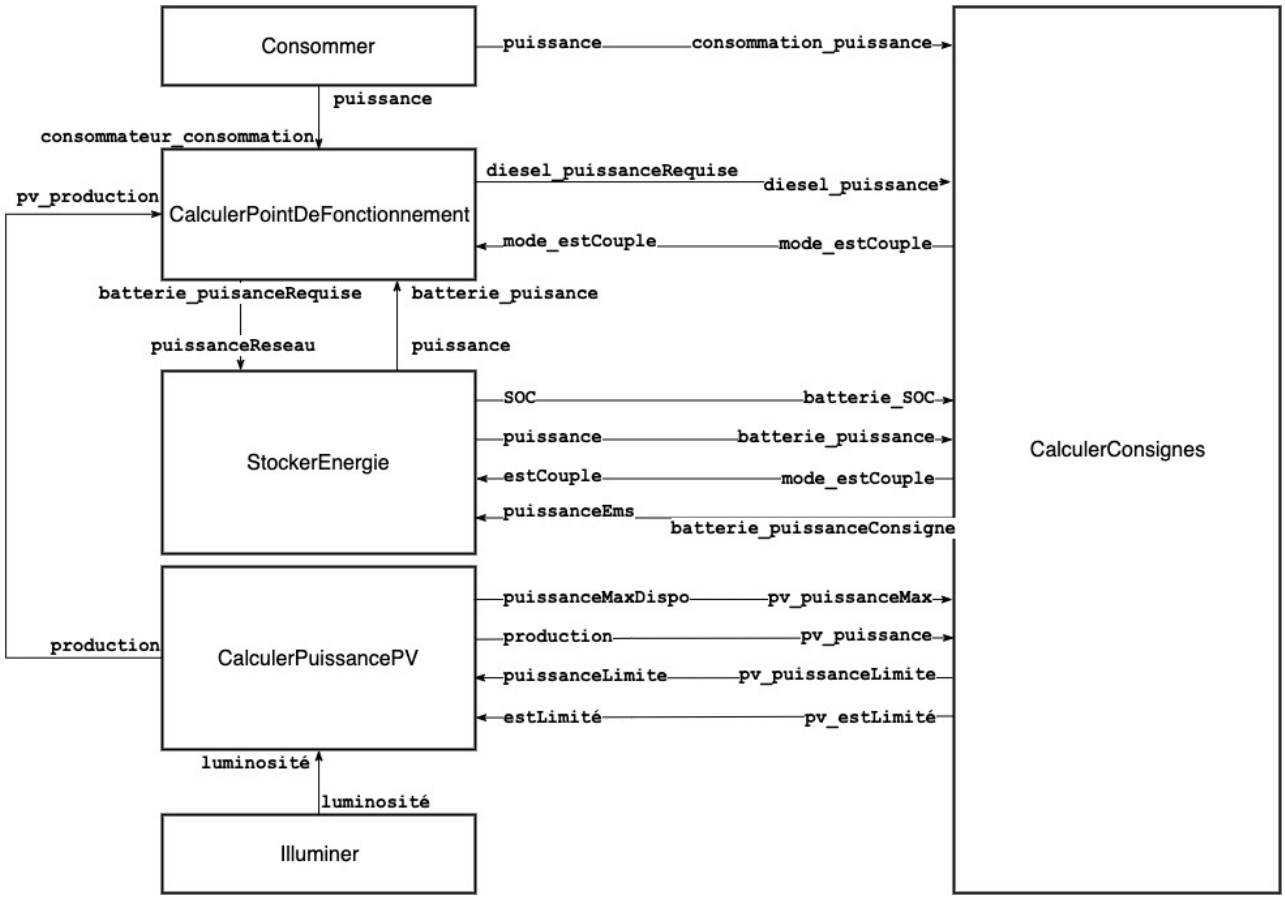

FIGURE 8.1 – `Connection` entre les `Function` du modèle SGridML de l'île de Sein

attribuer les comportements décrits à l'aide de SGridML.

Nous choisissons de modéliser le comportement de transmission `TransmettreTelecom` avec un modèle OMNeT++ . La fonction `CalculerConsignes` représentant les algorithmes complexes de l'EMS sera implémentée dans un modèle en Java. Les données produites par les fonctions `Illuminer` et `Consommer` sont fournies dans un fichier de données au format CSV. Toutes les autres fonctions représentent des comportements principalement basés sur des lois physiques, et adaptés à une modélisation par systèmes d'équations. Le langage Modelica est retenu pour la modélisation, et le modèle pourra directement être fourni au format FMU.

La figure 8.2 liste les unités de simulation instanciées dans le modèle CatalogML. Le tableau 8.3 liste les sept instances de mapping du modèle d'allocation entre les `SGridML::SimulationBehaviors` du modèle de comportement et les `CosiML::SimulationUnits` du modèle de catalogue.

8.2.1.3 Modèle de cosimulation

La troisième étape "Définir les interfaces et interconnexions" consiste à identifier les interfaces de chaque unité de simulation, et les connexions existantes entre les ports de sortie et d'entrée. Ces

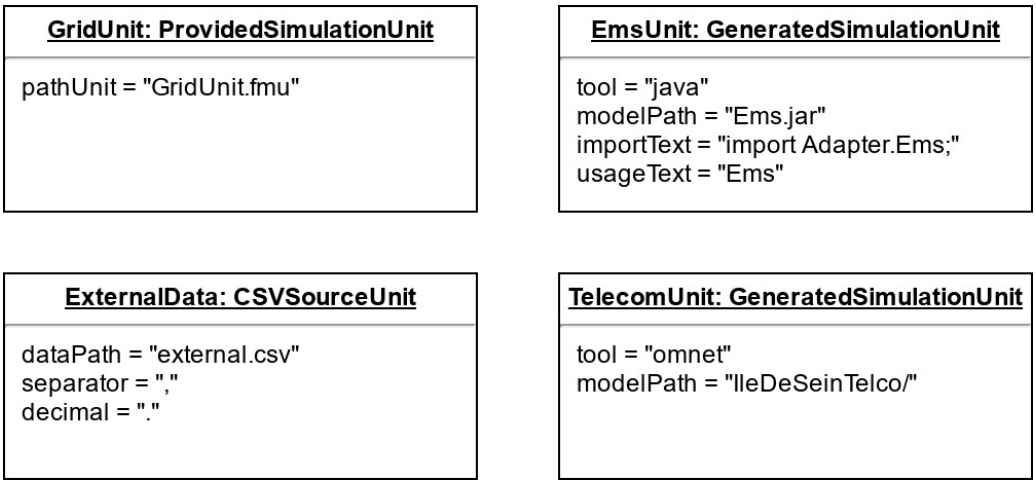

FIGURE 8.2 – `SimuationUnit` instanciées dans le modèle CatalogML

`simulationunit`		`simulationbehavior`	
Type instancié	nom	Type instancié	nom
Function	CalculerPointDeFonctionnement	ProvidedSimulationUnit	GridUnit
Function	CalculerPuissancePV	ProvidedSimulationUnit	GridUnit
Function	StockerEnergie	ProvidedSimulationUnit	GridUnit
Function	CalculerConsignes	GeneratedSimulationUnit	EmsUnit
Function	Illuminer	CSVSourceUnit	ExternalData
Function	Consommer	CSVSourceUnit	ExternalData
Nature	TransmettreTelecom	GeneratedSimulationUnit	TelecomUnit

TABLE 8.3 – Liste des `Mappings` du modèle d'allocation AllocationML de l'Île de Sein

connexions représentent la possibilité d'échanger un signal de donnée, variable au cours du temps.

CosiML est construit pour pouvoir représenter ces informations, et les centraliser au sein d'un unique modèle. Les règles sémantiques du langage garantissent qu'un modèle est valide lorsque les types de donnée en entrée et sortie d'une connexion sont identiques, et ont la même variabilité (discrète ou continue). Un avertissement est levé lorsque des ports d'entrée ne sont pas connectés.

S'il est possible de développer ce modèle directement, nous disposons au sein du *Smart Grid Simulation Framework* d'un outil de transformation de modèles, depuis trois modèles écrits en SGridML, CatalogML et AllocationML, vers un modèle écrit en CosiML. Ainsi, dans notre cas d'étude, les unités de simulation qui apparaîtront dans le scénario de cosimulation sont celles listées dans le modèle CatalogML (figure 8.2 : GridUnit, EmsUnit, ExternalData, et TelecomUnit), et leurs interfaces dépendent des fonctions du modèle SGridML qui leur sont allouées via le modèle AllocationML (figure 8.3).

Les règles de création des interfaces des éléments `SimulationUnit` et `Link` du modèle de cosimulation CosiML sont données à la sous-section 7.3.2.

Ainsi, les différents acteurs chargés du développement des unités de simulation – les *experts en modélisation et simulation* – ont le nom et la description des fonctions que leur modèle doit implémenter,

et l'interface qu'il doit réaliser (avec le nom des entrées, des sorties, leur type et leur variabilité).

8.2.1.4 Développement des unités de simulation

L'objectif de notre contribution est de faciliter l'entente entre les différents *experts en modélisation et simulation* chargés – individuellement – de développer les modèles de simulation, tout en réutilisant au maximum leur compétences et outils de travail existants. Nous expliquons ci-dessous succinctement la façon dont sont développés ces modèles de simulation, sachant qu'ils sont considérés dans notre approche comme des boîtes noires dont seules les interfaces nous intéressent.

GridUnit Comme indiqué dans le début de la section, nous nous attendons à ce que l'expert chargé de l'unité de simulation `GridUnit` fournisse directement une FMU valide avec le modèle CosiML. Pour notre implémentation, nous avons choisi le langage Modelica, et l'éditeur Dymola (voir capture d'écran figure 8.3) qui implémente la fonctionnalité automatique d'export d'un modèle au format FMU.

Depuis les *modèles de conception* établis aux étapes précédentes, l'*expert en M&S* chargé du développement de cette unité sait qu'il doit implémenter les comportements de simulation suivants :

— `CalculerPointDeFonctionnement`,
— `CalculerPuissancePv`,
— `StockerEnergie`.

Modelica permet d'exprimer un modèle sous forme d'équations principalement, mais également d'utiliser des fonctionnalités de détection d'évènement, comme des "fronts montants" ou "descendant". Ainsi, le modeleur peut directement implémenter notre convention d'encodage et décodage des signaux discrets, présentée à la section 5.3.

Notre modèle implémente une stratégie de *loadflow* – parfois appelé "modèle de débit de la charge" – qui est une pratique courante en électrotechnique pour évaluer l'évolution des puissances sur l'ensemble des nœuds d'un réseau électrique.

ExternalData Dans une première version, n'ayant pas encore accès aux données réelles du réseau, nous construisons les courbes de consommation électrique et la courbe des puissances de l'ensoleillement, en fonction des valeurs moyennes relevées sur l'île, et de l'allure classique de ces types de courbe. Nous construisons des courbes très simples, qui changent de valeur toutes les heures, afin de simplifier l'analyse des résultats et de vérifier que les autres modèles de simulation réagissent bien comme attendus.

La figure 8.4 montre l'allure de ces courbes, montrant l'évolution de la puissance active (en kW) en fonction du temps (en h).

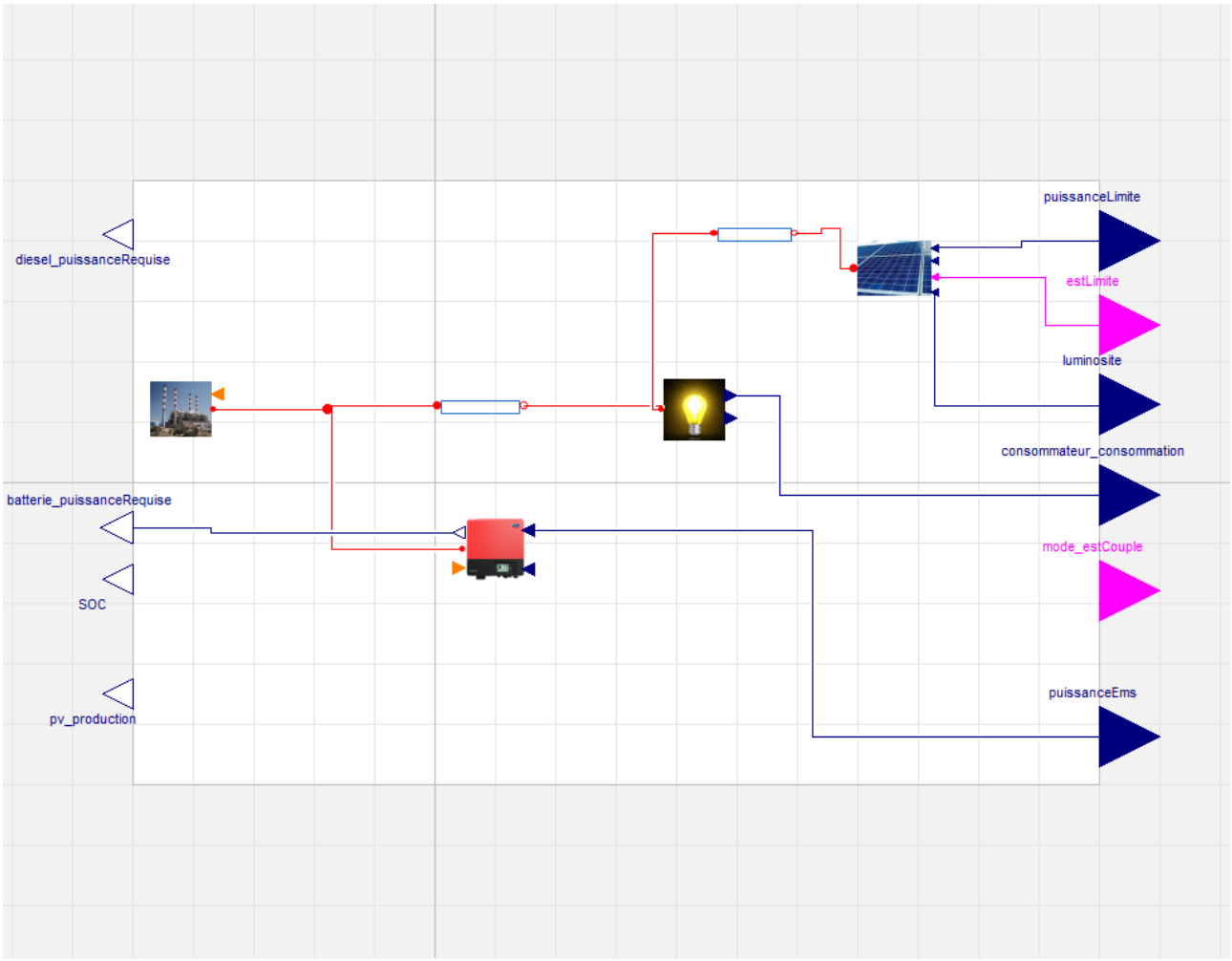

FIGURE 8.3 – Capture d'écran du logiciel Dymola et du modèle de *loadflow* du réseau électrique de l'île de Sein

TelecomUnit Nous utilisons OMNeT++ pour développer le modèle du réseau de télécommunications de notre cas d'étude. La figure 8.5 montre un extrait du *Network Description* écrit en NED, et définissant les modules du réseau et les connexions entre elles.

EmsUnit L'unité de simulation `EmsUnit` modélise le comportement de la fonction `CalculerConsignes` de l'EMS du réseau de l'île de Sein. Nous choisissons un mode de fonctionnement simple, dans lequel l'EMS ne dispose pas d'estimations de la consommation et de la production en avance. Nous modélisons un algorithme considérant un pas de temps d'envoi de consigne tous les quarts d'heure, en considérant que les valeurs mesurées de consommation et de production seront constantes sur ce pas de temps. En cas d'urgence, c'est-à-dire dans les cas où ces valeurs varient grandement, et que certains seuils de sécurité sont dépassés, cette fonction doit calculer et envoyer "immédiatement" de nouvelles consignes.

Nous construisons un modèle en Java, dont le diagramme de classe est donné dans la figure 8.6.

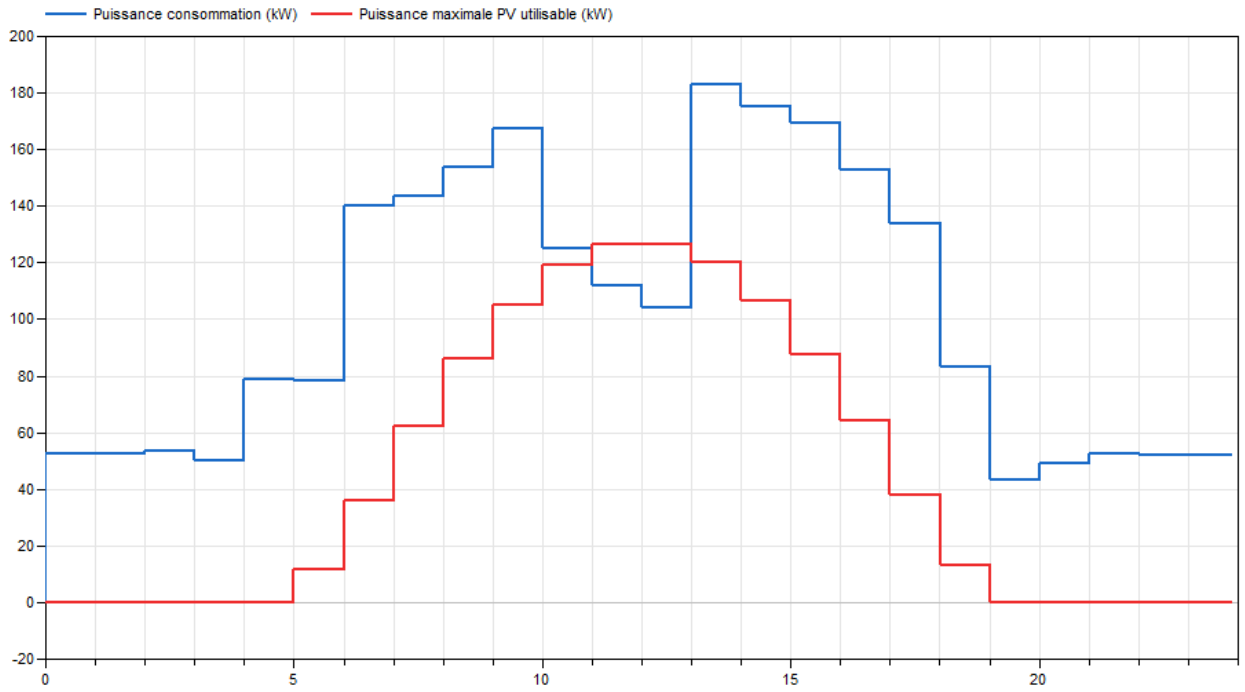

FIGURE 8.4 – Évolution de la puissance consommée et de la puissance photovoltaïque disponible

8.2.2 Phase 2 : Configurer la cosimulation FMI

La seconde phase de la démarche est réalisée de manière quasi-automatique lorsque nous utilisons les outils du *Smart Grid Simulation Framework*, comme illustré précédemment.

8.2.2.1 Adapter les modèles en FMU

Nous utilisons le standard FMI pour réaliser la cosimulation de notre scénario de cosimulation. Le format attendu pour l'ensemble des unités de simulation est donc le format FMU.

Nous avons déjà effectué dans notre modèle CatalogML (et copié dans le modèle CosiML) la distinction entre les unités de simulation fournies au format final, et celles qui requièrent une étape d'adaptation. En effet, seule l'unité `GridUnit` est directement fournie au format FMU, comme le montre le choix d'instancier un élément `CosiML::ProvidedSimulationUnit` dans nos modèles de haut niveau pour la représenter.

C'est le rôle des *générateurs d'adaptateur* fournis avec le langage CosiML de générer les éléments nécessaires à la transformation des unités de simulation non-FMU, en FMU. Les générateurs actuellement disponibles fonctionnent avec des modèles en Java, des modèles OMNeT++ , et des fichiers CSV. S'il ne l'est pas déjà, le modèle CatalogML doit être mis à jour avec les informations spécifiques aux modèles de simulation développés (chemins de l'emplacement, nom des modèles, etc.) et le modèle CosiML re-généré avec ces nouvelles informations. Puis, l'activation des *générateurs d'adaptateur* nous

```
network SI1 {
        submodules:
                Ems: CommModule { ... }
                PV: CommModule { ... }
                Batt: CommModule { ... }
                Diesel: CommModule { ... }
...

...
        connections:
                Ems.out++ --> { delay = 0.1ms; } --> PV.in++;
                Ems.in++ <-- { delay = 0.1ms; } <-- PV.out++;
                Ems.out++ --> { delay = 0.1ms; } --> Batt.in++;
                Ems.in++ <-- { delay = 0.1ms; } <-- Batt.out++;
                Ems.out++ --> { delay = 0.1ms; } --> Diesel.in++;
                Ems.in++ <-- { delay = 0.1ms; } <-- Diesel.out++;
```

FIGURE 8.5 – Extrait du *Network Description* du modèle OMNeT++ de TelecomUnit

permet de générer les fichiers et les scripts d'automatisation de la transformation pour nos unités de simulation `EmsUnit`, `TelecomUnit`, et `ExternalData`.

L'objectif de cette étape est généralement d'avoir à disposition toutes les FMU du scénario de cosimulation. Les scripts de transformation de nos modèles de simulation au format FMU peuvent être exécutés dès à présent. Mais en utilisant le *Smart Grid Simulation Framework* convenablement, nous savons qu'un script d'exécution de la cosimulation sera généré à la fin de cette phase, qui va se charger lui-même d'exécuter les scripts de transformation. Ceci dans le but de réduire le nombre d'actions de l'utilisateur, et de favoriser les actions de modification et d'itération sur les modèles.

8.2.2.2 Implémenter le scénario de cosimulation dans le *master* FMI

Cette étape correspond à la configuration du *master* FMI avec un scénario de cosimulation spécifique. Cela signifie définir les FMU qui interviennent dans la simulation, et les signaux pouvant être échangés entre elles. Notre démarche imposant à toutes nos unités de simulation d'être fournies au format FMU, n'importe quel logiciel implémentant un *master* compatible avec le standard FMI peut être utilisé. Néanmoins, notre *Smart Grid Simulation Framework* dispose d'un générateur du fichier de configuration spécifique au logiciel DACCOSIM. Ce fichier généré permet notamment d'instancier un scénario de cosimulation dans le logiciel, en explicitant les chemins vers les FMU à piloter, et les connexions réalisables entre elles. Il configure également les paramètres d'exécution comme la date de début et de fin de la cosimulation, et le pas de temps utilisé (seul le pas de temps constant est pour le moment géré).

Comme pour la précédente, cette étape est donc réalisée de manière complètement automatique dans

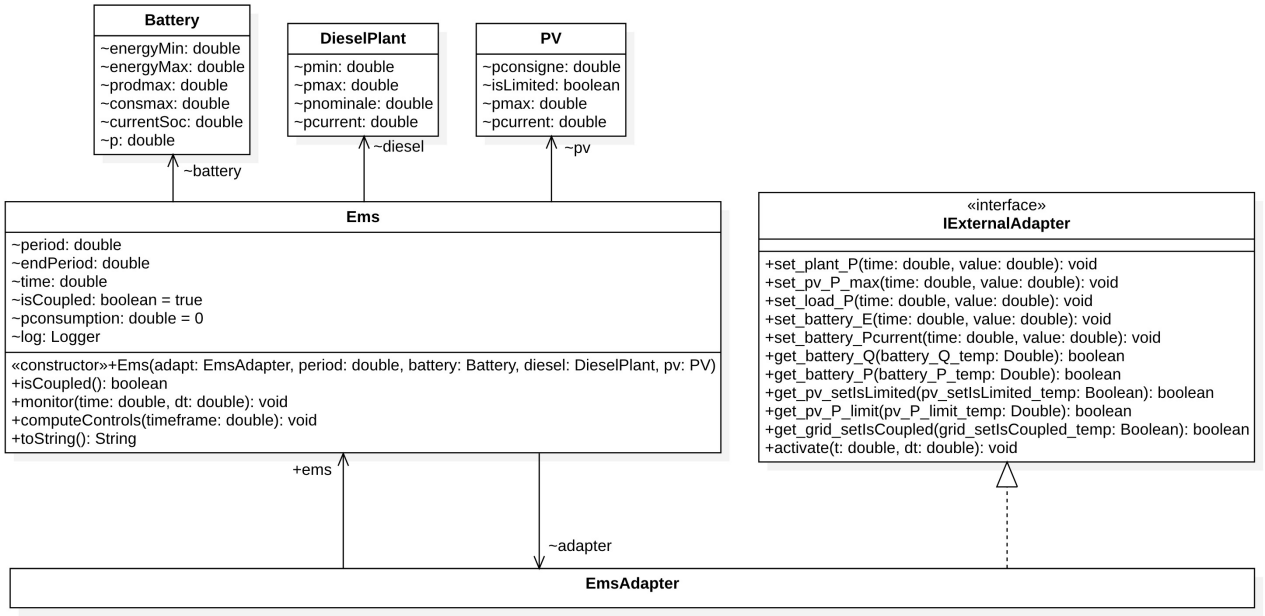

FIGURE 8.6 – Diagramme des classes Java du modèle de EMSUnit

notre cas d'étude.

8.2.3 Phase 3 : Analyser les résultats

La dernière phase de la démarche est l'analyse des résultats, permettant de décider s'il est nécessaire d'effectuer une itération sur les modèles construits et de reprendre le déroulement de la démarche depuis l'une des phases précédentes.

8.2.3.1 Évaluer les résultats de simulation

La cosimulation est réalisée par l'exécution du *script de cosimulation* généré précédemment. Les résultats sont obtenus au format CSV. La figure 8.7 affiche l'évolution des puissances consommées et produites par les différents équipements du réseau électrique simulé, ainsi que l'état de charge de la batterie (entre ses valeurs maximale et minimale).

Plusieurs remarques peuvent être effectuées sur ces résultats à différents niveaux d'analyse :

1. Aucune erreur n'est apparue lors de la simulation. Le *chargé de cosimulation* peut donc fournir les résultats aux *experts de la modélisation*.
2. Les résultats sont cohérents avec ce que nous pouvions attendre. Les courbes présentent un pas de variation d'une heure, comme le font les données de consommation et d'illumination utilisées en entrée de la cosimulation. De plus, les valeurs restent dans les intervalles de contrainte physiques : la charge de la batterie ne varie qu'entre 0 et 100%, les équipements ne produisent pas plus que

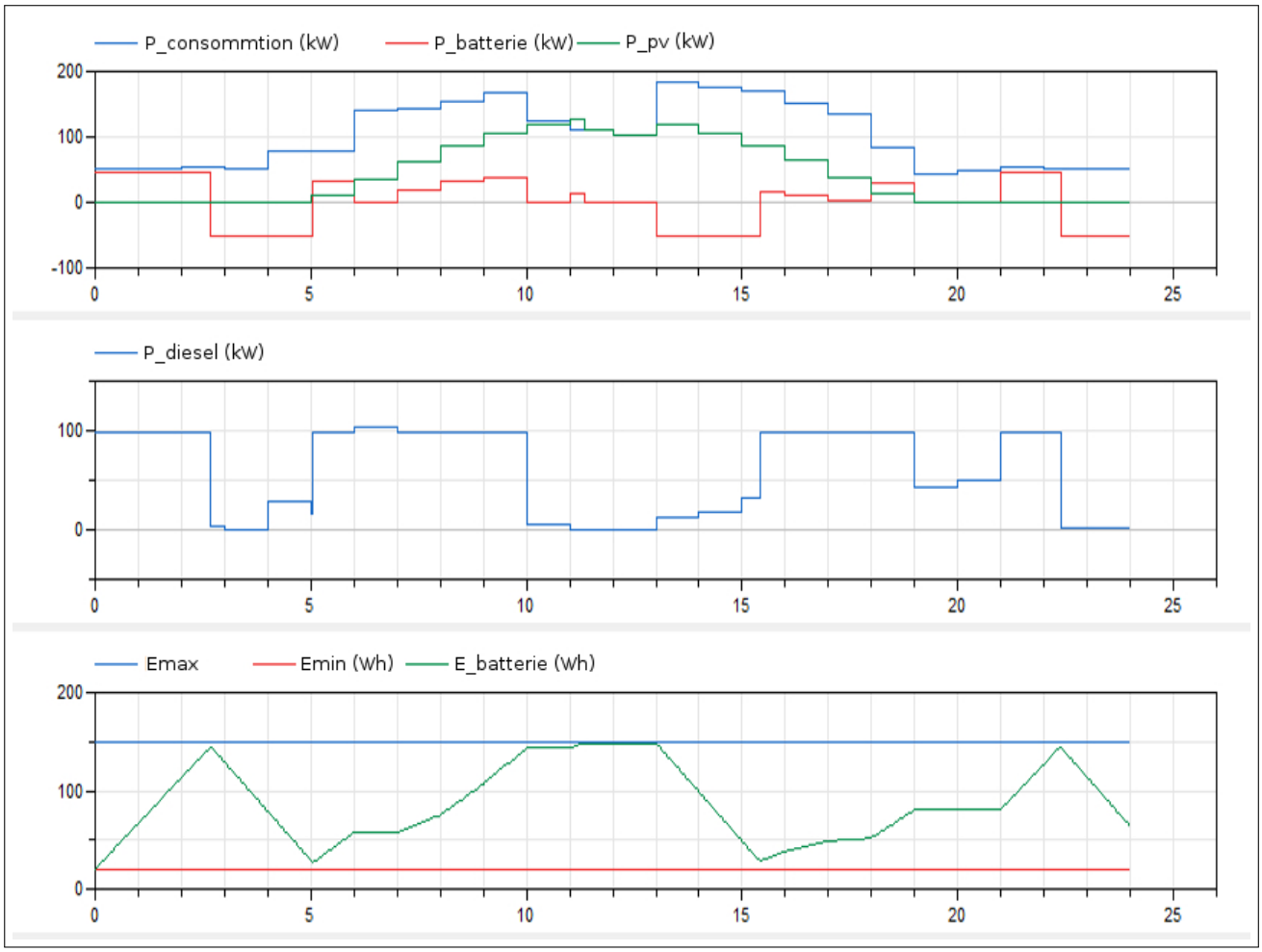

FIGURE 8.7 – Évolution de la puissance consommée et de la puissance photovoltaïque disponible

leur valeur maximale, etc. Les résultats peuvent donc être interprétés pour juger de l'efficacité du système modélisé en fonction des besoins initiaux.

3. Dans notre cas, principalement deux interprétations des résultats peuvent être données :

   — L'équilibre entre la consommation et la production a été maintenu tout au long de la durée simulée, et le réseau a fonctionné normalement. Cette contrainte critique ayant été satisfaite, la solution modélisée est acceptable sur les conditions initiales (c'est-à-dire sur une durée d'une journée avec un profil de consommation donné).

   — La batterie est utilisée et remplit globalement son rôle, qui est d'apporter de la flexibilité à la gestion de l'équilibre entre production. Néanmoins, autour de 12h, alors que la luminosité est maximale, la source photovoltaïque est contrainte à produire sous son maximum de puissance. En effet, nous pouvons constater que la batterie est saturée (plus de capacité de charge), et que la centrale diesel est éteinte. La gestion et le pilotage des sources n'a donc pas été optimal sur la période de fonctionnement.

8.2.3.2 Prendre les décisions d'itération ou valider le modèle de la solution

L'un des objectif principaux est d'optimiser l'utilisation du photovoltaïque par rapport à l'utilisation de la centrale diesel dans le réseau électrique de l'Île de Sein. L'efficacité de notre solution modélisée peut donc être évaluée en comparant le ratio d'énergie injectée dans le réseau entre les différentes sources de production, par rapport à celui obtenu dans d'autres configurations du système. Néanmoins, comme précisé au chapitre 2, sans le recours aux méthodes analytiques, il n'est pas possible de savoir si la solution est optimale ou non. Il est nécessaire d'itérer sur le modèle de la solution et d'effectuer de nouvelles simulations pour les comparer entre elles, et éventuellement avec les mesures effectuées sur le système réel lorsqu'il en existe déjà un.

Comme nous venons de le voir, nos résultats de simulation confirment que notre modèle actuel permet le fonctionnement normal du réseau électrique sur les conditions simulées. Néanmoins, le critère de maximisation de l'énergie injectée par la source photovoltaïque peut être amélioré, par exemple en évitant la saturation de la batterie au milieu de la journée. Plusieurs choix peuvent être réalisés afin d'améliorer ce critère :

1. Une première décision simple à mettre en œuvre pourrait être d'augmenter la capacité de la batterie utilisée. Une telle itération impacterait uniquement le modèle de simulation du comportement électrotechnique, à savoir le modèle de la FMU `GridUnit`. Les autres modèles (de simulation, ainsi que le modèle SGridML) restant inchangés, la simulation peut être entièrement et automatiquement ré-exécutée grâce à notre chaîne d'outils de génération. En revanche, ce genre de décision entraîne un coût matériel plus important lorsque il faudra implémenter la solution modélisée dans le système réel.
2. Une seconde décision, entraînant un coût matériel de déploiement moindre, pourrait être d'améliorer la fonction de pilotage intelligent de la production `CalculerConsignes` (voir sous-section 8.2.1), implémentée par l'EMS. Dans notre solution actuelle, cette fonction ne repose que sur des mesures effectuées sur le réseau. Nous pourrions l'améliorer en utilisant également des informations prédictives sur la consommation et l'illumination. Cette itération demande de modifier le modèle SGridML, en ajoutant de nouveaux éléments `Functions` produisant ces données prédictives, et en les connectant à l'élément `CalculerConsignes` existants, qui devra déclarer de nouveaux `Inputs` et `Outputs` (voir tableau 8.2). Notre chaîne de génération permet d'identifier les modèles impactés, et les nouvelles interfaces structurelles à implémenter dans les FMU, et de re-générer automatiquement les scripts de configuration et d'exécution.

8.3 Observations

Nous avons présenté dans ce chapitre l'utilisation de notre *Smart Grid Simulation Framework* sur le cas d'étude de l'Île de Sein. Les différents outils de génération développés permettent le développement facilité des FMU constituant la cosimulation finale, ainsi que la génération automatique des scripts de

configuration et déploiement de la cosimulation. Ainsi, nous permettons aux différents collaborateurs de la simulation de ne concentrer leurs efforts que sur les modèles de conception et de simulation, et d'implémenter des itérations sur la conception du système avec un minimum d'effort. Nous illustrons ceci en généralisant le cas d'étude de l'Île de Sein à travers plusieurs exemples de scénarios de modélisation et simulation de Smart Grid.

Dans un premier scénario, un architecte fonctionnel doit comparer plusieurs propositions entre différents fournisseurs, afin par exemple de trouver la meilleure solution d'EMS pour son système. Il peut préciser dans son appel d'offre la nécessité de fournir un modèle de simulation, au format FMU protégeant la propriété intellectuelle de leurs auteurs. Afin de garantir la bonne intégration du composant fourni avec la plateforme de cosimulation et les autres modèles déjà développés, il peut inclure les contraintes d'interface qu'il aura déduites de son modèle CosiML, automatiquement généré à partir de son modèle de conception en SGridML. La sélection du meilleur composant, et donc de la meilleure solution, est également simplifiée car :

— en utilisant *Smart Grid Simulation Framework*, l'architecte peut rapidement constituer un environnement de test unitaire du composant, en constituant notamment des fichiers de données CSV qui seront automatiquement générés en FMU (via l'élément `CosiML::CSVSourceUnit`),
— passer d'un composant à l'autre est alors aussi simple que de changer l'attribut `pathUnit` de l'élément `ProvidedSimulationUnit` du modèle de catalogue écrit avec CatalogML.

Smart Grid Simulation Framework permet d'obtenir rapidement les résultats de simulation dans chaque configuration, afin de les comparer à ceux obtenus dans les autres configurations.

Dans un deuxième scénario, nous impliquons un expert électrotechnique pour réaliser un modèle de simulation du réseau électrique, et l'intégrer à un projet de cosimulation existant, utilisant le `Smart Grid Simulation Framework`. Cela est possible sans recourir à une période d'apprentissage intensive, car il peut utiliser ses propres outils de modélisation et simulation (Dymola, PowerFactory, etc.), et il n'y a qu'un faible nombre de concepts (entrée, sortie, variabilité discrète et continue) qu'il doit éventuellement appréhender afin de comprendre la syntaxe du modèle de conception écrit avec SGridML. Il peut alors prendre en main l'ensemble de la chaîne de génération afin de tester seul son modèle et l'améliorer dans une stratégie itérative sans impliquer les autres collaborateurs, dont le travail est automatiquement et facilement intégré à l'unité de cosimulation.

Enfin, dans un troisième scénario nous considérons le cas où la modification d'un modèle de simulation implique une modification dans le modèle de conception écrit en SgridML, et plus particulièrement parmi les liens de couplage entre `Functions`, comme par exemple la modification du nom, ou l'ajout, d'un `Input`. Les trois phénomènes suivants apparaissent et garantissent le maintien de la cohérence entre les modèles de simulation :

1. Les règles de validation syntaxiques de SGridML imposent d'avoir un couplage cohérent entre les différentes fonctions de simulation. Par exemple tout `Input` doit être lié à un `Output`.

2. L'exécution du processus de génération des artefacts de simulation peut générer automatiquement une partie des ajustements nécessaires sur les modèles de simulation.

3. Avec la génération automatique des scripts de configuration et d'exécution de la cosimulation, il est facile et rapide d'identifier les modèles impactés par la modification. En effet, des erreurs sont levées à l'exécution de la cosimulation tant que les FMU impactées n'ont pas réalisé les ajustements nécessaires au niveau de leur interface.

8.4 Conclusion du chapitre

Smart Grid Simulation Framework est construit autour d'une approche d'ingénierie dirigée par les modèles. Il repose sur le développement de plusieurs modèles de haut niveau, d'opérations et de transformations, afin de garantir la cohérence entre le développement et l'exécution des différents modèles de simulation du Smart Grid.

Nous montrons dans ce chapitre comment intégrer cet environnement à un projet de modélisation d'un Smart Grid, intégrant la simulation comme outil de validation du comportement de la solution modélisée. Nous faisons notamment le lien entre les étapes de la démarche itérative établies dans le chapitre 5, et les différents outils de notre environnement présentés dans les chapitres 6 et 7.

Pour résumer, un modèle de conception du système à simuler est réalisé à l'aide du langage SGridML, qui est ensuite transformé automatiquement selon les informations d'allocation fournies dans des modèles écrits avec AllocationML et CatalogML, en un modèle de cosimulation conforme à CosiML. Le modèle de la cosimulation ainsi généré référence les unités de simulation fournies par les experts en modélisation et caractérise les liens de couplage entre elles. À partir de ce modèle, les scripts de configuration et d'exécution de la cosimulation sont automatiquement générés. Cette approche est fondée sur l'enchaînement de générations automatiques afin de minimiser l'effort fourni et le risque d'erreur à chaque itération de la démarche. Des liens entre les différents modèles de simulation sont tracés et évalués au cours des différentes étapes, ce qui facilite également la réalisation en parallèle du travail des différents collaborateurs de la modélisation.

Néanmoins, la cohérence entre les différents modèles de simulation est principalement évaluée au niveau de leur interface, et non au niveau de leur implémentation interne. Cela permet de contraindre les experts en modélisation à réaliser des modèles compatibles à l'exécution de la cosimulation, qui ne produiront pas d'erreur, mais ne donne pas de garanties fortes sur la pertinence et la validité de ces résultats. Il s'agit ici d'un compromis que nous avons dû faire, entre la faisabilité d'un prototype fonctionnel et évolutif mais ne garantissant pas une cohérence complète entre les modèles de simulation, et la mise en place d'une approche plus complète basée sur de l'analyse sémantique, mais plus lourde à développer et rendant l'intégration de nouveaux outils de modélisation et de simulation plus compliquée. Le contexte industriel de nos travaux et l'intérêt économique nous ont décidé à préférer la première option.

Troisième partie

Conclusions et Perspectives

9 Conclusions et perspectives

Sommaire

9.1 Conclusion générale 127

9.1.1 Rappel du contexte et des objectifs de nos travaux 127

9.1.2 Résumé des contributions 128

9.2 Perspectives 129

9.1 Conclusion générale

9.1.1 Rappel du contexte et des objectifs de nos travaux

Nos travaux de thèse s'inscrivent dans un enjeu industriel qui est de faciliter l'étude des systèmes énergétiques de demain, ou Smart Grids, en permettant la conception de composants plus sûrs, maîtrisés, afin de limiter les tests et déploiements matériels souvent coûteux dans ce secteur d'activité. Reconnaissant que la simulation est une pratique indispensable pour vérifier et valider au plus tôt les systèmes en cours de conception, nos travaux répondent à la problématique de recherche des outils et méthodes permettant de simuler un Smart Grid, et qui permettent de rester compatible avec les contraintes, notamment économiques, du milieu industriel (voir **problématique générale** à la sous-section 1.3.1, voir p. 5). Ces différentes contraintes sont synthétisées dans une liste de 5 critères qu'il est nécessaire de prendre en compte et d'optimiser (sous-section 1.3.1).

Les Smart Grids sont des systèmes complexes, combinant à l'instar de tous les systèmes cyber-physiques des comportements hétérogènes répartis entre plusieurs modèles, eux-mêmes développés par plusieurs personnes aux profils techniques et compétences différents. Nous considérons la cosimulation avec FMI comme la meilleure solution pour prendre en compte l'ensemble de ces modèles et évaluer le comportement d'un Smart Grid.

Notre problématique générale est détaillée dans nos **problématiques 1 et 2**. À partir de l'étude de la littérature, nous avons identifié les différents points devant être traités pour permettre de répondre à ces deux problématiques :

1. Répondre à notre **problématique 1 (hétérogénéité)** suppose de permettre l'intégration des outils de simulation des différents domaines du Smart Grid au sein d'un cosimulateur FMI. Un premier problème est qu'il n'existe pas de logiciel de simulation du réseau de télécommunication permettant l'export au format FMU. De plus, le fonctionnement du standard FMI n'est pas compatible avec l'échange de signaux discrets. Cela pose un second problème dans le cas de la cosimulation d'un système cyber-physique comme le Smart Grid, où les modèles du comportement informatique et des télécommunications manipulent des données évoluant de manière discrète.
2. Répondre à notre **problématique 2 (cohérence)** suppose de permettre la réalisation de modèles de simulation qui soient cohérents entre eux et avec les modèles de conception du Smart Grid étudié. Si les pratiques de l'IDM, et plus particulièrement la transformation de modèles, sont utilisées dans la littérature pour lier des modèles de différents niveaux d'abstraction, nous n'avons pas trouvé de références utilisant ces techniques pour mettre en cohérence des modèles de conception d'un Smart Grid et des modèles de simulation.

9.1.2 Résumé des contributions

Nos contributions facilitent l'utilisation de la simulation pour la conception d'un Smart Grid, en utilisant des outils et techniques d'IDM afin de proposer un environnement outillé complet : le *Smart Grid Simulation Framework*. Nous avons utilisé une approche dite "bottom-up" (de "bas en haut") pour construire notre environnement. Nous nous sommes d'abord concentrés sur les difficultés techniques liées à l'exécution de la cosimulation d'un Smart Grid avec FMI, ce qui nous a permis de traiter la problématique 1. Puis, nous nous sommes intéressés à la façon d'intégrer nos développements à une démarche industrielle de conception d'un Smart Grid, ce qui nous a conduit à répondre à la problématique 2.

Dans un premier temps nous proposons une convention de structuration des interfaces de deux FMU afin de leur permettre d'échanger des signaux discrets avec FMI. Nous avons alors pu développer une extension du logiciel OMNeT++ , `fmi4omnetpp`, permettant l'export d'un modèle de télécommunication au format FMU et son utilisation dans un scénario de cosimulation de Smart Grid avec FMI. Cette extension implémente notre convention d'échange de signaux discrets avec FMI.

Avec ces deux réalisations, il est possible d'intégrer un modèle du comportement de télécommunications, un modèle du comportement informatique et un modèle du comportement électrotechnique au sein d'une même cosimulation FMI, ce qui correspond à la problématique 1. Cette approche est entièrement compatible avec le standard FMI, et permet de conserver ses avantages en regard des critères à prendre en compte (voir sous-section 1.3.1). Ainsi, le format FMU protégeant la propriété intellectuelle des modèles, le **critère 4** est optimisé. L'utilisation d'un standard permet également d'optimiser le **critère 3** en facilitant la collaboration et les échanges entre plusieurs équipes.

Nous avons conçu le DSL CosiML afin de tirer parti des techniques d'IDM permettant de diminuer les erreurs et automatiser les tâches des collaborateurs de la cosimulation d'un Smart Grid, optimisant le critère 1 (diminution du coût et des efforts) et le critère 2 (intégration à une démarche itérative).

Enfin, nous avons créé le langage SGridML et développé une transformation automatique vers le langage CosiML afin de répondre à la problématique 2. SGridML permet de représenter un Smart Grid d'un point de vue fonctionnel et de construire un modèle d'analyse et de conception de la solution à évaluer par la simulation. La création des langages AllocationML et CatalogML nous permet de répartir les comportements à simuler entre différents modèles de simulation, La simplicité de SGridML lui permet d'être appréhendé par tous les experts en modélisation et simulation (critère 1 de coût et d'effort), et la transformation de SGridML vers CosiML permet de garantir la cohérence du modèle de conception avec le scénario de la cosimulation, notamment avec les interfaces des unités de simulation et les liens de couplage entre elles. Ici encore, l'automatisation et la génération automatique optimise le critère 1 (diminution du coût et des efforts) et le critère 2 (intégration à une démarche itérative).

Nos travaux ont conduit à la création d'un environnement de simulation de Smart Grids utilisable, testé sur un cas d'étude réel de EDF. Nous partageons le code de nos développements dans un dépôt [1] sur la plateforme GitHub, et fournissons les FMU (au format binaire, protégeant la propriété intellectuelle des modèles) du cas d'étude du réseau électrique de l'Île de Sein.

9.2 Perspectives

Nous identifions plusieurs axes d'amélioration de notre approche à court et long termes. Avant tout, l'environnement proposé reste un prototype fonctionnel, mais peut être amélioré en termes d'ergonomie, de documentation utilisateur, etc. L'une des pistes à court terme est de faire évoluer la syntaxe de nos langages spécifiques SGridML, AllocationML et CatalogML. En effet, une syntaxe à l'aide de diagrammes pourrait être plus accessible et parlante pour les utilisateurs. Une autre avancée importante pourrait être de gérer différents niveaux de composition hiérarchiques dans le modèle fonctionnel en SGridML, une étape supplémentaire d'aplanissage du modèle serait alors nécessaire pour pouvoir continuer d'utiliser la chaîne de génération fournie. Il s'agit néanmoins d'ajouts périphériques, ne modifiant pas le cœur du fonctionnement de *Smart Grid Simulation Framework*.

Une perspective d'amélioration plus ambitieuse concerne le support de la prochaine norme du standard FMI, la version 3. D'après les ébauches et les acteurs concernés dans l'évolution de la norme, cette nouvelle version apportera de nouveaux outils pour permettre la détection d'événements dans les FMU au cours d'une cosimulation, ce qui permettrait de supporter nativement l'échange de signaux discrets entre FMU. Une refonte des outils de génération de notre environnement, ainsi que de notre extension `fmi4omnetpp`, serait alors à réaliser afin d'utiliser ces outils plutôt que notre méthode par encodage/décodage de signaux continus.

1. `https://github.com/davidoudart-pro/SGridSF`

Par ailleurs, plusieurs difficultés n'ont pas été abordées dans cette thèse mais pourraient également faire l'objet de futurs travaux :

— afin d'augmenter la réutilisabilité dans d'autres projets de cosimulation des FMU développées, une approche d'adaptation sémantique pourrait être intégrée à notre environnement, comme proposé dans les travaux de Gomes [Gomes et al., 2018a],

— la gestion de différentes échelles de temps est une difficulté identifiée dans la cosimulation de systèmes complexes.

Bibliographie

[Andrén et al., 2017] Andrén, F., Strasser, T., and Kastner, W. (2017). Engineering Smart Grids : Applying Model-Driven Development from Use Case Design to Deployment. *Energies*, 10(3) :374. 33, 35

[Armour et al., 2003] Armour, F., Kaisler, S., Getter, J., and Pippin, D. (2003). A UML-driven enterprise architecture case study. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of The*, page 10 pp. IEEE. 52

[Awais et al., 2013] Awais, M. U., Palensky, P., Elsheikh, A., Widl, E., and Matthias, S. (2013). The high level architecture RTI as a master to the functional mock-up interface components. In *2013 International Conference on Computing, Networking and Communications (ICNC)*, pages 315–320, San Diego, CA. IEEE. 50

[Bernard, 2012] Bernard, S. A. (2012). *An Introduction to Enterprise Architecture : Third Edition*. AuthorHouse. 28

[Bezivin and Gerbe, 2001] Bezivin, J. and Gerbe, O. (2001). Towards a precise definition of the OMG/MDA framework. In *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pages 273–280. 12, 23

[Blochwitz et al., 2011] Blochwitz, T., Otter, M., Arnold, M., Bausch, C., Elmqvist, H., Junghanns, A., Mau\s s, J., Monteiro, M., Neidhold, T., Neumerkel, D., et al. (2011). The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference ; March 20th-22nd ; Technical Univeristy ; Dresden ; Germany*, pages 105–114. Linköping University Electronic Press. 47

[Boulanger et al., 2011] Boulanger, F., Hardebolle, C., Jacquet, C., and Marcadet, D. (2011). Semantic Adaptation for Models of Computation. In *2011 Eleventh International Conference on Application of Concurrency to System Design*, pages 153–162, Newcastle Upon Tyne. IEEE. 14

[Brambilla et al., 2017] Brambilla, M., Cabot, J., and Wimmer, M. (2017). *Model-Driven Software Engineering in Practice : Second Edition*. Morgan & Claypool Publishers, 2nd edition. 23

[Broman et al., 2015] Broman, D., Greenberg, L., Lee, E. A., Masin, M., Tripakis, S., and Wetter, M. (2015). Requirements for Hybrid Cosimulation Standards. In *Proceedings of the 18th International Conference on Hybrid Systems : Computation and Control*, HSCC '15, pages 179–188, Seattle, Washington. ACM. 54

[Bruinenberg et al., 2012] Bruinenberg, J., Colton, L., Darmois, E., Dorn, J., Doyle, J., Elloumi, O., Englert, H., Forbes, R., Heiles, J., Hermans, P., and Uslar, M. (2012). *CEN -CENELEC - ETSI : Smart Grid Coordination Group - Smart Grid Reference Architecture Report 1.0*. 32

[Camus, 2015] Camus, B. (2015). *Multi-Agent Environment for Multi-Modeling and Simulation of Complex Systems*. Theses, Université de Lorraine. 16

[Chatzivasileiadis et al., 2016] Chatzivasileiadis, S., Bonvini, M., Matanza, J., Yin, R., Nouidui, T. S., Kara, E. C., Parmar, R., Lorenzetti, D., Wetter, M., and Kiliccote, S. (2016). Cyber–Physical Modeling of Distributed Resources for Distribution System Operations. *Proceedings of the IEEE*, 104(4) :789–806. 51, 53, 54, 76

[Combemale, 2008] Combemale, B. (2008). Ingénierie Dirigée par les Modèles (IDM)–État de l'art. 23

[Combemale et al., 2017] Combemale, B., Barais, O., and Wortmann, A. (2017). Language engineering with the GEMOC studio. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 189–191. 25

[Couto and Pierce, 2017] Couto, L. D. and Pierce, K. (2017). Modelling Network Connections in FMI with an Explicit Network Model. In *THE 15TH OVERTURE WORKSHOP*. 53

[Cremona et al., 2016] Cremona, F., Lohstroh, M., Tripakis, S., Brooks, C., and Lee, E. A. (2016). FIDE : An FMI integrated development environment. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing - SAC '16*, pages 1759–1766, Pisa, Italy. ACM Press. 50, 76

[Dahmann and Morse, 1998] Dahmann, J. S. and Morse, K. L. (1998). High Level Architecture for simulation : An update. In *Proceedings. 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications (Cat. No.98EX191)*, pages 32–40. 46

[Dänekas et al., 2014] Dänekas, C., Neureiter, C., Rohjans, S., Uslar, M., and Engel, D. (2014). Towards a Model-Driven-Architecture Process for Smart Grid Projects. In Benghozi, P.-J., Krob, D., Lonjon, A., and Panetto, H., editors, *Digital Enterprise Design & Management*, volume 261, pages 47–58. Springer International Publishing, Cham. 32, 33

[de Lara and Vangheluwe, 2002] de Lara, J. and Vangheluwe, H. (2002). AToM3 : A Tool for Multi-formalism and Meta-modelling. In Goos, G., Hartmanis, J., van Leeuwen, J., Kutsche, R.-D., and Weber, H., editors, *Fundamental Approaches to Software Engineering*, volume 2306, pages 174–188. Springer Berlin Heidelberg, Berlin, Heidelberg. 36

[Durling et al., 2017] Durling, E., Palmkvist, E., and Henningsson, M. (2017). FMI and IP protection of models : A survey of use cases and support in the standard. In *The 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, pages 329–335. 48

[Électricité de France, 2018] Électricité de France (2018). *Les systèmes électriques de demain : un défi pour la transition énergétique*. OCLC : 1043837240. 2

[Elsheikh et al., 2013] Elsheikh, A., Awais, M. U., Widl, E., and Palensky, P. (2013). Modelica-enabled rapid prototyping of cyber-physical energy systems via the functional mockup interface. pages 1–6. IEEE. 36, 51, 60

[Erraissi and Belangour, 2018] Erraissi, A. and Belangour, A. (2018). Data sources and ingestion big data layers : Meta-modeling of key concepts and features. *International Journal of Engineering and Technology(UAE)*, 7 :3607–3612. 24

[Évora Gómez et al., 2019] Évora Gómez, J., Hernández Cabrera, J. J., Tavella, J.-P., Vialle, S., Kremers, E., and Frayssinet, L. (2019). Daccosim NG : Co-simulation made simpler and faster. In *The 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*, pages 785–794. 92

[Galán et al., 2009] Galán, J. M., Izquierdo, L. R., Izquierdo, S. S., Santos, J. I., del Olmo, R., López-Paredes, A., and Edmonds, B. (2009). Errors and artefacts in agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 12(1) :1. 16, 17, 61

[Galtier et al., 2017a] Galtier, V., Ianotto, M., Caujolle, M., Corniglion, R., Tavella, J.-P., Gómez, J. É., Cabrera, J. J. H., Reinbold, V., and Kremers, E. (2017a). Building Parallel FMUs (or Matryoshka Co-Simulations). In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, pages 663–671. Linköping University Electronic Press. 53

[Galtier et al., 2017b] Galtier, V., Ianotto, M., Caujolle, M., Corniglion, R., Tavella, J.-P., Gómez, J. É., Cabrera, J. J. H., Reinbold, V., and Kremers, E. (2017b). Experimenting with Matryoshka Co-Simulation : Building Parallel and Hierarchical FMUs. In *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, pages 663–671. Linköping University Electronic Press. 60

[Garau et al., 2017] Garau, M., Celli, G., Ghiani, E., Pilo, F., and Corti, S. (2017). Evaluation of smart grid communication technologies with a co-simulation platform. *IEEE Wireless Communications*, 24(2) :42–49. 45

[Garro and Falcone, 2015] Garro, A. and Falcone, A. (2015). On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation : DEVS Integrative M&S Symposium*, DEVS '15, pages 9–16, Alexandria, Virginia. Society for Computer Simulation International. 50

[Gomes et al., 2018a] Gomes, C., Meyers, B., Denil, J., Thule, C., Lausdahl, K., Vangheluwe, H., and De Meulenaere, P. (2018a). Semantic adaptation for FMI co-simulation with hierarchical simulators. *SIMULATION*, page 003754971875977. 130

[Gomes et al., 2018b] Gomes, C., Thule, C., Deantoni, J., Larsen, P. G., and Vangheluwe, H. (2018b). Co-simulation : The past, future, and open challenges. In Margaria, T. and Steffen, B., editors, *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems*, pages 504–520, Cham. Springer International Publishing. 37

[Gomes et al., 2018c] Gomes, C., Thule, C., Larsen, P. G., and Vangheluwe, H. (2018c). Co-Simulation : A Survey. *ACM Computing Surveys*, 51(3) :49 :1–49 :33. 13, 39, 44, 45, 50

[Guermazi, 2017] Guermazi, S. (2017). *Model-driven co-simulation of Cyber-Physical Systems*. PhD thesis. 39, 50

[Guermazi et al., 2015] Guermazi, S., Tatibouet, J., Cuccuru, A., Dhouib, S., Gérard, S., and Seidewitz, E. (2015). Executable modeling with fUML and alf in papyrus : Tooling and experiments. In *EXE@MoDELS*. 52

[Hardebolle, 2008] Hardebolle, C. (2008). *Composition of Models for the Multi-Paradigm Modeling of the Behavior of Systems*. Theses, Université Paris Sud - Paris XI. 18

[Hardebolle and Boulanger, 2009] Hardebolle, C. and Boulanger, F. (November/December 2009). Exploring Multi-Paradigm Modeling Techniques. *SIMULATION : Transactions of The Society for Modeling and Simulation International*, 85(11/12) :688–708. 13, 19

[Henzinger and Sifakis, 2006] Henzinger, T. A. and Sifakis, J. (2006). The Embedded Systems Design Challenge. In Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J. M., Mattern, F., Mitchell, J. C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M. Y., Weikum, G., Misra, J., Nipkow, T., and Sekerinski, E., editors, *FM 2006 : Formal Methods*, volume 4085, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg. 19

[Hermann et al., 2016] Hermann, M., Pentek, T., and Otto, B. (2016). Design Principles for Industrie 4.0 Scenarios. In *Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS)*, HICSS '16, pages 3928–3937, Washington, DC, USA. IEEE Computer Society. 5, 15

[Hopkinson et al., 2006] Hopkinson, K., Xiaoru Wang, 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. 45

[IEEE, 2001] IEEE (2001). IEEE standard for modeling and simulation (M S) high level architecture (HLA) - HLA object model template (OMT) specification. *IEEE Std 1516.2-2000*, pages 1–140. 46

[Jézéquel et al., 2012] Jézéquel, J.-M., Combemale, B., and Vojtisek, D. (2012). *Ingénierie dirigée par les modèles des concepts à la pratique*. Ellipses, Paris. OCLC : 801059275. 12, 22, 23, 31

[Kelly and Tolvanen, 2008] Kelly, S. and Tolvanen, J.-P. (2008). *Domain-Specific Modeling : Enabling Full Code Generation*. Wiley-Interscience : IEEE Computer Society, Hoboken, N.J. OCLC : ocn154798454. 25

[Kleppe et al., 2003] Kleppe, A. G., Warmer, J. B., and Bast, W. (2003). *MDA Explained : The Model Driven Architecture : Practice and Promise*. The Addison-Wesley Object Technology Series. Addison-Wesley, Boston. 26

[Kuhn and Hacking, 2012] Kuhn, T. and Hacking, I. (2012). *The Structure of Scientific Revolutions : 50th Anniversary Edition*. University of Chicago Press. 18

[Kühne, 2006] Kühne, T. (2006). Matters of (Meta-) Modeling. *Softw Syst Model*, 5(4) :369–385. 12

[Le Goaer et al., 2018] Le Goaer, O., Cariou, E., and Brunschwig, L. (2018). Xmodeling studio : Un outil pour définir des DSL exécutables. In *7ème Conférence En IngénieriE Du Logiciel (CIEL 2018)*, Grenoble, France. 25

[Lee and Sangiovanni-Vincentelli, 1996] Lee, E. and Sangiovanni-Vincentelli, A. (1996). Comparing models of computation. In *Proceedings of International Conference on Computer Aided Design*, pages 234–241, San Jose, CA, USA. IEEE Comput. Soc. Press. 14

[Lee, 2008] Lee, E. A. (2008). Cyber Physical Systems : Design Challenges. page 10. 15

[Li et al., 2011] Li, W., Monti, A., Luo, M., and Dougal, R. A. (2011). VPNET : A co-simulation framework for analyzing communication channel effects on power systems. In *2011 IEEE Electric Ship Technologies Symposium*, pages 143–149. 45

[Mellor et al., 2004] Mellor, S. J., Scott, K., Uhl, A., and Weise, D. (2004). *MDA Distilled : Principles of Model-Driven Architecture*. Addison-Wesley Object Technology Series. Addison-Wesley, Boston. 25

[Mens and Van Gorp, 2006] Mens, T. and Van Gorp, P. (2006). A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152 :125–142. 27

[Mets et al., 2011] Mets, K., Verschueren, T., Develder, C., Vandoorn, T. L., 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, Kyoto, Japan. IEEE. 45, 53

[Mosterman and Zander, 2016] Mosterman, P. J. and Zander, J. (2016). Industry 4.0 as a Cyber-Physical System study. *Software & Systems Modeling*, 15(1) :17–29. 5, 15

[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. 45

[Oudart, 2017] Oudart, D. (2017). Model driven design of IT systems for smart grids. In Burgueño, L., Corley, J., Bencomo, N., Clarke, P. J., Collet, P., Famelis, M., Ghosh, S., Gogolla, M., Greenyer, J., Guerra, E., Kokaly, S., Pierantonio, A., Rubin, J., and Ruscio, D. D., editors, *Proceedings of MODELS 2017 Satellite Event : Workshops (ModComp, ME, EXE, COMMitMDE, MRT, MULTI, GEMOC, MoDeVVa, MDETools, FlexMDE, MDEbug), Posters, Doctoral Symposium, Educator Symposium, ACM Student Research Competition, and Tools and Demonstrations Co-Located with ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017), Austin, TX, USA, September, 17, 2017*, volume 2019 of *CEUR Workshop Proceedings*, pages 484–486. CEUR-WS.org. xiii

[Oudart et al., 2018] Oudart, D., Cantenot, J., Boulanger, F., and Chabridon, S. (2018). Démarche de conception d'un réseau électrique intelligent et de son système d'information par cosimulation. In *CIEL 2018 : 7ème Conférence En IngénieriE Du Logiciel*, pages 9–1. CNRS. xiii

[Oudart et al., 2019] Oudart, D., Cantenot, J., Boulanger, F., and Chabridon, S. (2019). An Approach to Design Smart Grids and Their IT System by Cosimulation :. In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development*, pages 370–377, Prague, Czech Republic. SCITEPRESS - Science and Technology Publications. xiii

[Palensky et al., 2017] Palensky, P., Van Der Meer, A. A., Lopez, C. D., Joseph, A., and Pan, K. (2017). Cosimulation of Intelligent Power Systems : Fundamentals, Software Architecture, Numerics, and Coupling. *IEEE Industrial Electronics Magazine*, 11(1) :34–50. 39, 50

[Paris, 2019] Paris, T. (2019). *Complex System Modeling by Composition : A Hierarchical Approach for Heterogeneous Components Co-Simulation*. Theses, Université de Lorraine. 16, 34

[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. 16, 61

[Rivière et al., 2013] Rivière, P., Bizingre, J., and Paumier, J. (2013). *Les référentiels du système d'information : Données de référence et architectures d'entreprise*. Management des systèmes d'information. Dunod. OCLC : 1003593644. 29

[Sabonnadière and Hadjsaïd, 2012] Sabonnadière, J.-C. and Hadjsaïd, N. (2012). *SmartGrids : les réseaux électriques intelligents*. Hermès Science publications-Lavoisier, Paris. OCLC : 795500901. 2

[Schutte et al., 2011] Schutte, S., Scherfke, S., and Troschel, M. (2011). Mosaik : A framework for modular simulation of active components in Smart Grids. In *2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS)*, pages 55–60, Brussels, Belgium. IEEE. 34

[Seghiri, 2016] Seghiri, R. (2016). *Modélisation et Simulation d'une Architecture d'entreprise-Application Aux Smart Grids*. PhD thesis, Université Paris-Saclay. 51, 52

[Seghiri et al., 2016] Seghiri, R., Boulanger, F., Lecocq, C., and Godefroy, V. (2016). An executable model driven framework for enterprise architecture application to the Smart Grids context. In *System Sciences (HICSS), 2016 49th Hawaii International Conference On*, pages 4546–4555. IEEE. 29, 52

[Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5) :26–32. 12

[Seidewitz and Tatibouet, 2015] Seidewitz, E. and Tatibouet, J. (2015). Tool Paper : Combining Alf and UML in Modeling Tools - An Example with Papyrus -. In Brucker, A. D., Egea, M., Gogolla, M., and Tuong, F., editors, *OCL@MoDELS*, volume 1512 of *CEUR Workshop Proceedings*, pages 105–119. CEUR-WS.org. 52

[Sendall and Kozaczynski, 2003] Sendall, S. and Kozaczynski, W. (2003). Model transformation : The heart and soul of model-driven software development. *IEEE Softw.*, 20(5) :42–45. 26

[Shum et al., 2018] Shum, C., Lau, W.-H., Mao, T., Chung, H. S.-H., Tsang, K.-F., Tse, N. C.-F., and Lai, L. L. (2018). Co-Simulation of Distributed Smart Grid Software Using Direct-Execution Simulation. *IEEE Access*, 6 :20531–20544. 45

[Siegel, 2014] Siegel, J. (2014). MDA guide, revision 2.0. 30

[Steen et al., 2004] Steen, M. W. A., Akehurst, D. H., ter Doest, H. W. L., and Lankhorst, M. M. (2004). Supporting viewpoint-oriented enterprise architecture. In *Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004.*, pages 201–211. 29

[Suri et al., 2017] Suri, K., Cuccuru, A., Cadavid, J., Gerard, S., Gaaloul, W., and Tata, S. (2017). Model-based Development of Modular Complex Systems for Accomplishing System Integration for Industry 4.0. In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development - Volume 1 : MODELSWARD,*, pages 487–495. ScitePress. 5

[Tavella et al., 2016] Tavella, J.-P., Caujolle, M., Vialle, S., Dad, C., Tan, C., Plessis, G., Schumann, M., Cuccuru, A., and Revol, S. (2016). Toward an Accurate and Fast Hybrid Multi-Simulation with the FMI-CS Standard. In *21 St International Conference on : Emerging Technologies and Factory Automation (ETFA-2016)*, Berlin, Germany. 50

[URBA-EA, 2010] URBA-EA, C. (2010). *Urbanisme Des SI et Gouvernance : Bonnes Pratiques de l'architecture d'entreprise*. Management Des Systèmes d'information. Dunod. 28, 29, 30

[Uslar et al., 2019] Uslar, M., Rohjans, S., Neureiter, C., Pröstl Andrén, F., Velasquez, J., Steinbrink, C., Efthymiou, V., Migliavacca, G., Horsmanheimo, S., Brunner, H., and Strasser, T. (2019). Applying the Smart Grid Architecture Model for Designing and Validating System-of-Systems in the Power and Energy Domain : A European Perspective. *Energies*, 12(2) :258. 32, 34

[Vangheluwe et al., 2002] Vangheluwe, H., DE LARA, J., and MOSTERMAN, P. (2002). An introduction to multi-paradigm modelling and simulation. In *Proceedings of the AIS'2002 Conference (AI, Simulation and Planning in High Autonomy Systems), April 2002, Lisboa, Portugal / f. Barros and n. Giambiasi (Eds.)*, pages 9–20. 19

[Vaubourg et al., 2015] Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J.-P., Morais, H., Deneuville, B., and Chilard, O. (2015). Smart Grids Simulation with MECSYCO. In Demazeau, Y., Decker, K. S., Bajo Pérez, J., and de la Prieta, F., editors, *PAAMS'15*, volume 9086 of *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, page 4, Salamanca, Spain. Springer. 50

[Yang et al., 2013] Yang, C.-H., Zhabelova, G., Yang, C.-W., and Vyatkin, V. (2013). Cosimulation Environment for Event-Driven Distributed Controls of Smart Grid. *IEEE Trans. Industrial Informatics*, 9(3) :1423–1435. 38, 76

[Zhang et al., 2014] Zhang, Z., Eyisi, E., Koutsoukos, X., Porter, J., Karsai, G., and Sztipanovits, J. (2014). A co-simulation framework for design of time-triggered automotive cyber physical systems. *Simulation Modelling Practice and Theory*, 43 :16–33. 45, 50

INSTITUT POLYTECHNIQUE DE PARIS

Titre : Application de l'ingénierie dirigée par les modèles à la conception de Smart Grids : Approche par cosimulation avec FMI

Mots clés : Cosimulation, Smart Grid, Ingénierie Dirigée par les Modèles, Functional Mockup Interface

Résumé : Les Smart Grids sont des systèmes cyberphysiques qui interfacent les réseaux électriques avec les technologies de l'information et de la communication afin de les surveiller, d'automatiser la prise de décision et d'équilibrer la production avec la consommation. Nous souhaitons utiliser la simulation pour évaluer et comparer facilement plusieurs solutions avant un déploiement dans un environnement réel. L'objectif de cette thèse est ainsi de proposer des outils et méthodes pour modéliser et simuler un Smart Grid dans un contexte industriel. Nous avons identifié deux problématiques principales : Comment combiner les modèles hétérogènes d'un Smart Grid pour le simuler ? Comment assurer la cohérence entre les modèles produits par différents intervenants lors de la conception d'un Smart Grid ? Pour répondre à ces problématiques, nous proposons une approche de cosimulation, en utilisant la norme Functional Mockup Interface (FMI). Nos deux premières contributions sont la proposition d'une méthode pour permettre l'échange de signaux discrets entre plusieurs FMUs, et d'une extension du logiciel de simulation de télécommunications OMNeT++ implémentant cette méthode, appelée fmi4omnetpp. Une troisième contribution est la réalisation de l'environnement outillé Smart Grid Simulation Framework, qui automatise un certain nombre de tâches répétitives afin d'assurer la cohérence entre différents modèles de simulation. Enfin, une quatrième contribution est la formalisation de la démarche itérative de conception dans laquelle s'inscrit la cosimulation d'un Smart Grid, et la façon d'y intégrer notre environnement Smart Grid Simulation Framework. Pour cela, nous explicitons les différentes étapes de la démarche et le rôle des acteurs de la conception, puis nous présentons son application sur un cas d'étude réel pour lequel nous utilisons Smart Grid Simulation Framework.

Title : Model driven engineering applied to Smart Grids design: Cosimulation with FMI approach

Keywords : Cosimulation, Smart Grid, Model driven engineering, Functional Mockup Interface

Abstract : Smart Grids are cyber-physical systems that interface power grids with information and communication technologies to monitor them, automate decision making and balance production with consumption. We want to use simulation to easily evaluate and compare several solutions before deployment in a real environment. The objective of this thesis is thus to propose tools and methods to model and simulate a Smart Grid in an industrial context. We have identified two main issues: How to combine heterogeneous models of a Smart Grid to simulate it ? How to ensure consistency between the models produced by different stakeholders during the design of a Smart Grid ? To address these issues, we propose a cosimulation approach, using the Functional Mockup Interface (FMI) standard. Our first two contributions are the proposal of a method to allow the exchange of discrete signals between several FMUs, and an extension of the OMNeT++ telecommunications simulation software implementing this method, called fmi4omnetpp. A third contribution is the development of the Smart Grid Simulation Framework tooled environment, which automates a number of repetitive tasks in order to ensure consistency between different simulation models. Finally, a fourth contribution is the formalization of an iterative design approach for the cosimulation of a Smart Grid, and how to integrate our Smart Grid Simulation Framework into it. To do so, we explain the different steps of the approach and the role of the actors involved in the design process, then we present its application to a real case study for which we use our Smart Grid Simulation Framework.

Institut Polytechnique de Paris
91120 Palaiseau, France