Wrapping DEVS de modèles IP dans MECSYCO pour la co-simulation de systèmes cyber-physiques

Benjamin Camus, Julien Vaubourg, Thomas Paris, Yannick Presse, Christine Bourjot, Laurent Ciarletta, Vincent Chevrier

Wrapping DEVS de modèles IP dans MECSYCO pour la co-simulation de systèmes cyber-physiques

Benjamin Camus[*], Julien Vaubourg[*], Thomas Paris[*], Yannick Presse[*], Christine Bourjot[*], Laurent Ciarletta[*] et Vincent Chevrier[*]

[*] Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
prénom.nom@loria.fr

2017

Résumé : La modélisation et simulation des systèmes cyber-physiques, comme les réseaux électriques intelligents, devient une étape préalable incontournable à leur réalisation. Un défi central est alors d'intégrer de manière cohérente les différents domaines d'expertise et les différents points de vue sur le système cible, chacun disposant déjà de ses propres outils et formalismes. Cet article présente l'intérêt de la plateforme de co-simulation MECSYCO, basée sur le wrapping DEVS, en termes d'intégration de l'hétérogénéité des modèles et des simulateurs : formalismes, représentation du temps et des données, logiciels, etc. La plateforme permet également la modularité de la démarche de conception M&S : incrémentalité, remplacement de simulateurs, etc. Nous montrons comment nous avons réalisé l'intégration d'outils de simulation existants de référence dans le domaine des réseaux IP (e.g. NS-3) ou des standards de la simulation (e.g. FMU) dans la plateforme et comment il est possible de les utiliser pour la co-simulation. Nous nous appuyons sur un cas d'usage de co-simulation hybride de bâtiments intelligents pour illustrer nos propositions.
Mots-clés : multi-modélisation, systèmes complexes, réseaux IP, systèmes hybrides, co-simulation, DEVS

Abstract: *The modeling and simulation of cyber-physical systems (as smart-grids) is becoming an essential step of their construction process. One central challenge is to integrate in a consistent way each domain and point of view on the target system, each already having its own formalisms and modeling tools. This article presents the benefits of a DEVS wrapping based co-simulation platform (MECSYCO) in terms of integration of the heterogeneity of models and simulators: formalisms, time and data representations, software, etc. The platform provides modularity in the modeling ans simulaiton process: incrementalaty, simulators, exchanges, etc. We show how we have integrated tools like NS3 and OMNET++/INET (reference tools for IP network modeling), or simulation standard such as FMU in our platform and how they can be used in co-simulations. An hybrid co-simulation use case is presented to illustrate our proposal.*
***Keywords**: multi-modeling, complex system, IP network, hybrid system, co-simulation, DEVS*

1 Introduction

Cet article porte sur la modélisation et la simulation numérique (M&S) de systèmes cyber-physiques. Un système est dit cyber-physique dès lors qu'il met en œuvre des systèmes physiques qui sont contrôlés, supervisés, coordonnés et intégrés par des systèmes informatiques [Rajkumar et al., 2010]. Les réseaux IP font partie intégrante de ce type de système.

En expérimentant de manière rigoureuse sur la version simplifiée d'un système (i.e. un modèle) plutôt que sur un système réel, le processus de M&S s'affranchit des contraintes de coût, de temps et d'éthique, et se positionne alors comme un outil de choix pour l'étude des systèmes cyber-physiques. La démarche de M&S appliquée à l'étude de ces systèmes rencontre plusieurs défis scientifiques spécifiques. La plupart des questionnements sur ces systèmes nécessitent de prendre en compte plusieurs points de vue simultanément. Il faut alors considérer des phénomènes évoluant à des échelles (temporelles et spatiales) et des niveaux de résolution (microscopique à macroscopique) différents. De plus, l'expertise nécessaire pour décrire le système vient en général de plusieurs domaines scientifiques, chaque domaine ayant ses propres modèles et outils de M&S (formalismes et logiciels). **Les défis sont alors de concilier ces hétérogénéités, et d'intégrer l'existant de chaque domaine, en particulier les réseaux IP, tout en restant dans le cadre rigoureux de la démarche de M&S.**

La co-simulation constitue une stratégie très prometteuse pour répondre à ces défis. Elle consiste à effectuer une simulation en réutilisant des modèles déjà implémentés dans des logiciels de simulation différents et à gérer les échanges de données entre ces simulateurs pour faire interagir leurs modèles. La co-simulation permet alors à chaque spécialiste de continuer à utiliser les outils de M&S qui sont populaires dans son domaine. La co-simulation a l'avantage de pouvoir être distribuée sur plusieurs machines, afin de permettre son passage à l'échelle. La co-simulation fait cependant face à de nombreux problèmes causés par l'hétérogénéité des modèles et des outils qu'elle doit faire interagir.

Ce papier est organisé comme suit. La Section 2 détaille les différents défis de la co-simulation de systèmes cyber-physiques. La Section 3 présente comment le formalisme DEVS – et plus précisément la technique de wrapping DEVS – offre une solution essentielle à ces défis. La Section 4 présente notre première contribution : la plateforme de co-simulation MECSYCO (Multi-agent Environment for Complex SYstem CO-simulation) dédiée au wrapping DEVS de logiciels de simulation pré-existants. Cette plateforme permet une exécution parallèle, décentralisée et distribuable. La Section 5 propose une stratégie générique de wrapping DEVS pour les simulateurs de réseaux IP et sa mise en œuvre dans MECSYCO. Enfin, la Section 6 montre comment nous validons notre proposition avec le cas d'utilisation d'une M&S de bâtiments intelligents.

2 Les défis de la co-simulation

La co-simulation de systèmes cyber-physiques doit faire interagir plusieurs simulateurs existants. les problèmes rencontrés sont alors que ces simulateurs correspondent à des logiciels de simulation différents qui implémentent des formalismes de modélisation différents. Les défis sont alors d'intégrer ces formalismes et de gérer l'interopérabilité des logiciels de

simulation.

2.1 Intégration multi-formalisme

À cause de son hétérogénéité, plusieurs formalismes sont requis pour décrire un système cyber-physique [Vangheluwe et al., 2002]. Des équations différentielles ou algébriques peuvent par exemple être utilisées pour décrire l'évolution continue du système, alors que des automates à états finis ou des modèles événementiels peuvent servir à décrire la partie discrète du système. En conséquence, des modèles formels différents, discrets et continus, peuvent interagir et co-évoluer au sein d'une co-simulation. Au niveau de l'exécution de la simulation, cette hétérogénéité formelle implique une gestion de la politique d'exécution différente : pas de temps cyclique et acyclique ou événementielle. Un cadre formel rigoureux est alors nécessaire pour intégrer ces différents formalismes et définir le comportement de la simulation de manière univoque [Cellier, 1979]. Parmi les solutions existantes décrites par [Vangheluwe et al., 2002], nous retiendrons le recours à **un formalisme hybride** comme DEV&DESS [Praehofer, 1991] ou HFSS [Barros, 2003] qui décrit explicitement comment des modèles discrets et continus interagissent et co-évoluent. Ces formalismes hybrides synthétisent un ensemble de techniques utilisées dans le domaine de la M&S hybride. Ces techniques incluent principalement : 1) l'intégration d'entrées discrètes pendant l'évolution continue du système, et 2) la génération pendant la simulation de deux types d'événements discrets à partir de l'évolution continue du système : les événements temporels (time-events) et les événements d'état (state-events) [Cellier, 1979]. Alors que les premiers correspondent à des événements planifiés dans le temps simulé, les seconds correspondent à des événements dont l'occurrence est directement conditionnée par l'état continu du système (en général, lorsqu'une variable continue dépasse une certaine valeur). Du point de vue de la simulation, la difficulté est d'intégrer de manière générique cette logique discrète pendant la résolution numérique du système continu, ce dernier ayant pour objectif de trouver le meilleur compromis entre la précision des résultats de simulation et les performances de la simulation [Esquembre and Christian, 2007]. Notamment, la détection et la localisation précise des événements d'états est une difficulté bien connue dans le domaine des simulations hybrides [Mosterman, 2007].

2.2 Interopérabilité des logiciels de simulation

Pour implémenter et simuler un modèle, chaque domaine d'application dispose d'outils de simulation spécifiques. Ces outils, qui ne sont en général pas conçus pour communiquer ensemble, peuvent être écrits dans des langages de programmation différents (Java, C++, Python, Fortran, etc.), être compatibles avec des systèmes d'exploitation différents (GNU/Linux, Windows, Mac OS, en 32 ou 64 bits, etc.), et être disponibles sur un équipement spécifique (e.g. sur une machine disposant d'une licence d'utilisation de l'outil).

La co-simulation doit gérer l'interopérabilité de ces outils de simulation, c'est à dire gérer les échanges de données utilisables entre les simulateurs et la synchronisation de ces derniers. Cette interopérabilité peut être assurée de manière ad-hoc, en modifiant directement les outils de simulation pour les rendre interopérables, ou alors en utilisant un intergiciel. Cette dernière solution a l'avantage d'être plus flexible, car les outils de simulation

n'ont pas à être interopérables entre eux, mais doivent uniquement être interopérables avec l'intergiciel. Contrairement à une approche ad-hoc, l'intergiciel permet l'ajout, la suppression ou le changement d'un outil de simulation au multi-modèle numérique sans impacter les outils de simulation déjà intégrés.

2.3 Synthèse

Pour résumer, la mise en place d'une co-simulation requiert de résoudre un ensemble de problèmes spécifiques au niveau des formalismes et des logiciels de simulation. Les solutions qui doivent être développées sont directement liées à l'hétérogénéité présente à chacun de ces niveaux. De plus, afin de garantir l'expérimentation sur les modèles (point fondamental dans le processus de M&S), la co-simulation a besoin d'être modulaire : il faut pouvoir ajouter, enlever ou modifier des modèles et des logiciels de simulation et leurs interconnexions sans avoir à redéfinir et ré-implémenter l'ensemble de la co-simulation.

Pour satisfaire ce pré-requis, les solutions ad-hoc doivent être évitées et un cadre générique et rigoureux est nécessaire. Dans la suite, nous détaillons comment le formalisme DEVS fournit ce cadre générique.

3 DEVS comme formalisme pivot pour l'intégration de l'hétérogénéité

3.1 Le formalisme DEVS

Le Discrete EVent System specification (DEVS) est un formalisme événementiel de M&S créé par Bernard P. Zeigler dans les années 1970 [Zeigler et al., 2000]. Une caractéristique importante de ce formalisme est son universalité qui le positionne comme un formalisme pivot pour l'intégration de formalismes hétérogènes [Vangheluwe, 2000]. En effet, outre le fait qu'il soit universel pour décrire les modèles événementiels [Zeigler et al., 2000], DEVS est également à la base d'un long travail d'intégration de formalismes. Ainsi, au fil des années, il a été montré que DEVS pouvait intégrer les formalismes des Réseaux de Petri [Jacques and Wainer, 2002], des statecharts [Borland and Vangheluwe, 2003], des automates à états finis temporels [Dacharry and Giambiasi, 2005], des automates cellulaires [Wainer and Giambiasi, 2001], et des équations différentielles [Zeigler et al., 2000, Vangheluwe, 2000, Quesnel et al., 2005, Cellier et al., 2008]. De plus, plusieurs formalisations du paradigme multi-agent en DEVS ont été proposées [Duboz et al., 2006, Bisgambiglia and Franceschini, 2013, Uhrmacher and Schattenberg, 1998]. Enfin, il a été montré que DEVS pouvait également intégrer le formalisme hybride DEV&DESS [Zeigler, 2006] et donc faire interagir des modèles discrets et continus.

Comme résumé dans [Quesnel et al., 2005], l'intégration d'un formalisme dans DEVS peut se faire par mapping ou par wrapping. Alors que le mapping consiste à établir une équivalence entre le formalisme du modèle et DEVS, le wrapping [Kim and Kim, 1998] consiste à faire le pont entre le simulateur abstrait du modèle et celui de DEVS. Le wrapping a alors l'avantage de permettre de réutiliser des modèles existants déjà implémentés [Mittal et al., 2015].

3.2 Positionnement

DEVS apporte une solution de référence en ce qui concerne l'intégration de l'hétérogénéité formelle. Nous proposons alors de l'utiliser comme formalisme pivot dans le contexte de la co-simulation des systèmes cyber-physiques. Plus précisément, afin de répondre à la fois aux besoins d'intégration formelle et de gestion de l'interopérabilité logicielle, nous proposons la procédure de co-simulation suivante :

1. intégrer séparément chaque modèle en DEVS sans avoir à le ré-implémenter grâce à la technique du wrapping ;
2. faire interagir l'ensemble des modèles au sein d'un modèle couplé DEVS ;
3. simuler le modèle couplé DEVS en utilisant le protocole de simulation DEVS pour exécuter la co-simulation de manière unifiée.

Dans la section suivante, nous détaillons la plateforme MECSYCO qui implémente cette proposition. MECYSCO joue alors le rôle d'intergiciel de co-simulation.

4 La plateforme MECSYCO

4.1 Un environnement multi-agent de M&S

L'intergiciel MECSYCO [Camus et al., 2015a, Camus, 2015] que nous proposons est une plateforme de wrapping DEVS qui s'appuie sur l'universalité de DEVS pour la co-simulation des systèmes cyber-physiques. Il a été montré dans des travaux précédents que la plateforme supporte la modélisation multi-niveau [Camus et al., 2015b]. MECSYCO est actuellement utilisé pour la M&S des réseaux électriques intelligents (smart grids) en partenariat entre le LORIA/Inria et EDF R&D [Vaubourg et al., 2015].

MECSYCO est basé sur le paradigme AA4MM (Agent & Artifacts for Multi-Modeling) [Siebert, 2011] (lui-même issu d'une idée originale de [Bonneaud, 2008]) qui considère une co-simulation hétérogène comme un système multi-agent. Dans cette perspective, chaque couple modèle/simulateur correspond à un agent et les échanges de données entre les simulateurs correspondent aux interactions entre les agents. La co-simulation du système correspond alors à la dynamique des interactions entre les agents. L'autonomie des agents permet, grâce à l'utilisation de wrappeurs, d'encapsuler des systèmes hérités (legacy system) [Jennings, 2001]. L'originalité par rapport à d'autres approches de multi-modélisation multi-agent, est d'envisager les interactions de manière indirecte grâce à des entités passives de calcul appelées artéfacts [Ricci et al., 2007]. En suivant ce paradigme multi-agent des concepts jusqu'à leurs implémentations, MECSYCO garantit une architecture de co-simulation extensible (i.e. des fonctionnalités peuvent être facilement ajoutées, comme un système d'observation) décentralisée et distribuable. MECSYCO implémente les concepts de AA4MM en suivant le protocole de simulation DEVS pour coordonner les exécutions des simulateurs et générer les interactions entre les modèles.

(a) Modèle. (b) M-agent. (c) Artéfact de couplage. (d) Artéfact de modèle.

FIGURE 1 – Symboles des concepts de MECSYCO.

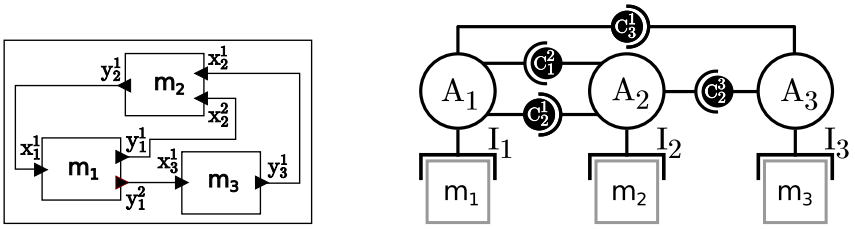
4.2 Les concepts de MECSYCO

Pour décrire une co-simulation, MECSYCO repose sur quatre concepts, chacun d'entre eux étant associé à une notation graphique :

- un **modèle** m_i (symbole à la Figure 1a) déjà implémenté dans un logiciel de simulation. MECSYCO considère ce modèle comme étant doté d'ensembles X_i de ports d'entrée et Y_i de ports de sortie ;
- un **m-agent** $\mathcal{A}_i$ (symbole à la Figure 1b) est associé à chaque modèle m_i. $\mathcal{A}_i$ a pour rôle de gérer l'exécution de m_i, et les interactions de celui-ci avec les autres modèles. Un m-agent joue alors le rôle du simulateur abstrait parallèle de son modèle ;
- chaque **artéfact de modèle** $\mathcal{I}_i$ est un wrapper DEVS pour m_i -i.e. $\mathcal{A}_i$ se sert de $\mathcal{I}_i$ pour contrôler m_i comme un modèle atomique DEVS (symbole à la Figure 1d). **C'est alors dans les artéfacts de modèle que les modèles sont intégrés en DEVS suivant notre proposition** ;
- chaque interaction (i.e. échange d'événements externes) d'un modèle m_i vers un modèle m_j est instanciée par un **artéfact de couplage** $\mathcal{C}_j^i$ (symbole à la Figure 1c) gérant les interactions de $\mathcal{A}_i$ vers $\mathcal{A}_j$. Un artéfact de couplage $\mathcal{C}_j^i$ est orienté : il gère les interactions de $\mathcal{A}_i$ vers $\mathcal{A}_j$. $\mathcal{C}_j^i$ est alors pour $\mathcal{A}_i$ un artéfact de couplage de sortie, et un artéfact de couplage d'entrée pour $\mathcal{A}_j$. $\mathcal{C}_j^i$ fonctionne comme une boîte aux lettres : l'artéfact contient un tampon d'événements dans lequel $\mathcal{A}_i$ peut déposer ses événements de sortie, événements que $\mathcal{A}_j$ pourra ensuite récupérer comme entrées de son modèle. Les artéfacts de couplage peuvent transformer les données échangées entre les modèles en appliquant des opérations de transformation. Ces opérations permettent alors de passer d'une représentation du système à une autre (e.g. convertir des kilomètres en mètres, des heures en minutes, ou faire des opérations d'agrégation/désagrégation).

En accord avec le paradigme multi-agent de MECSYCO, chaque m-agent n'a qu'une connaissance locale des interconnexions entre les modèles de la co-simulation. Cette caractéristique permet à MECSYCO d'avoir une représentation modulaire et décentralisée d'un multi-modèle. L'ensemble IC des couplages internes du modèle couplé DEVS décrivant la co-simulation est alors éclaté de manière à ce que chaque m-agent $\mathcal{A}_i$ sache :

- À quel artéfact de couplage de sortie il doit envoyer les données de simulation produites par chaque port de sortie de m_i. $\mathcal{A}_i$ connaît alors l'ensemble des liens



(a) modèle couplé DEVS. (b) co-simulation MECSYCO

FIGURE 2 – Schémas fonctionnels de la co-simulation MECSYCO de la Table 1 et du modèle couplé DEVS équivalent.

TABLE 1 – Co-simulation MECSYCO du modèle couplé DEVS de la Figure 2a

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

de sortie OUT_i comprenant les couples (n, j) reliant le port de sortie $y_i^n \in Y_i$ avec l'artéfact de couplage de sortie $\mathcal{C}_j^i$.

- Auprès de quel artéfact de couplage d'entrée il doit récupérer les données de simulation pour chacun des ports d'entrée de m_i. $\mathcal{A}_i$ connaît alors l'ensemble des liens d'entrée IN_i comprenant les couples (j, k) reliant l'artéfact de couplage d'entrée $\mathcal{C}_i^j$ avec le port d'entrée $x_i^k \in X_i$.

Les connexions entre les ports de sortie d'un modèle m_i et les ports d'entrée d'un modèle m_j sont effectuées par l'artéfact de couplage $\mathcal{C}_j^i$. Cet artéfact contient l'ensemble L_j^i contenant les triplets $(n, k, o_{j,k}^{i,n})$ reliant le port de sortie $y_i^n \in Y_i$ avec le port d'entrée $x_j^k \in X_j$ en utilisant l'opération de transformation $o_{j,k}^{i,n}$. L'opération de transformation par défaut correspond à l'identité *id*. À titre d'exemple, la Table 1 et la Figure 2b montrent comment le modèle couplé DEVS de la Figure 2a est décrit de manière décentralisée et distribuable grâce à MECSYCO.

4.3 Spécifications opérationnelles

Le comportement de chaque m-agent correspond à celui du simulateur abstrait conservatif DEVS qui est lui-même basé sur l'algorithme de Chandy-Misra-Bryant (CMB) [Chandy and Misra, 1979, Bryant, 1979]. Cet algorithme est prouvé comme ne pouvant pas entrainer de situation d'interblocage (deadlock) et comme respectant la contrainte de causalité [Zeigler et al., 2000] – i.e. s'assurant que « *l'exécution d'une simulation sur*

un ordinateur parallèle produira exactement les mêmes résultats qu'une exécution sur un ordinateur séquentiel » [Fujimoto, 2001].

Chaque m-agent A_i partage son EOT_i (*Earliest Output Time*) avec son environnement. EOT_i correspond au temps (simulé) avant lequel A_i assure que son modèle ne génèrera pas de nouveaux événements. Chaque artéfact de couplage C_j^i connecté à un port de sortie de A_i va recevoir EOT_i comme son temps de lien noté LT_i (correspondant au temps simulé jusqu'auquel A_i a simulé le lien de m_i à m_j) [Chandy and Misra, 1979]. Chaque m-agent A_i utilise les temps de lien de chacun des artéfacts de couplage connectés à un des ses ports d'entrée pour calculer son EIT_i (*Earliest Input Time*) correspondant au temps simulé avant lequel m_i ne recevra pas de nouvel événement. Il est égal au minimum des temps de lien des artéfacts de couplage connectés aux ports d'entrée de A_i. Pour chaque m-agent A_i, tous les événements (interne ou externe) dont le temps est inférieur ou égal à EIT_i sont dits valides pour être exécutés. Chaque m-agent ne doit exécuter que les événements valides, dans l'ordre croissant de leur temps d'occurrence, pour assurer que la contrainte de causalité est respectée. Chaque EOT_i est donné par la fonction suivante : $\texttt{Lookahead}_i$ function :

$$Lookahead_i() = min\{nt_i, \mathsf{EIT}_i + D_i, t_{in_i+D_i}\} \tag{1}$$

avec nt_i le temps du prochain événement interne de m_i, t_{in_i} le temps du prochain événement externe à intégrer venant des ports d'entrée de A_i, et D_i ($D_i > 0$) le délai minimum de propagation des événements de m_i. Ce délai minimum de propagation correspond à la durée (simulée) minimum avant laquelle un événement externe ne peut pas générer de nouvel événement interne dans le modèle m_i. D_i doit être déterminé pour chaque modèle m_i de la co-simulation. Le comportement permettant de simuler les modèles est formalisé par l'algorithme 1. Cet algorithme est basé sur les spécifications des artéfacts de couplage. Pour chaque artéfact de couplage C_j^i, six fonctions sont mises à disposition des agents A_i et A_j :

- `post(`e_{out}^n`,` t_i`)` permet de stocker l'événement e_{out}^n dans la pile d'événement de l'artéfact $\mathcal{C}_j^i$, il est en outre modifié selon l'opération associée au couplage. e_{out}^n est généré au temps t_i, il provient du port de sortie y_i^n associé à $\mathcal{C}_j^i$.
- `getEarliestEvent(`k`)` renvoie le prochain événement externe devant arriver au $k^{ième}$ port d'entrée de m_j, x_j^k.
- `getEarliestEventTime(`k`)` retourne la date du prochain événement externe devant arriver pour x_j^k.
- `removeEarliestEvent(`k`)` retire l'événement dont la date d'occurrence est la plus basse de la pile de l'artéfact de couplage associé à x_j^k.
- `setLinkTime(`t_i`)` met à jour la valeur LT_j^i avec t_i.
- `getLinkTime()` retourne LT_j^i.

Dans cet algorithme, chaque m-agent $\mathcal{A}_i$ peut manipuler m_i comme un modèle atomique DEVS en utilisant les primitives logicielles du protocole de simulation DEVS proposées par son artéfact de modèle $\mathcal{I}_i$. Ces fonctions, qui sont listées ci-dessous, doivent donc être définies pour chaque logiciel de simulation afin de wrapper les modèles en DEVS :

```
1  nt_i ← I_i.getNextEventTime()
2  t_in_i ← +∞
3  EOT_i ← 0
4  EIT_i ← 0
   /* Tant que la fin de la simulation n'est pas atteinte */
5  while (¬endOfSimulation) do
6      EIT_i ← +∞
7      t_in_i ← +∞
8      forall (j, k) ∈ IN_i do
           /* Calcul de EIT_i */
9          if C_i^j.getLinkTime() < EIT_i then
10             EIT_i ← C_i^j.getLinkTime()
           /* Prendre le prochain événement externe */
11         if C_i^j.getEarliestEventTime(k) < t_in_i then
12             t_in_i ← C_i^j.getEarliestEventTime(k)
13             ein_i ← C_i^j.getEarliestEvent(k)
               /* Sauvegarde du port recevant le prochain événement, */
14             p ← k
               /* et de l'artefact contenant le prochain événement. */
15             c ← j
       /* Calcul de EOT_i + MAJ des artefacts de couplage de sortie */
16     if EOT_i ≠ Lookahead_i(nt_i, EIT_i, t_in_i) then
17         EOT_i ← Lookahead_i(nt_i, EIT_i, t_in_i)
18         ∀(k, j) ∈ OUT_i : C_j^i.setLinkTime(EOT_i)
       /* Calcul de l'événement valide le plus imminent */
       /* S'il s'agit d'un événement interne */
19     if (nt_i ≤ t_in_i) and (nt_i ≤ EIT_i) and (nt_i ≤ Z) then
20         I_i.processInternalEvent(nt_i)
           /* Envoi des événements externes résultants */
21         forall (k, j) ∈ OUT_i do
22             eout_i^k ← I_i.getOutputEvent(y_i^k)
23             if eout_i^k ≠ ∅ then
24                 C_j^i.post(eout_i^k, nt_i)
25         nt_i ← I_i.getNextInternalEventTime()
       /* Sinon, s'il s'agit d'un événement externe */
26     else if (t_in_i < nt_i) and (t_in_i ≤ EIT_i) and (t_in_i ≤ Z) then
           /* Exécution de l'événement externe */
27         I_i.processExternalEvent(ein_i, t_in_i, x_i^p)
28         C_i^c.removeEarliestEvent(p)
29         nt_i ← I_i.getNextInternalEventTime()
```

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

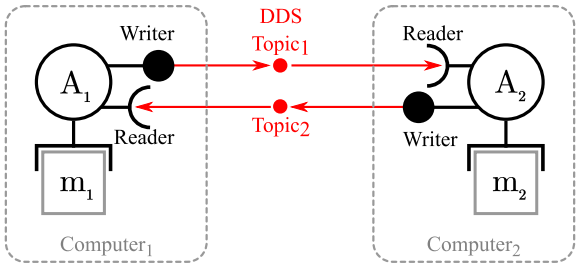


FIGURE 3 – Distribution d'une co-simulation MECSYCO.

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

4.4 Implémentation

MECSYCO est actuellement implémenté en Java (disponible sur `http://mecsyco.com`) et en C++. Pour assurer l'interopérabilité de ces deux implémentations et permettre la distribution des co-simulations, MECSYCO utilise le format JSON et l'implémentation OpenSplice du standard DDS (Data Distribution Service) de l'OMG. Avec OpenSplice, les artéfacts de couplage sont séparés en deux parties, lecture et écriture, pour distribuer la co-simulation. DDS étant basé sur le patron de communication publish-subscribe, les parties lecture et écriture des artéfacts de couplage jouent respectivement le rôle de publieur et de souscripteur (voir Figure 3). L'implémentation des concepts de MECSYCO suit une programmation orientée objet (montrée sur la Figure 4). Cette implémentation suit notre paradigme multi-agent car chaque concept de MECSYCO correspond à une classe d'objet, et chaque m-agent correspond à un thread. Nous conservons alors les avantages de notre paradigme : notre architecture logicielle est composée d'un ensemble modulaire de briques logicielles permettant une co-simulation parallèle, décentralisée et distribuable.

4.5 Wrapping DEVS de logiciel de simulation

Nous avons déjà développé dans MECSYCO des wrappeurs DEVS à la fois pour des modèles discrets [Camus et al., 2015b] et des modèles continus [Camus et al., 2016]. Nous avons notamment intégré à notre plateforme le simulateur multi-agent à pas de temps discrets NetLogo [Wilensky, 1999], et le standard d'export de modèles continus Functional

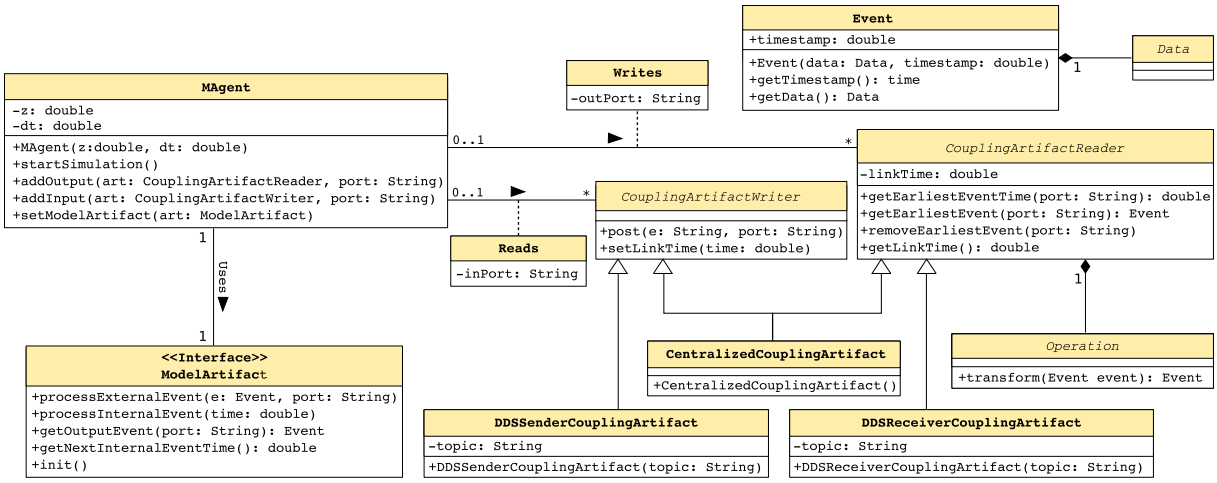


FIGURE 4 – Schéma UML de l'architecture logicielle de MECSYCO.

Mock-up Interface (FMI) [Blochwitz et al., 2012]. Nous sommes alors en mesure d'intégrer les nombreux outils de modélisation compatibles avec le standard FMI (e.g. Dymola, OpenModelica, Matlab/Simulink). Dans cet article, nous nous focalisons sur le wrapping DEVS des outils de simulation de réseaux IP comme NS-3 ou OMNeT++/INET. Ces outils sont basés sur des paradigmes de modélisation événementielle proches de DEVS. D'un point de vue formel, le wrapping de ces outils en DEVS est alors assez direct. Cependant, ces outils décrivent en général des systèmes fermés à un niveau de détail très élevé. D'un point de vue pratique, le wrapping DEVS de ces outils est donc non trivial, et doit être alors spécifié clairement pour rester dans le cadre rigoureux de la démarche de M&S.

5 Wrapping DEVS de simulateurs de réseaux IP

5.1 Hypothèses et objectifs

L'objectif de ce wrapping est d'intégrer des modèles IP à des co-simulations DEVS, de façon à ce qu'ils soient en capacité de représenter le transport de données applicatives, qui sont produites par d'autres modèles. Cet objectif correspond à un mode d'intégration particulier, qu'on qualifie de « structurel » [Vaubourg et al., 2016]. Dans le cadre d'un couplage structurel entre deux modèles d'une même co-simulation, chacun des modèles représente un même système, au même instant, mais selon des points de vue différents. Ainsi, un même appareil pourra être représenté en sa qualité d'équipement de réseau IP dans un modèle IP, alors qu'il sera représenté d'un point de vue applicatif (i.e. en tant que générateur ou consommateur de données applicatives) par un modèle issu d'un autre domaine d'expertise (e.g. modèle physique à base d'équations différentielles).

L'utilisation de wrappeurs DEVS nous permet d'éviter d'utiliser des solutions ad-hoc. Il est donc possible de passer d'un modèle ou d'un simulateur IP à l'autre, sans modifier le reste de la co-simulation. Afin de pouvoir généraliser notre discours et de pouvoir nous focaliser sur les simulateurs IP principaux, nous faisons quelques hypothèses sur les simulateurs IP dont les modèles sont à intégrer :

1. Ils reposent sur le formalisme des événements discrets.
2. Leur exécution dépend d'une pile d'événements.
3. Des comportements applicatifs peuvent être modélisés.

Il convient de noter que ces hypothèses restrictives portent uniquement sur le domaine des réseaux IP, et n'interdisent en rien l'intégration de modèles continus pour d'autres domaines (e.g. systèmes physiques). Nous savons que ces hypothèses sont valides au moins pour les simulateurs IP les plus populaires, tels que NS-3, OMNeT++/INET ou encore OpNet. En outre, nous souhaitons que notre solution n'impose pas de modifier le code du simulateur lui-même, afin de permettre d'utiliser des instances du logiciel qui sont déjà installées.

Les simulateurs IP existants ne sont généralement pas conçus pour intégrer des co-simulations, et fonctionnent de façon tout à fait autonome. Par conséquent, pour intégrer un modèle IP et son simulateur à une co-simulation, il est nécessaire de :

1. ouvrir le modèle IP en lui ajoutant la notion de port DEVS ;
2. ouvrir le simulateur IP en lui ajoutant des fonctions issues du protocole de simulation DEVS ;
3. asservir le simulateur IP, pour contrôler sa dynamique.

5.2 Ouverture des modèles

Les ports DEVS permettent de créer des passerelles entre le modèle IP et la plateforme de co-simulation. Les ports de sortie doivent être capables de recevoir des données applicatives qui transitent dans le réseau IP modélisé, pour les transmettre sous forme d'événements externes sortants, à la plateforme de co-simulation. À l'inverse, les ports d'entrée doivent être capables de réceptionner des données applicatives sous la forme d'événements externes d'entrée, de la part de la plateforme de co-simulation, et de les envoyer dans le réseau IP modélisé.

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

La Figure 5 propose un exemple de double couplage structurel, avec deux nœuds reliés via un réseau IP minimaliste. Les nœuds sont équipés de ports DEVS (coloration rouge) et le sens de la flèche indique si ce sont des ports d'entrée, de sortie, ou les deux.

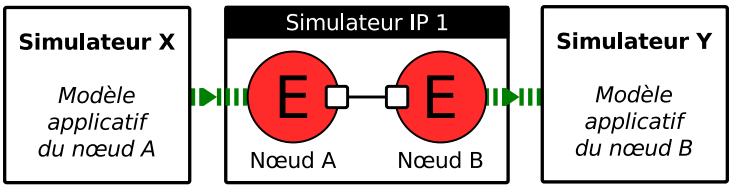


FIGURE 5 – Couplage structurel entre un nœud A qui envoie des données (générées par un modèle applicatif externe) à un nœud B (qui traite ces données via un autre modèle applicatif externe), au travers d'un réseau IP minimaliste.

Les objets correspondants aux ports DEVS dans les modèles doivent être capables de proposer quelques primitives, permettant d'échanger des informations avec le wrappeur DEVS. Ainsi, les objets doivent notamment être capables de 1) notifier au wrappeur DEVS lorsqu'ils sont prêts à être utilisés, 2) recevoir une donnée et la transmettre au réseau IP simulé, et 3) indiquer s'ils ont reçu des données applicatives à leur transmettre, et les leur transmettre le cas échéant. Afin d'illustrer ces interactions, le diagramme de séquence de la Figure 6 propose un exemple de transmission d'une donnée applicative, entre les deux nœuds de la Figure 5. Dans cet exemple, la transmission dans le réseau IP simulé est réalisée avec le protocole TCP : selon les cas, les ports doivent être capables de fonctionner a minima avec UDP et TCP (les deux protocoles de transports largement majoritaires dans les réseaux IP).

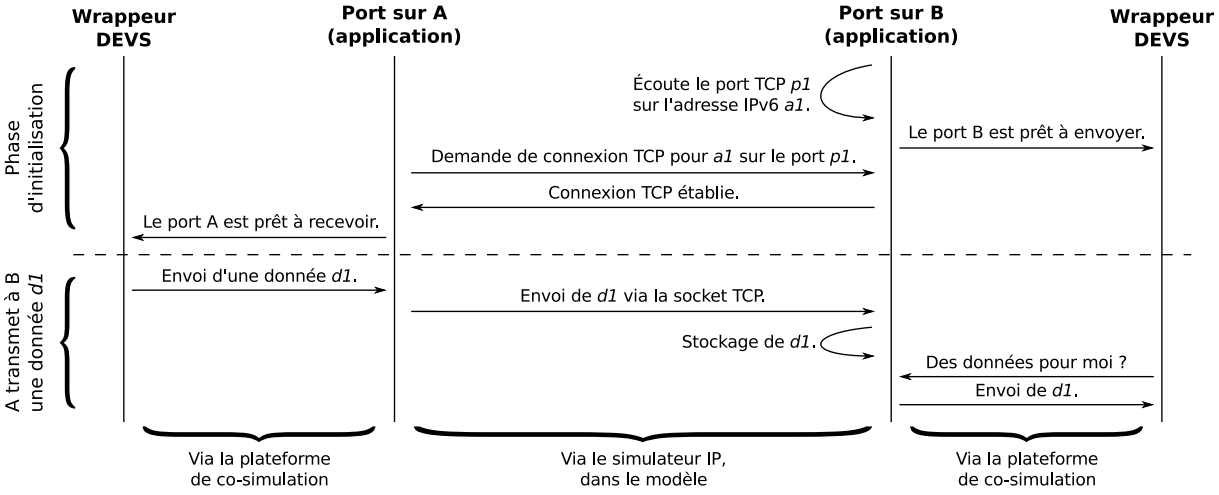


FIGURE 6 – Diagramme de séquence du fonctionnement des ports structurels, avec l'exemple d'une donnée applicative *d1* envoyée de A à B en TCP. Pour des raisons de lisibilité, l'unique wrappeur DEVS est représenté deux fois.

5.3 Ouverture du simulateur

Rôle du wrappeur DEVS

Les modèles étant maintenant ouverts grâce à la notion de port, nous pouvons nous focaliser sur l'ouverture et l'asservissement du simulateur. Cela nécessite de spécifier le

wrappeur DEVS qui correspond à l'interface chargée de faire le lien entre la plateforme de co-simulation, et le modèle IP et son simulateur. Il a pour rôles :

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

Du point de vue de MECSYCO, il s'agit de l'artéfact de modèle. D'un point de vue logiciel, il s'agit d'un thread séparé de celui du simulateur IP, qui est en capacité de communiquer avec ce dernier.

Implémentation du protocole de simulation DEVS

Wrapper le simulateur IP en DEVS implique de lui ajouter les fonctionnalités liées au protocole de simulation DEVS. Dans le cadre de MECSYCO, il suffit d'implémenter les fonctions qui ont été présentées dans la Section 4.3.

Grâce aux objets correspondants aux ports DEVS que nous avons définis, le développement de ces fonctions est dans la grande majorité trivial. Elles consistent à utiliser à la fois des fonctions proposées par ces objets, et utiliser des fonctions directement fournies par l'API du simulateur IP (e.g. récupération du temps courant de simulation). Le seul point qui n'est pas trivial est de contrôler l'exécution du simulateur IP. Il est donc nécessaire de modifier la dynamique de ce dernier, de façon à ce qu'il puisse être asservi par le wrappeur DEVS.

5.4 Asservissement du simulateur IP

Les simulateurs IP existants ne sont généralement pas conçus pour être intégrés à des co-simulations. Ils utilisent par conséquent des ordonnanceurs qui reposent sur une boucle principale dont l'objectif est d'exécuter le plus rapidement possible tous les événements internes dans la pile d'événements, jusqu'à épuisement.

Le remplacement dynamique de l'ordonnanceur à l'exécution (durant l'initialisation de la simulation) est une fonctionnalité qui est généralement proposée, notamment pour permettre d'utiliser des ordonnanceurs « temps réel ». Nous proposons donc d'utiliser cette possibilité pour ajouter un nouvel objet ordonnanceur (spécifique à la plateforme de co-simulation) à la bibliothèque du simulateur IP. L'Algorithme 2 présente un exemple d'ordonnanceur modifié.

L'ajout de verrous dans la boucle principale (lignes 2, 3 et 15) permet d'arrêter et relancer celle-ci à loisir. En manipulant ces verrous, le wrappeur DEVS a la possibilité d'ordonner au simulateur IP de traiter son prochain événement interne, et d'attendre le prochain ordre, avant de passer au suivant. Ce système de verrous permet de créer une alternance, entre l'exécution du thread principal du simulateur IP, et celui du wrappeur.

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

Algorithme 2 . Exemple de boucle principale utilisée par un simulateur IP, modifiée pour pouvoir être contrôlée par un wrappeur DEVS.

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

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

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

6 Cas d'utilisation

Notre cas d'utilisation est inspiré par différents travaux autour des systèmes de chauffage intelligents [Floros et al., 2014, Gilpin et al., 2014]. Il correspond à la modélisation et simulation d'un système de chauffage intelligent pour une collectivité, doté d'une fonction d'effacement de consommation électrique. Nous voulons simuler l'évolution de la température dans deux bâtiments équipés de radiateurs électriques et calculer l'énergie consommée par ces radiateurs. Un contrôleur devra ensuite se charger de limiter les pics de consommation dus au chauffage. Pour ce faire, celui-ci pourra temporairement désactiver des radiateurs en fonction des informations qu'il recevra sur la température et la consommation énergétique. L'interaction avec les bâtiments se fera via un réseau IP dont nous souhaitons évaluer l'impact sur les résultats.

La partie thermique comporte trois modèles. Le premier décrit l'évolution de la température extérieure. Le second modèle et le troisième modèle décrivent chacun l'évolution de la température d'un bâtiment en fonction de la température extérieure. Nous connectons ces trois modèles de façon à ce que les modèles des bâtiments reçoivent l'évolution de la température extérieure, pendant la co-simulation.

Nous définissons ensuite le modèle du contrôleur qui sera utilisé deux fois (une fois pour chaque bâtiment). Chaque contrôleur reçoit en entrée les températures et les puissances instantanées consommées par toutes les pièces de son bâtiment. Si nécessaire, chaque contrôleur peut générer des ordres de coupure ou de réouverture des chauffages qui sont transmis aux bâtiments via des ports d'entrée dédiés.

Le modèle de réseau IP est situé entre les modèles des bâtiments et des contrôleurs. Les sorties des modèles de bâtiment vont d'abord dans le modèle réseau, et transitent par celui-ci avant d'arriver aux modèles des contrôleurs. Réciproquement, les ordres du contrôleur vont aussi transiter par le modèle réseau avant d'arriver aux modèles de bâtiments. Le modèle réseau permet alors d'ajouter des délais et des perturbations (i.e. des pertes de paquets, du bruit) au système. La Figure 7 présente les interactions entre les différents composants du multi-modèle.

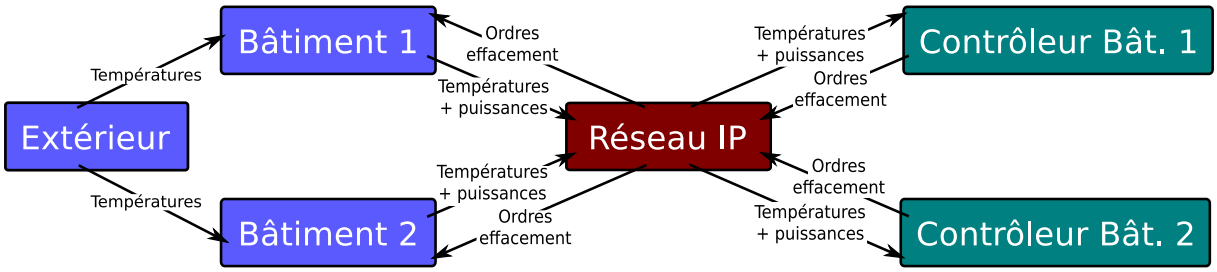


FIGURE 7 – Vue intuitive du multi-modèle du système de chauffage intelligent.

Ce cas d'utilisation est un exemple jouet qui n'a pas la prétention d'être particulièrement réaliste. Nous conservons des modèles individuels très simples pour nous concentrer sur la démonstration des propriétés ci-dessous de MECSYCO :

- **Gestion de l'interopérabilité logicielle :** les modèles utilisés dans la co-simulation sont implémentés dans des logiciels de M&S différents. Les modèles thermiques sont définis en utilisant Modelica, puis sont exportés en FMU de type ¡¡ échange de

modèles ¿¿ et ¡¡ co-simulation ¿¿, le réseau IP est construit sur NS-3, et le modèle de contrôleur est défini de manière ad-hoc en Java. Nous montrons que MECSYCO gère proprement l'échange de données entre ces logiciels hétérogènes.

- **Intégration multi-formalisme :** les modèles utilisés dans la co-simulation sont issus de différents formalismes. Les modèles thermiques sont hybrides composés d'équations différentielles et discrètes. Le modèle du réseau est un à événements discrets, tandis que le modèle du contrôleur est à pas de temps. Nous montrons que MECSYCO permet l'intégration rigoureuse de ces modèles hétérogènes.
- **Intégration multi-représentation :** les modèles évoluent à différentes échelles de temps, à savoir la seconde pour les modèles du contrôleur et les modèles thermiques, et la nanoseconde pour le modèle IP. Nous montrons que MECSYCO permet la synchronisation rigoureuse de ces modèles durant la co-simulation.
- **Exécution distribuée et multi-plateforme :** nous exécutons la co-simulation sur deux ordinateurs connectés en réseau local. Ces deux ordinateurs fonctionnent avec différents systèmes d'exploitation et différentes implémentations de MECSYCO. Le modèle du réseau IP est exécuté sur GNU/Linux Debian avec l'implémentation C++ de MECSYCO, tandis que les autres modèles sont simulés sur des machines Microsoft Windows avec la version Java de MECSYCO.

6.1 Présentation des modèles

Des annexes fournissent des détails supplémentaires sur chaque modèle.

Modèles du système thermique

Deux types de modèles ont été créés pour le système thermique. Le premier permet de simuler l'évolution de la température extérieure (dans notre cas une simple sinusoïde représentant un cycle de températures jour/nuit), tandis que le second modélise un bâtiment et permet d'obtenir l'évolution de la température dans les pièces ainsi que la consommation électrique pour les chauffer. Les deux bâtiments à modéliser étant identiques, ce deuxième modèle est utilisé deux fois lors de la co-simulation.

Chaque bâtiment est composé de 10 pièces liées par un couloir suivant la Figure 8. Chaque pièce subit l'influence de la température extérieure, des pièces adjacentes, et contient un radiateur électrique avec un thermostat. Ce thermostat allume le radiateur lorsque la température tombe sous un seuil prédéfini et l'éteint lorsque la température atteint une température maximale. Cela signifie que pour chaque pièce de chaque bâtiment il faudra, lors de la co-simulation, gérer les événements d'état liés à l'allumage et à l'extinction des radiateurs. Nous avons ensuite des modèles de pièce (qui calculent l'évolution des températures), de mur (qui déterminent l'échange thermique entre deux pièces ou entre une pièce et l'extérieur), et de radiateur électrique (qui donnent les puissances instantanées consommées pour chauffer). Pour simplifier, nous considérons ici que tous les radiateurs sont configurés de la même façon (même température de consigne, puissance et tolérance).

Ces modèles ont été construits avec la bibliothèque standard de Modelica. La partie thermique a été modélisée via la bibliothèque *Modelica.Thermal.HeatTransfer* (les pièces

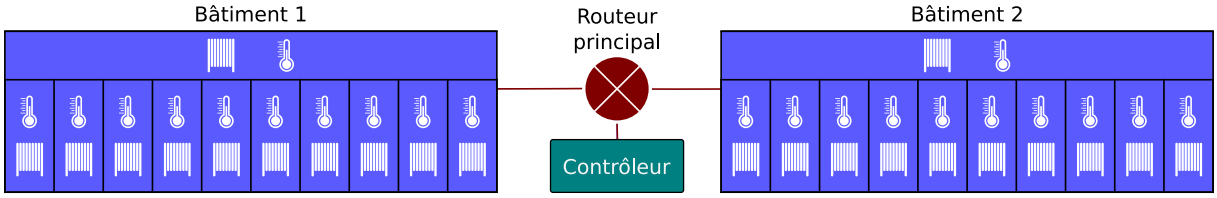


FIGURE 8 – Système de chauffage intelligent, avec deux bâtiments de onze pièces.

sont vues comme des capacités thermiques tandis que les murs sont des conducteurs thermiques) et la partie électrique a été modélisée via la bibliothèque *Modelica.Electrical.Analog* (le radiateur est représenté par une résistance électrique). La liste ci-dessous présente les entrées et les sorties qui permettent au modèle de bâtiment d'interagir avec les autres modèles.

Entrées :

- $blackout_i$: entrée booléenne qui permet d'éteindre le radiateur de la pièce i.
- T_{out} : entrée continue représentant la température extérieure en K.

Sorties :

- R_iTemp est une sortie discrète qui représente la température de la pièce i échantillonnée (selon le paramètre *period*) et envoyée régulièrement par un thermomètre au contrôleur. La mise à jour de cette sortie a donc lieu lors d'événements temporels.
- R_iPow est une sortie discrète qui représente la puissance instantanée consommée dans la pièce i. Cette variable est mise à jour à chaque fois que le radiateur s'allume ou s'éteint. Cette sortie évolue donc lors d'événements de changement d'état.

Le modèle du bâtiment combine donc des comportements discrets et continus. Sa simulation nécessite de résoudre des équations différentielles (pour l'évolution de la température) en prenant en compte des événements temporels (l'échantillonnage des températures de sortie) et des événements d'état (les seuils d'allumage et d'extinction des radiateurs). En considérant les 11 pièces du bâtiment, nous avons 11 équations différentielles à résoudre et 22 seuils d'événements d'état à superviser simultanément. De plus, il est nécessaire de prendre en compte la température extérieure T_{out} qui évolue de manière continue, ainsi que les commandes $blackout_i$ du contrôleur qui évoluent de manière événementielle.

Modèle du contrôleur

Le contrôleur doit permettre de réduire la durée des pics de consommation énergétique des bâtiments. Pour ce faire, il désactive temporairement certains radiateurs lorsque la puissance instantanée totale consommée par le bâtiment dépasse un seuil Pow_{max}. Nous acceptons alors que la température dans les pièces tombe temporairement sous la température de consigne. Cependant, nous assurons que la température soit supérieure à un second seuil $Temp_{min}$ en K, inférieur à la température de consigne des radiateurs, afin d'assurer un confort minimal dans chaque pièce.

Le contrôleur reçoit en entrée la température et la puissance instantanée consommée pour chaque pièce du bâtiment, il conserve en mémoire les dernières valeurs reçues pour chaque pièce (dans des variables Pow_i et $Temp_i$). Il évalue ensuite à une certaine fréquence si des radiateurs doivent être désactivés ou réactivés, et envoie les ordres correspondant aux radiateurs. L'évaluation se déroule en deux temps :

1. Pour chaque pièce, si la température est inférieure à $Temp_{min}$, il envoie un ordre de réactivation au radiateur de la pièce correspondante.
2. Il calcule la puissance instantanée consommée par le bâtiment et la compare au seuil Pow_{max}. Si c'est supérieur, il calcule le nombre N_{off} de radiateurs devant être éteints et envoie ensuite un ordre de désactivation aux N_{off} pièces les plus chaudes.

Ce contrôleur est décrit par un modèle à pas de temps où chaque pas de temps correspond à un point d'évaluation. Les variables Pow_i et $Temp_i$ sont mis à jour grâce à des ports d'entrée spécifiques. Les ordres d'activation et de désactivation sont des booléens envoyés par les ports de sortie $blackout_i$. Du point de vue du wrapping DEVS, chaque pas de temps est vu comme un événement interne et chaque entrée/sortie est vue comme un événement externe.

Modèle de réseau IP

La topologie IP correspondant au système qui a été décrit est visible en Figure 9. Les radiateurs et thermomètres sont représentés individuellement, puisqu'ils correspondent chacun à un équipement réseau distinct. On sait que dans le système, les liens entre les routeurs de sortie des bâtiments et le routeur principal sont soumis à des parasites, entraînant du bruit dans les transmissions. La représentation de ces liens dans le modèle utilise donc un modèle d'erreurs de type *RateErrorModel* (fourni par la bibliothèque NS-3), permettant d'inverser régulièrement un bit dans les communications transmises, correspondant à l'effet du bruit. Le nombre de bits inversés dépend d'un taux configuré dans le modèle (e.g. un bit sur mille inversé).

Concernant les ports structurels DEVS, ils sont au nombre de 132 dans le modèle IP (x représentant le numéro du bâtiment et y le numéro de la pièce dans le bâtiment) :

- Chaque nœud représentant un thermomètre est équipé d'un port d'entrée $BxRyTempIn$.
- Chaque nœud représentant un radiateur est équipé à la fois d'un port d'entrée $BxRyPowIn$ et d'un port de sortie $BxRyBlackoutOut$.
- Le nœud représentant le contrôleur est équipé de tous les ports de sortie $BxRyTempOut$ et $BxRyPowOut$ et de tous les ports d'entrée $BxRyBlackoutIn$, pour x valant 1 et 2 et y variant de 1 à 11.

Les ports *Pow* correspondent à des échanges de puissances, les ports *Temp* à des échanges de températures, et les champs *Blackout* à des échanges d'ordre d'effacement. Les températures et les puissances sont transmises sous forme de double, tandis que les ordres sont des booléens (valant ¡¡ vrai ¿¿ lorsque le radiateur doit obligatoirement s'éteindre et ¡¡ faux ¿¿ lorsqu'il a de nouveau le droit de se rallumer). Les ports d'entrée *In* établissent tous une connexion TCP ou UDP (selon l'expérimentation ci-dessous) avec le port de sortie *Out* correspondant.

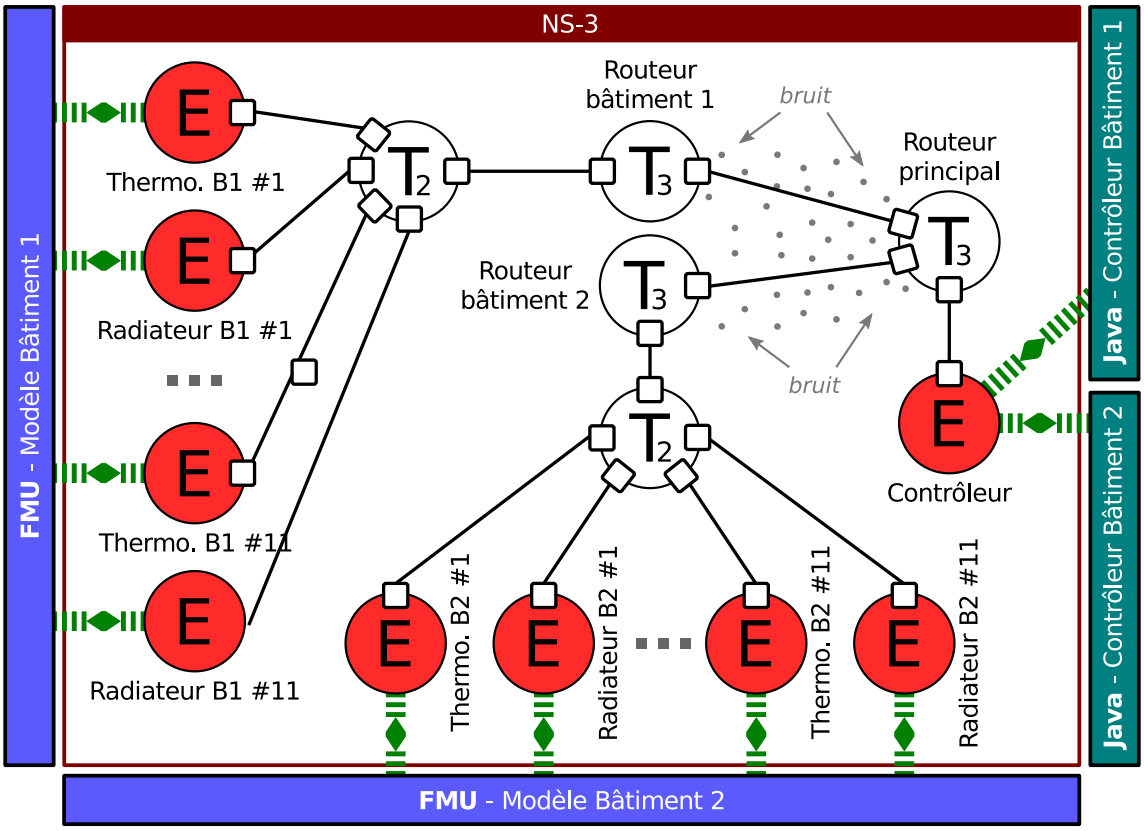


FIGURE 9 – Couplages structurels du modèle IP du système de chauffage intelligent (avec deux pièces par bâtiment – #1 et #11 – au lieu de onze).

6.2 Co-simulations et résultats

Nous avons exporté nos modèles de bâtiments en FMU pour l'échange de modèle (FMU sans solveur) de sorte à gérer les événements discrets. Le modèle de la température extérieure a lui été exporté en FMU pour la co-simulation (FMU avec solveur intégré). Pour simuler chaque FMU bâtiment, nous avons implémenté dans l'artéfact de modèle un solveur hybride QSS2 [Kofman, 2002] que nous avons adapté au standard FMI. L'intérêt de ce solveur est que, de par sa nature événementielle, il peut être directement exprimé en DEVS et est parfaitement adapté à la simulation hybride. En effet, QSS rend naturel la détection des state-events et l'intégration d'entrées discrètes pendant la résolution du système continu [Kofman, 2004]. Cette caractéristique est due au fait que, contrairement aux autres solveurs d'équations différentielles, QSS fonde sa stratégie de résolution sur la quantification de l'espace d'état plutôt que sur la discrétisation du temps.

Les contrôleurs sont paramétrés de sorte à ne tolérer qu'un seul radiateur actif à la fois par bâtiment. L'évaluation des commandes à envoyer aux bâtiments est réalisée toutes les minutes, les capteurs des bâtiments envoient eux-mêmes leurs informations toutes les minutes mais avec 30 secondes de décalage. Nous connectons le modèle wrappé de sorte à former le modèle couplé de la Figure 10a.

Enfin, le modèle du réseau IP est ajouté à la co-simulation pour former le modèle couplé de la Figure 10. Nous avons configuré NS-3 de façon à utiliser un protocole UDP sans somme de contrôle, avec un modèle d'erreur inversant un bit sur 10000.

Comme NS-3 évolue à une échelle de temps de l'ordre de la nanoseconde alors que nos FMU ont une évolution de l'ordre de la seconde, nous ajoutons des opérations temporelles aux artéfacts de couplage qui lient NS-3 et les FMU, de sorte à faire les conversions nécessaires sur les dates des événements échangés.

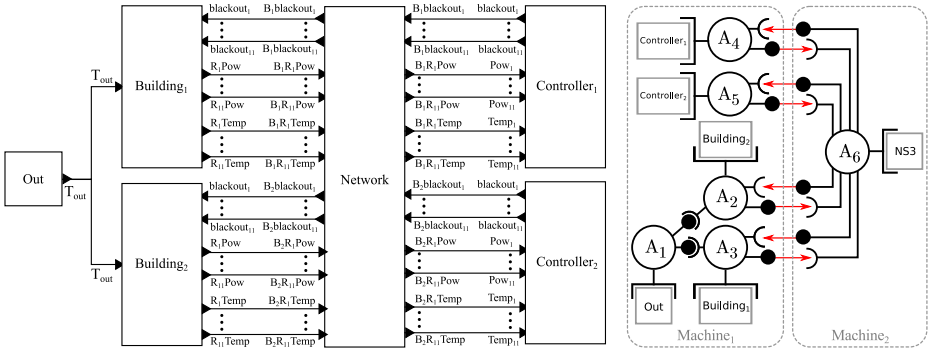
Il est important de noter que les modèles sont compatibles avec des systèmes d'exploitation différents : les FMU que nous avons compilées ne peuvent être exécutées que sur Microsoft Windows, tandis que le modèle NS-3 ne fonctionne que sur GNU/Linux. De plus, ils utilisent des implémentations de MECSYCO différentes pour wrapper les modèles : les FMU sont wrappées dans la version Java de MECSYCO alors que le modèle NS-3 est wrappé en utilisant la version C++. De ce fait, nous devons distribuer la co-simulation sur deux ordinateurs. Le premier fonctionne sur Windows 10 et utilise la version Java de MECSYCO pour exécuter les FMU et le modèle du contrôleur. Le second fonctionne sur GNU/Linux Debian et utilise la version C++ de MECSYCO pour simuler le modèle NS-3.

La Figure 11a montre les résultats de simulation. Par souci de concision, ces résultats montrent seulement l'évolution du premier bâtiment. Sur ce graphe, les zones grisées représentent les périodes de temps pendant lesquelles les radiateurs sont désactivés suivant les sorties du contrôleur. Les résultats de la simulation sont en accord avec le modèle. Nous pouvons voir en effet que les ordres du contrôleur sont bien pris en compte par le bâtiment : lorsque le contrôleur envoie un ordre de désactivation, le radiateur correspondant arrête immédiatement de fonctionner et la température de la pièce décroit. À l'inverse, lorsqu'un ordre d'activation est envoyé, la température de la pièce correspondante recommence immédiatement à remonter et oscille entre les deux seuils d'événement d'état (i.e. les radiateurs s'allument et s'éteignent au moment où la température de leur pièce atteint un des seuils de state-events).

De plus, si nous définissons un modèle d'erreur inversant 1 bit sur 1000 au lieu de 10000, de sorte à augmenter des perturbations dans les communications, les résultats de la simulation sont changés comme le montre la Figure 11b. Nous pouvons noter que le bruit est si élevé que certains ordres du contrôleur n'arrivent pas à destination (par exemple un ordre de désactivation d'un radiateur au temps 8370s). Nous observons que, comme attendu, plus nous ajoutons de perturbations dans le réseau, plus cela perturbe le système dans son ensemble. Il faut noter aussi que, comme NS-3 utilise des modèles d'erreur introduisant des processus stochastiques, les Figures 11a et 11b présentent seulement un exemple de résultat de simulation.

7 Conclusion

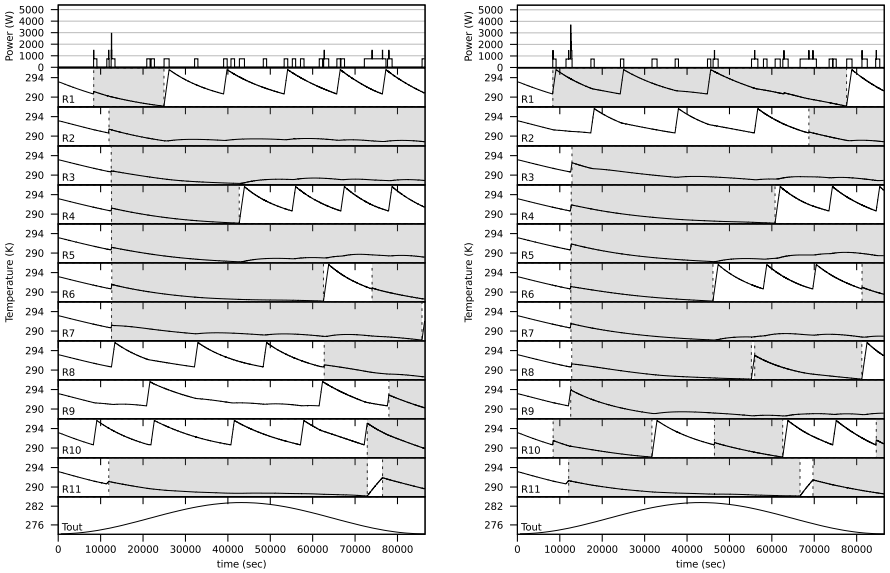
Dans cet article, nous avons présenté l'intergiciel MECSYCO permettant la M&S rigoureuse des systèmes cyber-physiques grâce à une approche de co-simulation. MECSYCO repose sur l'universalité de DEVS pour intégrer des modèles écrits dans différents formalismes. Cette intégration se fait grâce à une stratégie de wrapping de façon à pouvoir intégrer des modèles déjà implémentés dans des logiciels de simulation. L'intergiciel effectue alors la co-simulation de manière parallèle, décentralisée et distribuable grâce à son architecture multi-agent modulaire. Il est important de noter que, si MECSYCO permet d'interfacer de manière générique des modèles aux niveaux logiciel et formel, il n'offre toutefois pas de garantie que ces modèles puissent être composés de manière sémantiquement



(a) Schéma fonctionnel correspondant au modèle DEVS.

(b) Vue MECSYCO de la co-simulation.

FIGURE 10 – Co-simulation du système de bâtiments avec le contrôleur et un réseau IP.



(a) Exemple de résultats de simulation avec UDP et 1 bit inversé tous les 10000 bits.

(b) Exemple de résultats de simulation avec UDP et 1 bit inversé tous les 1000 bits.

FIGURE 11 – Résultat de co-simulation MECSYCO pour le système de chauffage intelligent, pour un seul bâtiment.

valide. Toutes les co-simulations MECSYCO doivent alors être validées pour pouvoir donner des résultats exploitables.

Comme discuté précédemment, bien que l'interfaçage de modèles dans MECSYCO repose sur les garanties formelles offertes par DEVS, sa mise en œuvre s'avère particulièrement difficile dans le cadre des simulateurs de réseaux IP. Nous avons alors proposé une stratégie générique de wrapping DEVS des simulateurs IP pour répondre à ce problème. Cette proposition a été mise en œuvre avec succès pour wrapper les simulateurs de réseaux IP NS-3 et OMNeT++/INET. Il est à noter que cette stratégie n'est pas uniquement applicable dans le cadre de MECSYCO. Elle peut en effet être mise en œuvre dans n'importe quelle plateforme DEVS pour faire interagir des modèles événementiels de réseaux IP avec des modèles écrits dans d'autre formalismes.

Enfin, nous avons montré, à travers le cas d'usage d'une M&S de bâtiment intelligent, comment notre proposition permet de faire interagir de manière rigoureuse dans une co-simulation hybride un modèle du simulateur réseaux NS-3 avec des modèles continus et des modèles à pas de temps.

Une première perspective est de confronter notre proposition à d'autres simulateurs IP pour évaluer son degré de généricité. De la même manière, on peut envisager le wrapping d'autres logiciel de simulation pour construire des co-simulations plus importantes. Au niveau conceptuel, il serait intéressant de proposer un langage graphique pour la modélisation de réseau IP et sa traduction dans différents simulateurs IP. De manière plus générale, la construction de multi-modèle pose la problématique de co-initialisation, y compris dans le cadre de réseaux IP, et des solutions accessibles doivent être proposées pour généraliser ce type d'approche.

Références

[Barros, 2003] Barros, F. J. (2003). Dynamic structure multiparadigm modeling and simulation. *ACM Trans. Model. Comput. Simul.*, 13(3).

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

[Blochwitz et al., 2012] Blochwitz, T., Otter, M., Åkesson, J., et al. (2012). Functional mockup interface 2.0 : The standard for tool independent exchange of simulation models. In *Proc. 9th International Modelica Conference*, pages 173–184.

[Bonneaud, 2008] Bonneaud, S. (2008). *Des agents-modèles pour la modélisation et la simulation de systèmes complexes - Application à l'écosystémique des pêches.* PhD thesis.

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

[Bryant, 1979] Bryant, R. E. (1979). Simulation on a distributed system. In *Proc. of the 16th Design Automation Conf.*

[Camus, 2015] Camus, B. (2015). *Environnement Multi-agent pour la Multi-modélisation et Simulation des Systèmes Complexes.* PhD thesis, Université de Lorraine.

[Camus et al., 2015a] Camus, B., Bourjot, C., and Chevrier, V. (2015a). Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems (WIP). In *Proc. TMS/DEVS 15*. SCS.

[Camus et al., 2015b] Camus, B., Bourjot, C., and Chevrier, V. (2015b). Considering a multi-level model as a society of interacting models : Application to a collective motion example. *JASSS*.

[Camus et al., 2016] Camus, B., Galtier, V., Caujolle, M., Chevrier, V., Vaubourg, J., Ciarletta, L., and Bourjot, C. (2016). Hybrid Co-simulation of FMUs using DEV&DESS in MECSYCO. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative M&S Symposium*.

[Cellier, 1979] Cellier, F. E. (1979). Combined continuous/discrete system simulation languages–usefulness, experiences and future development. *Methodology in systems modelling and simulation*.

[Cellier et al., 2008] Cellier, F. E., Kofman, E., Migoni, G., and Bortolotto, M. (2008). Quantized state system simulation. *Proc. GCMS'08, Grand Challenges in Modeling and Simulation*, pages 504–510.

[Chandy and Misra, 1979] Chandy, K. M. and Misra, J. (1979). Distributed simulation : A case study in design and verification of distributed programs. *IEEE Trans. Software Engineering*.

[Dacharry and Giambiasi, 2005] Dacharry, H. P. and Giambiasi, N. (2005). Formal verification with timed automata and DEVS models : a case study. In *Proc. of Argentine Symposium on Software Engineering*. Citeseer.

[Duboz et al., 2006] Duboz, R., Versmisse, D., Quesnel, G., Muzy, A., and Ramat, E. (2006). Specification of dynamic structure discrete event multiagent systems. *Simulation Series*, 38(2) :103.

[Esquembre and Christian, 2007] Esquembre, F. and Christian, W. (2007). Ordinary differential equations. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*. CRC Press.

[Floros et al., 2014] Floros, X., Bergero, F., Ceriani, N., Casella, F., Kofman, E., and Cellier, F. (2014). Simulation of smart-grid models using quantization-based integration methods. In *Proceedings of the 10 th International Modelica Conference ; March 10-12 ; 2014 ; Lund ; Sweden*, number 096, pages 787–797. Linköping University Electronic Press.

[Fujimoto, 2001] Fujimoto, R. M. (2001). Parallel simulation : parallel and distributed simulation systems. In *Proceedings of the 33nd conference on Winter simulation*, WSC '01. IEEE Computer Society.

[Gilpin et al., 2014] Gilpin, L., Ciarletta, L., Presse, Y., Chevrier, V., and Galtier, V. (2014). Co-simulation Solution using AA4MM-FMI applied to Smart Space Heating Models. In *7th International ICST Conference on Simulation Tools and Techniques*, pages 153–159, Lisbon, Portugal.

[Jacques and Wainer, 2002] Jacques, C. J. and Wainer, G. A. (2002). Using the CD++ DEVS toolkit to develop petri nets. In *Summer Computer Simulation Conference*, pages 51–56. Society for Computer Simulation International ; 1998.

[Jennings, 2001] Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Commun. ACM*, 44(4) :35–41.

[Kim and Kim, 1998] Kim, Y. J. and Kim, T. G. (1998). A heterogeneous simulation framework based on the DEVS BUS and the high level architecture. In *Proc. of WSC '98*, volume 1. IEEE.

[Kofman, 2002] Kofman, E. (2002). A second-order approximation for devs simulation of continuous systems. *Simulation*, 78(2) :76–89.

[Kofman, 2004] Kofman, E. (2004). Discrete event simulation of hybrid systems. *SIAM Journal on Scientific Computing*, 25(5).

[Mittal et al., 2015] Mittal, S., Ruth, M., Pratt, A., et al. (2015). A system-of-systems approach for integrated energy systems modeling and simulation. In *Proc. of SummerSim' 15*, pages 1–10. SCS/ACM.

[Mosterman, 2007] Mosterman, P. (2007). Hybrid dynamic systems : Modeling and execution. In Fishwick, P. A., editor, *Handbook of dynamic system modeling*, chapter 15, pages 1–26. CRC Press.

[Praehofer, 1991] Praehofer, H. (1991). System theoretic formalisms for combined discrete-continuous system simulation. *International Journal of General System*, 19(3) :226–240.

[Quesnel et al., 2005] Quesnel, G., Duboz, R., Versmisse, D., and Ramat, É. (2005). DEVS coupling of spatial and ordinary differential equations : VLE framework. In *Proc. OICMS '05*.

[Rajkumar et al., 2010] Rajkumar, R. R., Lee, I., Sha, L., and Stankovic, J. (2010). Cyber-physical Systems : The Next Computing Revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA. ACM.

[Ricci et al., 2007] Ricci, A., Viroli, M., and Omicini, A. (2007). Give agents their artifacts : the A&A approach for engineering working environments in MAS. In *AAMAS '07*. ACM.

[Siebert, 2011] Siebert, J. (2011). *Approche multi-agent pour la multi-modélisation et le couplage de simulations. Application à l'étude des influences entre le fonctionnement des réseaux ambiants et le comportement de leurs utilisateurs.* These, Université Henri Poincaré, Nancy 1.

[Uhrmacher and Schattenberg, 1998] Uhrmacher, A. M. and Schattenberg, B. (1998). Agents in discrete event simulation. In *ESS'98*. SCS.

[Vangheluwe, 2000] Vangheluwe, H. (2000). DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Proc. of CACSD '00*.

[Vangheluwe et al., 2002] Vangheluwe, H., De Lara, J., and Mosterman, P. J. (2002). An introduction to multi-paradigm modelling and simulation. In *Proc. AIS2002.*, pages 9–20.

[Vaubourg et al., 2016] Vaubourg, J., Chevrier, V., Ciarletta, L., and Camus, B. (2016). Co-simulation of ip network models in the cyber-physical systems context, using a devs-based platform. In SCS/ACM, editor, *Communications and Networking Simulation Symposium (CNS'16)*.

[Vaubourg et al., 2015] Vaubourg, J., Presse, Y., Camus, B., et al. (2015). Multi-agent multi-model simulation of smart grids in the MS4SG project. In *Proc. PAAMS 15*, pages 240–251. Springer.

[Wainer and Giambiasi, 2001] Wainer, G. A. and Giambiasi, N. (2001). Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation*, 76(1) :22–39.

[Wilensky, 1999] Wilensky, U. (1999). Netlogo. http ://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

[Zeigler et al., 2000] Zeigler, B., Praehofer, H., and Kim, T. (2000). *Theory of Modeling and Simulation : Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

[Zeigler, 2006] Zeigler, B. P. (2006). Embedding DEV&DESS in DEVS. In *Proc. DEVS Integrative M&S Symp.*

A Détails du modèle thermique

Pour décrire notre système thermique nous définissons des modèles de pièce qui calculent leur température, des modèles de murs qui déterminent le flux de chaleur entre deux pièces (ou entre une pièce et l'extérieur), et des modèles de radiateurs électriques qui fournissent la puissance consommée pour chauffer. Pour construire ces modèles, nous avons utilisé la bibliothèque standard de Modelica. Les parties thermique et électrique sont construites respectivement avec les bibliothèques `Modelica.Thermal.HeatTransfer` et `Modelica.Electrical.Analog`.

Les pièces sont modélisées comme des capacités thermiques. Chaque salle est vue comme un volume d'air avec une température. Les perturbations venant des murs et des radiateurs sont modélisées comme des échanges de chaleur. Le comportement de la salle i est donné par l'équation :

$$C_i * \frac{dT_i}{dt} = Q_{in_i} + Q_{heater_i}$$

Où :

- C_i constante, la capacité thermique de la pièce i en J/K.
- T_i température de la pièce i en K.
- Q_{in_i} somme des flux de chaleur provenant des murs connectés à la pièce i.
- Q_{heater_i} est la flux de chaleur provenant du radiateur. Nous considérons que ce flux est égal à la puissance instantanée consommée en W -i.e. $R_i Pow = Q_{heater_i}$.

Les flux de chaleur sont calculés de la manière suivante. C'est le modèle du mur qui calcule le flux de chaleur entre les deux volumes d'air k et l auxquels il est connecté. Notons qu'un volume d'air peut aussi bien être une pièce que l'environnement extérieur. Le flux de chaleur dépend de la température des pièces k et l ainsi que de la conductance thermique du mur. Cela donne les équations suivantes :

$$Q_{kl} = G_i * (T_k - T_l) \qquad\qquad Q_{lk} = -Q_{kl}$$

Où :

- G_{kl} conductance thermique du mur (constante) en J/K.
- Q_{kl} (resp. Q_{lk}) flux de chaleur allant du volume d'air k (resp. l) vers le volume d'air l (resp. k) en J.

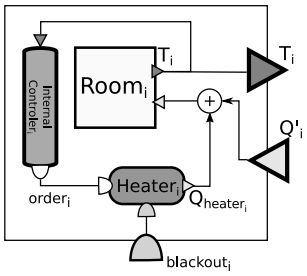


FIGURE 12 – Modèle "R_i" d'une salle avec son chauffage

Q_{heater_i} est calculé par le radiateur électrique qui est modélisé comme un circuit électrique basique avec une résistance électrique, une tension constante et un interrupteur. Le comportement du radiateur est donné par les équations suivantes :

$$\textbf{if } order_i \textbf{ and not } blackout_i \textbf{ then } Q_{heater_i} = \frac{U^2}{R} \textbf{ else } Q_{heater_i} = 0$$

Où :

- U tension constante en V.
- R en Ω, résistance électrique constante de chaque radiateur du bâtiment.
- $order_i$ booléen représentant la commande du contrôleur interne au radiateur. Si Vrai alors le radiateur est allumé.

$order_i$ devient Vrai lorsque la température de la pièce devient inférieure à une valeur minimale, et redevient Faux lorsque cette température atteint une valeur maximale. Cela correspond aux conditions suivantes :

$$\textbf{when } T_i \leq T_{wanted} - \frac{bandwidth}{2} \textbf{ then } order_i = \textbf{true}$$
$$\textbf{else when } T_i \geq T_{wanted} + \frac{bandwidth}{2} \textbf{ then } order_i = \textbf{false}$$

Où :

- T_{wanted} température de consigne dans toutes les pièces du bâtiment.
- $bandwidth_i$ tolérance qui détermine les seuils de température minimale et maximale.

Chaque port discret $R_i Temp$ échantillonne l'évolution de la température de la salle i selon le code Modelica suivant :

$$\textbf{when } sample(0, period) \textbf{ then } \; R_i\,Temp \; = T_i \; \textbf{ end when};$$

Où *period* est un intervalle de temps constant en s. La fonction Modelica $sample(0, period)$ est utilisée pour mettre à jour $R_i\,Temp$ à chaque *period* de temps pour obtenir le signal discret envoyé régulièrement par les thermomètres aux contrôleurs.

La Figure 12 représente le modèle d'une pièce avec son radiateur et son contrôleur interne sous forme de diagramme bloc. En utilisant ce modèle, l'ensemble du bâtiment est représenté Figure 13. OpenModelica indique que ce modèle contient 1622 équations dont 11 équations différentielles.

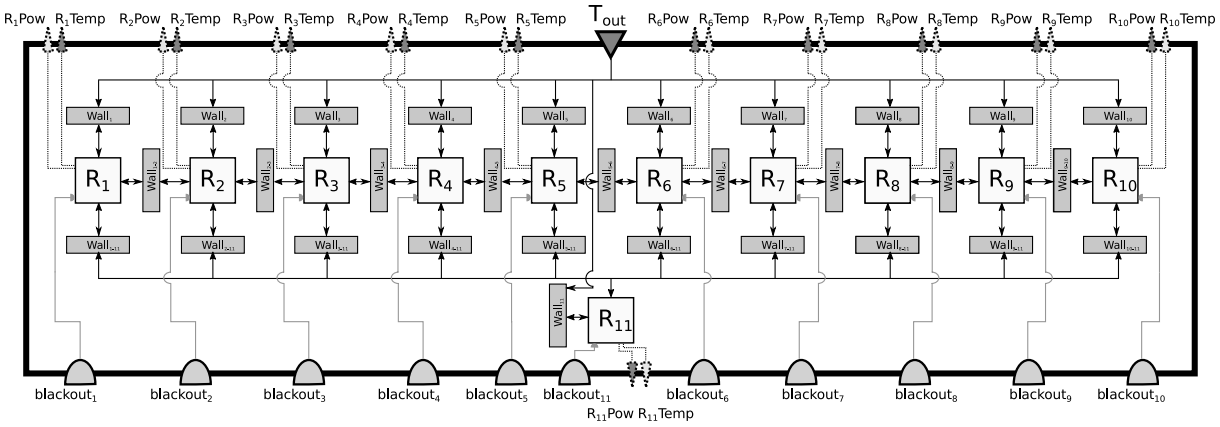


FIGURE 13 – Modèle du bâtiment

B Détail du modèle du Contrôleur

Les variables $Temp_i$ et Pow_i du contrôleur servent à stocker respectivement les dernière valeurs de la température et de la puissance instantanée consommée reçues par les capteurs de chaque pièce i du bâtiment. En fonction de ces variables, le contrôleur évalue à une certaine fréquence de temps si certains radiateurs doivent être éteints ou rallumés. Si c'est le cas, il envoie les ordres correspondants aux radiateurs.

Le mécanisme pour calculer les ordres à chaque évaluation est le suivant :

1. Le contrôleur vérifie pour chaque pièce i si $Temp_i \leq Temp_{min}$. Si oui, il envoie une commande d'activation au radiateur.

2. Pour savoir si les radiateurs doivent être désactivés, le contrôleur calcule la puissance instantanée totale consommée par le bâtiment Pow_{tot} :

$$Pow_{tot} = \left(\sum_{i=1}^{11} Pow_i\right) + \frac{U^2}{R} * N_{on}$$

   Où :

   - N_{on} nombre de radiateurs qui viennent d'être activés à l'étape 1.
   - U tension constante aux bornes des radiateurs en V.
   - R résistance électrique constante des radiateurs en Ω.

   Si $Pow_{tot} \geq Pow_{max}$, alors le contrôleur détermine le nombre $N_{off} \in \mathbb{N}$ de radiateurs qui doivent être éteints pour ramener Pow_{tot} sous Pow_{max}. Le contrôleur désactive alors les N_{off} pièces ayant les plus hautes températures. N_{off} est calculé comme suit (Avec $int : \mathbb{R} \to \mathbb{N}$ la fonction troncature) :

$$N_{off} = \mathsf{int}\left(\frac{Pow_{tot} - Pow_{max}}{U^2/R}\right) + 1$$

C Détail du modèle de réseau IP

La Figure 14 présente la topologie du réseau IP (avec seulement 3 salles au lieu de 11). Nous considérons qu'il y a un switch (S) par bâtiment, connectant tous les radiateurs et les

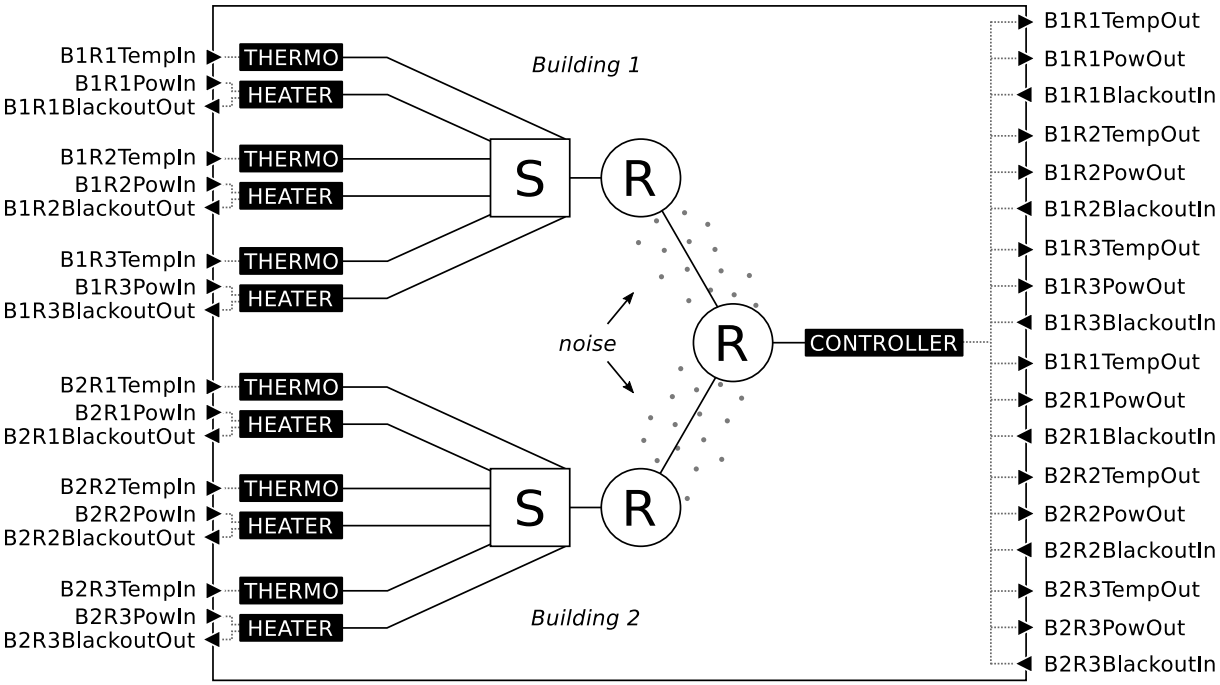


FIGURE 14 – Topologie du réseau IP, avec trois pièces par bâtiment, et avec les ports DEVS sur les côtés

thermomètres au même réseau local. Ensuite, chaque bâtiment est connecté à Internet avec son propre routeur (R). Internet est modélisé comme un unique routeur central et le contrôleur est directement connecté dessus. Les équipements réseaux sont connectés aux modèles extérieurs avec des ports d'entrée et de sortie (repéré sur le côté de la Figure), pour recevoir et transmettre des données. Dans ce cas, les modèles extérieurs correspondent à la couche applicative des appareils réseaux.

Les radiateurs et les thermomètres peuvent échanger leurs mesures ou les commandes avec le contrôleur au travers de notre imitation d'Internet, ceci grâce à des connections TCP ou UDP en fonction du choix de l'expérimentateur. Le choix de TCP (protocole fiable) ou d'UDP (protocole non fiable) est important pour le modèle d'erreur introduit sur le lien entre les routeurs des bâtiments et Internet, celui-ci permet de modéliser du bruit sur le réseau. Le modèle d'erreur peut être configuré en choisissant le taux d'erreur (par exemple, un bit incorrect sur mille envoyés). Le modèle réseau est construit en utilisant la bibliothèque standard de NS3.