Une approche composant pour DEVS

Thomas Paris, Laurent Ciarletta, Vincent Chevrier

▶ **To cite this version:**

Thomas Paris, Laurent Ciarletta, Vincent Chevrier. Une approche composant pour DEVS. JDF 2018 - Les Journées DEVS Francophones, Apr 2018, Cargèse, France. hal-01902758

HAL Id: hal-01902758
https://hal.science/hal-01902758v1

Submitted on 23 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A component approach for DEVS

Une approche composant pour DEVS

Thomas Paris*, Laurent Ciarletta* et Vincent Chevrier*

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

Mai 2018

Résumé : Dans cet article, nous nous intéressons à la réutilisation de modèles pour permettre la conception de multi-modèles de systèmes complexes. Par exemple, la modélisation d'un réseau électrique intelligent nécessite des modèles de trois domaines différents (réseau électrique, télécommunication et système de contrôle). De nombreux outils de M&S dédiés permettent déjà de développer ces types de modèle. Il serait donc intéressant de pouvoir réutiliser directement les modèles issus de ces logiciels dédiés pour concevoir des multi-modèles de systèmes complexes. La norme FMI, basée sur l'export et l'import de modèles sous forme de composants logiciels, permet aux outils basés sur les ODE et les DAE d'échanger des modèles et d'effectuer des co-simulations. Devant le succès de ce standard, nous proposons dans cet article quelques pistes de réflexion sur l'intérêt d'une démarche similaire utilisant le formalisme DEVS.
Mots-clés : Standardisation, Co-simulation, DEVS, Réutilisation

Abstract: *In this article, we are interested in the reuse of models to allow the design of complex system multi-models. For instance, smart-grid modeling needs at least models from three domains (electrical grid, IP network, control system). Many existing M&S tools are already dedicated to the design of such models. Then, it would be profitable to be able to reuse the models written in these tools to design complex system multimodels. The FMI standard, based on the export and import of models as software components, allows the exchange of models and the co-simulation between ODE and DAE based M&S software. The success of the FMI standard leads us to study the interests of a similar approach based on the DEVS formalism.*
Keywords*: Standardization, Co-simulation, DEVS, Reuse*

1 Introduction

L'un des enjeux actuels en Modélisation et Simulation (M&S) est la modélisation de systèmes complexes. Les systèmes complexes sont définis comme étant des ensembles composés d'un grand nombre d'entités hétérogènes et dont le comportement global émerge de l'interaction entre ces entités [7].

Les systèmes cyber-physiques, comme les réseaux électriques intelligents, sont des systèmes complexes composés de multiple entités provenant de différents domaines (ici réseaux électriques, réseaux de télécommunication et systèmes de décision) [12].

La modélisation de tels systèmes requiert de faire collaborer des experts de ces différents domaines qui vont chacun apporter, avec leur expertise spécifique, des éclairages différents mais complémentaires sur le système étudié. Cependant, cela nécessite de développer de nouvelles approches multi-perspectives et multi-domaines [8].

Face aux challenges de la M&S de systèmes complexes, la multi-modélisation et la co-simulation [4] semblent des approches prometteuses. La première consiste à développer des modèles pour chaque entité composant le système complexe et à les faire interagir pour former un multi-modèle. La seconde consiste à permettre la synchronisation et l'échange de données entre plusieurs simulateurs, chacun

gérant la simulation d'un modèle particulier. Ces approches nécessitent néanmoins (1) d'ouvrir les modèles pour permettre leur assemblage en multi-modèle et (2) d'ouvrir leur simulateur pour permettre leur interaction et leur co-simulation.

Dans l'hypothèse d'une généralisation de la co-simulation et d'une réutilisation des modèles existants, il faut s'accorder sur une norme. Nous proposons ici une première réflexion construite à partir de nos expériences sur MECSYCO[1] qui utilise à la fois DEVS (*Discrete EVent System specification*)[18] et FMI (*Functional Mockup Interface*)[2].

D'une part, dans le cas particulier des systèmes dynamiques représentables avec des ODE (*Ordinary Differential Equation*) et des DAE (*Differential Algebraic Equation*), la norme FMI a été créée. Cette norme, indépendante d'outils particuliers, permet l'échange de modèles et la co-simulation entre logiciels de M&S de systèmes dynamiques. Ainsi des experts de différents domaines peuvent travailler sur leurs outils dédiés et collaborer en échangeant leurs modèles.

D'autre part, le formalisme DEVS possède de nombreux avantages pour la multi-modélisation et la co-simulation, notamment l'intégration rigoureuse de composants hétérogènes.

Au regard du succès de la norme FMI et des avantages apportés par DEVS, nous proposons dans cet article le développement d'une norme similaire à FMI mais basée sur DEVS. Le développement d'une norme basée sur DEVS ne pouvant se faire qu'à partir d'une réflexion commune de la communauté DEVS, nous nous contenterons dans cet article d'expliciter les motivations et les intérêts d'une telle norme et de donner un exemple basé sur l'intergiciel MECSYCO.

L'article est organisé comme suit. La Section 2 rappellera les besoins liés à la multi-modélisation et co-simulation. Les Sections 3 et 4 présenteront respectivement le standard FMI et le formalisme DEVS, leurs intérêts et leurs limites. Ensuite, la Section 5 reprendra les éléments des sections précédentes pour motiver le développement d'une approche composant basée sur DEVS. La Section 6 présentera un cas d'exemple basé sur l'intergiciel MECSYCO. Enfin, l'approche sera discutée et des travaux similaires seront présentés en Section 7.

[1] Multi-agent Environment for Complex SYstem CO-simulation.

2 Besoins pour la M&S de systèmes complexes

La modélisation et simulation de systèmes complexes nécessite de faire collaborer des experts venant de différents domaines et utilisant des formalismes et des outils de M&S différents. Dans l'approche par multi-modélisation et co-simulation, il est nécessaire de pouvoir faire interagir des modèles écrits dans des formalismes différents et des simulateurs ayant des dynamiques différentes.

Au niveau technique et logiciel, de nombreux outils spécialisés existent déjà pour produire et simuler des modèles. Il est important de pouvoir réutiliser ces outils qui sont déjà maitrisés par les experts domaines, et surtout de pouvoir exploiter les modèles produits dans ces outils. Pour cela, il est nécessaire que les modèles soient ouverts, i.e. qu'ils acceptent des évènements extérieurs, et que les simulateurs soient ouverts, i.e. qu'il soit possible de contrôler la boucle de simulation pour intégrer des évènements. Il faut donc définir des standards logiciels pour réutiliser les modèles et les simulateurs disponibles dans les outils actuels.

En résumé, nous avons besoin de méthodes pour intégrer différents formalismes hétérogènes et de techniques pour permettre l'utilisation de modèles issus de différents logiciels, i.e. il faut un wrapping formel et un wrapping logiciel.

3 FMI

3.1 Présentation et principes

Le développement du standard FMI a été initié par Daimler AG dans le but de faciliter l'échange de modèles de simulation entre industriels [1]. L'objectif est donc avant tout pratique : faciliter la collaboration entre industriels

en leur permettant d'échanger leurs modèles.

Pour cela, le standard FMI repose sur des principes du génie logiciel, notamment l'utilisation de composant logiciel. Les modèles sont donc vus comme des boites noires contenant du code compilé appelées FMU (*Functionnal Mockup Unit*), et le standard définit les méthodes permettant d'interagir avec celles-ci.

FMI propose deux stratégies : *Model-Exchange (ME)* et *Co-simulation (CS)*. La première consiste à produire un modèle de système dynamique (défini par des équations différentielles, algébriques et/ou discrètes) ainsi qu'une interface pour évaluer ces équations au besoin. Ce modèle exporté sous forme de FMU peut alors être réutilisé dans un autre logiciel de M&S qui implémente un solveur pour le résoudre. La seconde consiste à produire un modèle de système dynamique associé à un solveur. Toujours sous la forme d'une FMU, cette fois-ci des fonctions permettront d'interagir avec le solveur intégré dans la FMU pour effectuer une co-simulation.

3.2 Architecture

Une FMU est un dossier zip contenant :

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

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

3.3 Limite

La norme FMI se concentre sur l'API logicielle et ne propose pas de solution d'intégration formelle. Aussi, la norme a été pensée pour les systèmes dynamiques écrits sous forme d'équations et n'est pas adaptée aux modèles évènementiels. Cette limite pour la gestion des évènements a déjà été soulevée et des solutions techniques d'amélioration du standard sont proposées [9].

4 DEVS

4.1 Présentation

DEVS [18] est un formalisme évènementiel proposé par Bernard P. Zeigler dans les années 70. Sa propriété d'universalité le place comme un formalisme pivot pour l'intégration de formalismes hétérogènes [11]. De nombreux travaux ont ainsi déjà montré l'intégration dans DEVS de formalismes évènementiels (Réseaux de Petri...), mais aussi de formalismes hybrides comme DEV&DESS [17] qui spécifient l'interaction entre modèles discrets et continus.

Plusieurs travaux ont déjà souligné l'intérêt du développement de standards basés sur DEVS [15]. Ces intérêts sont liés aux propriétés du formalisme.

Universalité et unicité : Tout système évènementiel est représentable par un modèle DEVS, et cette représentation est unique.

Fermeture par composition et algorithme abstrait : DEVS définit une structure de modèle couplé (un modèle composé de plusieurs sous-modèles DEVS) équivalente à un modèle atomique DEVS (permettant une construction hiérarchique) et la structure n'impacte pas les résultats de simulation ce qui garantit une exécution correcte.

Séparation entre le modèle, le simulateur et le cadre expérimental : DEVS définit comment représenter un modèle évènementiel, et fournit un algorithme abstrait pour le simuler.

4.2 Simulateur abstrait

La simulation de modèle DEVS est réalisée grâce à cinq fonctions [18] :

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

Ces 5 méthodes sont suffisantes pour assurer la synchronisation entre plusieurs modèles DEVS. De plus, la propriété d'universalité de DEVS nous assure que ce protocole est suffisant pour simuler n'importe quel modèle évènementiel. Cela appuie l'intérêt d'utiliser cette interface pour la co-simulation de systèmes évènementiels.

4.3 Limites

Depuis la création de DEVS, de nombreux outils de M&S l'utilisant sont apparus. Cependant, ces outils ne sont pas interopérables à cause des hétérogénéités logicielles [15]. Cela signifie que malgré un fort lien formel entre ces outils de M&S, le manque de consensus de développement empêche à l'heure actuelle de les utiliser en collaboration.

5 Proposition

5.1 Motivations

Nous avons d'une part, le formalisme DEVS qui possède de nombreux avantages pour la multi-modélisation et la co-simulation, notamment grâce à sa capacité à intégrer d'autres formalismes, mais qui ne fournit pas de standard logiciel pour souder sa communauté.

D'autre part, nous avons le standard FMI qui propose une API logicielle permettant à toute une communauté, les experts utilisant des ODE et des DAE, de collaborer en échangeant des modèles et en effectuant des co-simulations tout en préservant l'indépendance de chaque outil, mais qui ne fournit pas de solution d'intégration formelle.

5.2 Principe

Nous proposons donc de définir des règles simples pour faciliter, et à terme systématiser, les possibilités d'échanges de modèles entre outils de M&S basés sur DEVS. Ces règles s'organiseront selon 3 axes.

1) Isoler la description du comportement des modèles pour permettre leur export. Cette règle est déjà mise en avant par le formalisme DEVS lui-même grâce à la distinction entre le modèle et son simulateur.

2) Définir les signatures des méthodes permettant d'interagir avec les modèles et leurs simulateurs. Accorder les signatures des méthodes d'interaction permet d'avoir une vision homogène, au moins au niveau formel, des modèles lors des co-simulations.

3) Définir une structure de fichier XML qui définisse les interfaces des modèles (ports d'entrée et de sortie, paramètres, informations...).

Nous insistons ici sur le fait que ces règles doivent être définies par la communauté. Dans la suite, nous nous contentons d'illustrer l'intérêt et la faisabilité de leur mise en œuvre.

5.3 Intérêts d'une approche composant basée sur DEVS

Outre les avantages inhérents à DEVS, l'approche composant logiciel apporte les principales propriétés suivantes.

5.3.1 Indépendance à des technologies ou outils particuliers

La création de composant logiciel pose des contraintes techniques en fonction des langages. En contrepartie, cela ne nécessite pas d'autres technologies (réseau de communication, middleware, autres standards...) ni d'autres logiciels. Cela assure aussi une forme de simplicité en terme du nombre d'éléments à mettre en place (peu de fonctions à implémenter).

5.3.2 Indépendance des outils de M&S

Chaque outil pourra développer l'export de modèles, mais aussi l'import pour gérer les co-simulations dans son environnement propre. Cela permet notamment à chaque utilisateur de lancer les co-simulations dans le logiciel qui fournit les outils (visualisation, analyse...) les plus adaptés à ses besoins.

5.3.3 Collaboration

Le respect d'une interface de simulation simplifie les co-simulations. En effet, cela permet à chaque expert d'écrire le modèle dans l'outil le plus adapté, en assurant que le modèle sera utilisable au sein du multi-modèle lors de la co-simulation.

5.3.4 Vérification et reproductibilité

Permettre à des experts de simuler un modèle indépendamment de l'outil dans lequel il a été écrit simplifie aussi le travail de vérification des modèles. En effet, en exportant le modèle dans un format compréhensible pas ses pairs et leurs outils propres, l'expert ayant écrit le modèle leur permet de vérifier la reproductibilité de son expérience et de vérifier le comportement du modèle sur d'autres jeux de paramètre.

5.3.5 Souplesse et ouverture

L'approche composant est aussi une solution pour permettre aux outils existants de faire un pas vers la communauté DEVS en profitant des avantages du formalisme pour la co-simulation. En effet, même si un outil n'est pas basé sur DEVS, un wrapping formel peut être effectué avant l'export du modèle pour le rendre compatible. L'interface de simulation étant basée sur le protocole abstrait DEVS, elle est particulièrement adaptée aux simulateurs évènementiels.

6 Exemple d'implémentation

Le but de cette section est de mettre en lumière les problèmes et les besoins qui peuvent être rencontrés lorsqu'on souhaite utiliser un modèle DEVS en dehors du logiciel dans lequel il est créé.

Dans MECSYCO, nous intégrons des modèles issus d'autres logiciels en utilisant un wrapping DEVS. Dans cet exemple, nous illustrons l'effort à fournir pour obtenir un composant logiciel DEVS, i.e. un programme manipulable par un autre simulateur DEVS, depuis MECSYCO.

Pour cela nous allons (1) définir un simulateur DEVS classique en Java, (2) exporter des modèles depuis MECSYCO et (3) simuler ces modèles dans le simulateur (1). Les signatures des méthodes d'interaction avec le modèle seront définies avec une interface Java, une description XML du modèle sera fournie.

Dans notre cas, nous travaillons avec le système de Lorenz dans lequel l'évolution de chaque variable est décrite dans un modèle. Ici, nous réutilisons les modèles déjà écrits dans NetLogo [10].

6.1 MECSYCO

MECSYCO[2] [3] est un intergiciel de co-simulation basé sur DEVS. MECSYCO repose sur une stratégie intégrative, i.e. le but du logiciel n'est pas de créer des modèles mais de permettre l'intégration de modèles issus d'autres logiciels pour les faire co-simuler. Cette démarche intégrative se base formellement sur le wrapping DEVS.

[2] mecsyco.com

L'intergiciel existe en version Java et C++ et des wrappers sont disponibles pour FMI [13], NetLogo [6], NS3 et Omnet++ [12]

Pour l'instant, le travail de wrapping DEVS est effectué dans MECSYCO. Le développement d'une approche composant peut permettre d'effectuer ce travail directement dans les logiciels produisant les modèles. Cela permettrait aussi d'utiliser les modèles déjà intégrés dans MECSYCO dans d'autres outils de M&S.

6.2 Étapes de l'exemple

1. Écriture d'une interface DEVS en Java (Figure 1), et d'une classe Java *Event* ne contenant qu'un temps et une donnée, elle est destinée à représenter les évènements échangés entre les modèles DEVS.

```
public void initialize();
public void processInternalEvent(double time);
public void processExternalInputEvent(String port,
                                      Event<?> anEvent);
public double getNextInternalEventTime();
public Event<?> getExternalOutputEvent(String port);
```

FIGURE 1 : Signature des méthodes de l'interface DEVS utilisée.

2. Écriture d'un simulateur/coordinateur DEVS basique compatible avec l'interface de modèle définie précédemment.

3. Définition de la structure du document XML décrivant les modèles DEVS (ports, paramètres). Voici un schéma simplifié des **balises** et des *attributs* utilisés pour décrire le modèle :

- **ModelDescription** *name="LorenzX"*
  - **ModelInterface**
    - **Parameter** *name="a"*
      -**Real** *default="1.0"*
    - ...
    - **Input** *name="y"*
      -**Real** *start="1.0"*
    - ...
    - **Output** *name="x"*
      -**Real** *start="1.0"*
  - ...
  - **ModelSimulator**
    - **SimulVariables**
      - **SimulVariable**
        name="timeStep"
        -**Real** *default="0.01"*
      - ...

L'idée est de spécifier une structure de base définissant les éléments du modèle qui seront modifiables lors des co-simulations. Mais le document pourra aussi contenir des éléments nécessaires aux modèles. Dans le cas de NetLogo par exemple, la notion de port n'existe pas mais un interpréteur de commande est présent. Le document XML est alors utilisé pour associer des commandes aux ports et aux paramètres du modèle [6].

4. Export d'un modèle depuis MECSYCO. Dans MECSYCO, les modèles sont représentés par des artéfacts de modèle : objet Java permettant de manipuler un modèle issus d'un autre logiciel grâce aux 5 méthodes du protocole de simulation DEVS. Les modèles sont donc déjà distincts du simulateur. Le travail a consisté en la création d'une classe faisant le lien entre l'interface DEVS définie, et l'interface de l'artéfact modèle. Cette classe est ensuite exportée sous forme de Jar avec, un document XML définissant l'interface d'un modèle, et un modèle issu d'un simulateur (dans l'exemple un modèle NetLogo).

5. Utilisation du simulateur/coordinateur pour effectuer une co-simulation entre 3 modèles NetLogo représentant un système de Lorenz (un exemple utilisant des FMU a aussi été réalisé).

6.3 Bilan

La réalisation que nous avons faites des différentes étapes soulève plusieurs points intéressants :

1) Facilité d'implémentation : Dans notre cas, les modèles MECSYCO ayant déjà une structure basée sur DEVS, l'adaptation à l'interface basique définie a été très simple. De notre expérience, cette étape de wrapping DEVS peut être nettement plus compliquée [12].

2) Le choix des types : Nous avons utilisé dans notre exemple des types simples (réel, entier,

chaine de caractère et booléen) puisqu'ils sont communs à toutes les plateformes. Ce choix étant limitant, des types complexes (tableau, liste, table) devront être définis et standardisés.

3) La structure du fichier XML : Dans notre exemple, les descriptions des fichiers NetLogo contiennent des informations supplémentaires en plus des ports et paramètres. Le document de description doit donc aussi permettre l'ajout d'informations supplémentaires particulières à chaque outil.

4) Initialisation : Dans notre cas, le fichier XML sert autant à la configuration du modèle au moment de son chargement, qu'à la description de son interface. Il sera nécessaire de définir des méthodes pour initialiser les paramètres (dans MECSYCO, nous utilisons par exemple des tables).

5) Dépendance : Pour utiliser le modèle exporté, il faut rendre accessible les bibliothèques pour exécuter le modèle (dans notre cas, les Jar rendant les modèles NetLogo exécutables). Cet aspect n'a pas été considéré dans l'export.

7 Discussion et travaux connexes

Contrainte technique
Les outils de M&S sont écrits dans différents langages (C, C++, Java, Python...) ce qui posera des contraintes au niveau technique/logiciel, i.e. gestion des hétérogénéités de langage (Java/C++ etc) et de plateformes (Windows/Unix). Cependant des solutions existent déjà pour faire le pont entre les logiciels écrits dans différents langages (par exemple JavaFMI qui fait le pont entre les FMU et Java, PyFmi qui permet de simuler des FMU à partir de Python...). Nous supposons donc que les solutions techniques existent déjà.

Standardisation basée sur DEVS
Plusieurs travaux ont déjà souligné l'intérêt du développement de standards basés sur DEVS [15], deux voies sont proposées.

1. La standardisation de la représentation d'un modèle DEVS [16]. L'idée est de trouver un moyen de représenter un modèle DEVS indépendamment d'un langage de programmation particulier, puis de pouvoir transformer cette représentation standard en une représentation interprétable par un simulateur. DEVSML [5] utilise par exemple une représentation XML d'un modèle DEVS qui peut ensuite être utilisée sur plusieurs plateformes. Dans notre cas, nous ne travaillons pas sur la manière de représenter le modèle DEVS lui-même, mais seulement sur la manière d'interagir avec un modèle, ce qui permet notamment d'intégrer des modèles sans les réécrire.

2. La standardisation de l'interface de simulation DEVS [14]. La proposition de cet article rentre dans cette catégorie où la représentation du modèle DEVS n'est pas importante et où seule importe l'interface d'interaction avec les modèles générés. La plupart des approches existantes (comme par exemple DEVS/SOA) utilisent des outils et des approches liées aux services web pour permettre l'interaction entre simulateurs basés sur DEVS. Ces approches visent à créer un intergiciel capable de coordonner différents logiciels de M&S basés sur DEVS pour réaliser des co-simulations. La différence par rapport à ces approches et que nous ne cherchons pas à interfacer différents outils et à les coordonner mais plutôt à permettre l'échange de modèles entre ces différents outils. Chaque outil pourra servir de coordinateur entre les différents modèles DEVS intégrés et utiliser ses propres technologies.

Standard FMI
Notre proposition s'inspire du succès du standard FMI. Bien que ces deux approches soient similaires en terme d'architecture logicielle et répondent aux mêmes problématiques de collaboration et de partage de modèles, elles ne sont pas opposées mais complémentaires. En effet, alors que la norme FMI propose une interface adaptée aux modèles de systèmes continus ou hybrides, l'interface basée sur DEVS est naturellement plus adaptée aux modèles évènementiels.

8 Conclusion

L'objectif de cet article est d'engager la réflexion sur la création de règles de

développement, basées sur DEVS, qui facilitent l'utilisation de logiciels existants pour le développement rigoureux de multi-modèles et leur co-simulation. Ces règles comporteraient notamment une définition d'interface logicielle pour manipuler les composants logiciels avec un coordinateur DEVS, et une structure XML pour détailler l'interface des modèles. L'adoption de règles communes par la communauté DEVS permettrait alors de développer des composants logiciels DEVS réutilisables.

Ce travail est une proposition qui fait écho aux différents travaux existants proposant des standardisations à base du formalisme DEVS. Le grand avantage de l'approche est de préserver l'indépendance des outils de M&S tout en leur permettant d'échanger des modèles.

Par exemple, pour MECSYCO, s'accorder sur une interface DEVS permettrait d'exporter les modèles déjà intégrés pour les utiliser simplement dans n'importe quel outil basé sur DEVS, évitant de recommencer le travail d'encapsulation (ou éventuellement vérifier sa correction).

Références

[1] FMI for Model-Exchange and Co-Simulation v2.0.

[2] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, H. Elmqvist, A. Junghanns, J. Mauß, M. Monteiro, T. Neidhold, D. Neumerkel, et al. The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference; March 20th-22nd*, number 063, pages 105–114. Linköping University Electronic Press, 2011.

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

[4] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe. Co-simulation : State of the art. *arXiv preprint arXiv :1702.00686*, 2017.

[5] S. Mittal and J. L. Risco-Martín. DEVSML studio : a framework for integrating domain-specific languages for discrete and continuous hybrid systems into DEVS-based M&S environment. In *Proceedings of the Summer Computer Simulation Conference*, page 41. Society for Computer Simulation International, 2016.

[6] T. Paris, L. Ciarletta, and V. Chevrier. Designing co-simulation with multi-agent tools : a case study with NetLogo (Accepted). In *Proceedings of the 15th European Workshop on Multi-Agent Systems. EUMAS*, 2017.

[7] E. Ramat. Introduction à la simulation : principaux concepts. In *Modélisation et Simulation Multi-Agent : application pour les Sciences de l'Homme et de la Société*, pages 37–60. 2006.

[8] M. D. Seck and H. J. Honig. Multi-perspective modelling of complex phenomena. *Computational & Mathematical Organization Theory*, pages 1–17, 2012.

[9] J.-P. Tavella, M. Caujolle, C. Tan, G. Plessis, M. Schumann, S. Vialle, C. Dad, A. Cuccuru, and S. Revol. Toward an hybrid co-simulation with the fmi-cs standard. 2016.

[10] S. Tisue and U. Wilensky. Netlogo : A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004.

[11] H. L. Vangheluwe. DEVS as a common denominator for multi-formalism hybrid systems modelling. In *Computer-Aided Control System Design, 2000. CACSD 2000. IEEE International Symposium on*, pages 129–134. IEEE, 2000.

[12] J. Vaubourg, V. Chevrier, and L. Ciarletta. Co-simulation of IP network models in the Cyber-Physical systems context, using a DEVS-based platform. In *Proceedings of the 19th Communications & Networking Symposium*, page 2. Society for Computer Simulation International, 2016.

[13] J. Vaubourg, Y. Presse, B. Camus, C. Bourjot, L. Ciarletta, V. Chevrier, J.-P. Tavella, and H. Morais. Multi-agent multi-model simulation of smart grids in the MS4SG project. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 240–251. Springer, 2015.

[14] G. Wainer, K. Al-Zoubi, O. Dalle, D. R. C. Hill, S. Mittal, J. L. R. Martin, H. Sarjoughian, L. Touraille, M. K. Traoré, and B. P. Zeigler.

Standardizing DEVS Simulation Middleware. In *Discrete-Event Modeling and Simulation : Theory and Applications*, page 459. 2010.

[15] G. Wainer, K. Al-Zoubi, D. Hill, S. Mittal, J. Martín, H. Sarjoughian, L. Touraille, M. Traoré, and B. Zeigler. An Introduction to DEVS Standardization. In G. Wainer and P. Mosterman, editors, *Discrete-Event Modeling and Simulation*, volume 20101361, pages 393–425. CRC Press, Dec. 2010. DOI : 10.1201/b10412-21.

[16] G. Wainer, K. Al-Zoubi, D. Hill, S. Mittal, J. Martín, H. Sarjoughian, L. Touraille, M. Traoré, and B. Zeigler. Standardizing DEVS Model Representation. In G. Wainer and P. Mosterman, editors, *Discrete-Event Modeling and Simulation*, volume 20101361, pages 427–458. CRC Press, Dec. 2010. DOI : 10.1201/b10412-22.

[17] B. P. Zeigler. Embedding DEV&DESS in DEVS. In *DEVS Integrative Modeling & Simulation Symposium*, pages 125–132, 2006.

[18] B. P. Zeigler, H. Praehofer, and T. G. Kim. *Theory of modeling and simulation : integration discrete event and continuous complex dynamic systems*. Academic press, 2000.