HAL
open science

Environnement de coopération de simulation pour la conception de systèmes cyber-physiques

Gilles Lasnier, Janette Cardoso, Claire Pagetti, Pierre Siron

To cite this version:

Gilles Lasnier, Janette Cardoso, Claire Pagetti, Pierre Siron. Environnement de coopération de simulation pour la conception de systèmes cyber-physiques. Journal Européen des Systèmes Automatisés (JESA), 2013, vol. 47, pp. 13-27. 10.3166/jesa.47.13-27 . hal-00926971

HAL Id: hal-00926971

https://hal.science/hal-00926971

Submitted on 10 Jan 2014

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.

Open Archive Toulouse Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in: http://oatao.univ-toulouse.fr/
Eprints ID: 10430

To link to this article: DOI: 10.3166/jesa.47.13-27
URL: http://dx.doi.org/10.3166/jesa.47.13-27

To cite this version: Lasnier, Gilles and Cardoso, Janette and Pagetti, Claire and Siron, Pierre *Environnement de coopération de simulation pour la conception de systèmes cyber-physiques.* (2013) Journal Européen des Systèmes Automatisés (JESA), vol. 47 (n° 1-3). pp. 13-27. ISSN 1269-6935

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@inp-toulouse.fr

Environnement de coopération de simulation pour la conception de systèmes cyber-physiques

Gilles Lasnier [1], Janette Cardoso [1], Claire Pagetti [2], Pierre Siron [1,2]

1 ISAE, Toulouse, France
2 ONERA, Toulouse, France / IRIT-ENSEEIHT, Toulouse, France
prenom.nom@{isae,onera}.fr

RÉSUMÉ. *La conception de systèmes cyber-physiques (CPS) est une activité complexe qui requiert l'utilisation de plusieurs méthodes et outils pendant le processus de développement. L'objectif du travail présenté est de fournir un moyen de simuler plusieurs types de modèles réalisés à plusieurs niveaux dans la phase de conception. Nous avons développé un environnement de co-simulation permettant la coopération de deux outils de simulation open source : Ptolemy II et HLA/CERTI. Ptolemy II permet la modélisation hiérarchique de systèmes hétérogènes tandis qu'HLA/CERTI permet des simulations distribuées interopérables (parfois en intégrant des éléments matériels). Dans le papier, nous détaillons les points communs et différences sémantiques de la gestion du temps. Trois nouveaux objets Ptolemy : un attribut* HlaManager *ainsi que deux acteurs,* HlaPublisher *et* HlaSubscriber*, ont été développés afin de réaliser l'interface entre les deux outils de simulation. Une étude de cas est détaillée à la fin du papier.*

ABSTRACT. *This paper describes a collaborative simulation framework that allows to combine Ptolemy II simulations with a distributed simulation provided by a IEEE HLA compliant tool named CERTI. Both simulation environments have different advantages. Ptolemy permits to express hierarchical composition of heterogeneous models in a same model with a well defined semantics. HLA/ CERTI provides an infrastructure for integrating various simulation environments through a distributed management entity. Simulation entities, so-called federates, are connected through a shared Run-Time Infrastructure (RTI). After presenting the characteristics of both frameworks, the difference and similitudes of them are discussed. The cooperation is done by introducing a new Ptolemy attribute, called* `HlaManager`*, and two new actors* `HlaPublisher` *and* `HlaSubscriber`*. These components perform the interface between Ptolemy and HLA/CERTI environments. A simple case study is presented for illustrating our approach.*

MOTS-CLÉS : *systèmes cyber-physiques, simulation, coopération d'outils.*

KEYWORDS: *cyber-physical systems, simulation, co-simulation.*

1. Introduction

1.1. Contexte

La conception de systèmes cyber-physiques (CPS) est une activité complexe qui requiert l'utilisation de plusieurs méthodes et outils pendant le processus de développement. Si de plus, le système cyber-physique est conçu par plusieurs équipes et sous-traitants, il faut appliquer une méthodologie propre et non ambiguë. Les approches basées « modèle » sont une réponse classique à cette problématique. En effet l'utilisation de modèle facilite la spécification, la communication entre les personnes impliquées, la vérification formelle de propriété, la validation et même éventuellement la génération de code.

Pour les systèmes de contrôle/commande par exemple, l'environnement Matlab/Simulink (MathWorks, 2013) est souvent utilisé pour spécifier la dynamique de vol et le contrôleur. Une fois les modèles réalisés, il est possible de simuler le comportement et de vérifier la stabilité et la robustesse du contrôle en observant les traces fonctionnelles. Le contrôleur est alors traduit en code de bas niveau (manuellement ou automatiquement) puis exécuté sur la cible réelle (par exemple, un processeur simple ou une plateforme distribuée). Les analyses de l'exécution sur la cible arrivent tardivement dans le processus de développement. Il est donc essentiel de proposer des méthodes et outils permettant d'analyser le plus tôt possible les comportements (même partiels) futurs sur la cible, en tenant compte par exemple de l'interaction avec des capteurs / actionneurs, ou encore du placement sur la plateforme distribuée.

Une telle activité repose sur la connaissance de plusieurs domaines transverses (logiciel, matériel ...) et nécessite l'utilisation de modèles hétérogènes (continus, discrets ...) L'utilisation seule de temps pires est insuffisante pour valider le système et l'utilisation de simulation est une solution complémentaire pour analyser les comportements en mélangeant des spécifications de haut niveau et des éléments de bas niveau. Prenons l'exemple d'un système de commande de vol composé de deux sous-fonctions distribuées sur deux calculateurs distants reliés par un réseau Ethernet déterministe. Pour déterminer le comportement réel sur la plateforme, il faut décrire le comportement fonctionnel de chaque fonction mais également le comportement temporel du réseau pour délivrer les données entre les deux fonctions. Une autre capacité de la simulation est de connecter ces sous-fonctions aux entrées lues par des capteurs et aux sorties produites par actionneurs réels.

1.2. Contribution

L'objectif de ce travail est de fournir un moyen pour simuler des modèles de différentes natures réalisés à différents niveaux du processus de développement. Pour ce faire, nous avons développé un environnement de co-simulation permettant la coopération de deux outils de simulation open source : Ptolemy II et HLA/CERTI. Chaque approche se place effectivement à différents niveaux d'abstraction.

Ptolemy II (Ptolemaeus, 2014) est un environnement open source de modélisation et de simulation de modèles hiérarchiques hétérogènes, développé à l'université de Californie à Berkeley. Cet outil est particulièrement bien adapté pour modéliser des systèmes cyber physiques (Derler *et al.*, 2012) en utilisant plusieurs modèles de calcul (models of computation MoC) tels que continu ou à événements discrets.

Le standard IEEE HLA (High-Level Architecture) (Department of Defense (DoD) Specifications, 1998a ; IEEE, 2010) est dédié à la simulation distribuée. Un système est vu comme une *fédération* regroupant plusieurs *fédérés* communicant par des mécanismes de publications / souscription. Cette décomposition par fédérés permet de combiner plusieurs types de composants comme des modèles de simulation, des codes fonctionnels (en C++ ou Java) ou des équipements matériel. Le bénéfice principal de HLA est donc l'interopérabilité et la réutilisation. Nos expérimentations et environnement sont basés sur l'outil open source HLA/CERTI (Noulard *et al.*, 2009 ; ONERA, 2013).

Notre approche permet d'exécuter une simulation Ptolemy comme un fédéré d'une fédération HLA. La coopération de ces deux environnements permet d'utiliser les avantages de chacun : hétérogénéité fournie par Ptolemy (possibilité d'utiliser plusieurs modèles de calcul) et interopérabilité fournie par HLA (possibilité d'utiliser plusieurs modèles de simulation, des codes et des équipements physiques).

1.3. Travaux connexes

L'interopérabilité est un besoin réel dans les simulations à événements discrets et cette importance est soulignée par les nombreux travaux sur la co-simulation. De nombreux standards ont été définis afin de normaliser l'interopérabilité d'applications distribuées en général : de la simple invocation distante de procédures (Microsystems, 1988), à l'utilisation d'objets répartis CORBA (OMG, 2004). Le standard HLA repose sur le paradigme de passage par message et définit spécifiquement des services de haut niveau pour la gestion de l'avancée du temps. Les algorithmes implantés assurent qu'il n'y a pas de boucle de causalité dans les simulations HLA. En effet, chaque fédéré déclare un *lookahead* qui définit un horizon temporel pendant lequel il n'émettra aucun message. Plus le *lookahead* est petit, plus il y aura d'échanges de message. Lorsque le *lookahed* est nul, des algorithmes assurent l'absence d'interblocages. HLA répond donc davantage à nos besoins de co-simulation.

Il existe plusieurs outils de coopération basés sur HLA et deux d'entre eux ont des points communs avec notre approche. Les auteurs de (Hadj-Amor, Soriano, 2012) ont développé des connexions entre HLA et Modelica (Modelica Association, 2012) afin de réaliser des prototypages virtuels de systèmes mécatroniques. Ceux de (ForwardSim Inc. Simulation and Technologies, 2013) ont implanté une coopération entre HLA et MATLAB/Simulink. Cette coopération peut se faire selon deux modes : soit génération d'un fédéré en C++ à partir d'un modèle MATLAB/Simulink ; soit utilisation d'un bloc HLA dans les modèles MATLAB/Simulink permettant l'interaction entre l'environnement MATLAB/Simulink et une fédération. Les solutions proposées par

MATLAB/Simulink sont payantes et aucun détail sémantique de leurs implantations n'est disponible dans la littérature. Le deuxième mode est similaire à ce que nous avons développé, à la différence que notre environnement est open source et que la sémantique de gestion des temps de simulation est formulée.

Un consortium composé d'industriels et d'académiques travaille depuis quelques temps à la définition d'un standard pour unifier les interfaces des différents outils de simulation et simplifier les coopérations. Le standard s'appelle FMI (pour Functional Mock-Up Interface for Model Exchange and Co-Simulation) et un certain nombre de concepts ont été publiés dans (Modelisar, 2012 ; FMI Development Group, 2012). L'objectif est donc de fournir des règles d'interface et de certains comportements mais aucune coopération réelle n'est implantée, contrairement à notre travail. Le contexte est également plus large car le consortium s'intéresse à d'autres problématiques qui ne se posent pas entre les outils considérés dans ce papier. Les auteurs de (Awais *et al.*, 2013) ont montré que HLA est un standard adapté pour gérer l'interopérabilité, ce à quoi nous adhérons totalement. Ils montrent comment réaliser une coopération entre des outils conformes à FMI en s'intégrant à une fédération HLA.

Le papier est organisé comme suit. Les sections 2 et 3 décrivent les environnements de simulation HLA et Ptolemy II. La section 4 décrit l'environnement de co-simulation développé. La coopération est illustrée au travers d'un cas d'étude complet : un modèle producteur-consommateur centralisé est transformé en trois fédérés HLA et le comportement dans les deux cas est équivalent, voir section 4.3. Enfin nous concluons et donnons certaines perspectives.

2. Simulation distribuée avec HLA

Cette section présente brièvement le standard HLA et la manière dont une simulation est implantée. Nous décrivons ensuite le sous-ensemble de services HLA, essentiellement de gestion du temps, nécessaire à l'intégration de modèles Ptolemy dans une fédération.

2.1. Présentation générale de HLA

Le standard HLA (High-Level Architecture) (Department of Defense (DoD) Specifications, 1998a ; IEEE, 2010) décrit un ensemble de règles et de services pour réaliser des simulations à événements discrets distribuées. Ce genre de simulation est utile pour les entraînements et l'ingénierie des systèmes. L'approche favorise la réutilisabilité et l'interopérabilité. Dans la terminologie HLA, le système est vu comme une *fédération* qui est une collection de *fédérés*, i.e. des entités de simulation réalisant une suite de calculs interconnectés par une *Run-Time Infrastructure* (RTI). Le RTI est le middleware (intergiciel) sous-jacent dont le rôle est d'assurer le bon déroulement de la simulation. La figure 1 décrit l'architecture globale d'une simulation HLA.

Le standard HLA définit :

1. une interface de spécification précisant l'ensemble de services requis pour gérer les fédérés et leurs interactions. Par exemple, l'interface décrit comment un fédéré peut joindre ou créer une fédération ;

2. un template de modèle objet (basé sur le standard OMT (Department of Defense (DoD) Specifications, 1998b)) fournissant un environnement commun de descriptions des communications entre simulations HLA. Pour chaque fédération, un Federation Object Model (FOM) décrit les objets partagés, les attributs et les interactions.

3. un ensemble de règles décrivant les responsabilités des fédérés et de la fédération. Par exemple, une des règles précise que *tous les échanges de données entre fédérés doivent passer par le RTI.*

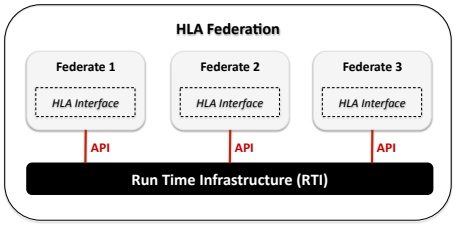


Figure 1. Architecture d'une fédération HLA

Tableau 1. Synthèse des services HLA

Classes	Services	Acronyme	Description (informelle)	Invocation
Federation	createFederationExecution()		création d'une fédération	normal
	joinFederationExecution()		rejoindre une fédération	normal
	resignFederationExecution()		quitter une fédération	normal
	destroyFederationExecution()		détruire une fédération	normal
	registerFed...Sync...Point()		définir un point de synchronisation	normal
	sync...PointReg...Succeeded()		acquittement définition du point de synchronisation	callback
	announceSynchronizationPoint()		attendre un point de synchronisation	callback
	synchronizationPointAchieved()		fin du point de synchronisation	normal
	federationSynchronized()		fédération synchronisée	callback
	tick()	TICK	récupération des callbacks depuis le RTI	normal
Declaration	publishObjectClass()		publication d'un objet	normal
	subscribeObjectClassAttributes()		abonnement à un objet	normal
	unsubscribeObjectClass()		désabonnement à un objet	normal
	unpublishObjectClass()		arrêt de la publication d'un objet	normal
Object	registerObjectInstance()		définir une instance d'objet	normal
	discoverObjectInstance()		découverte d'instance d'objet	callback
	updateAttributeValues()	UAV	mise à jour d'une valeur à la fédération	normal
	reflectAttributeValues()	RAV	réception d'une mise à jour de valeur par la fédération	callback
Time	enableTimeRegulation()		déclaration d'un fédéré de type régulateur	normal
	timeRegulationEnabled()		succès de la déclaration du fédéré en tant que régulateur	callback
	enableTimeConstrained()		déclaration d'un fédéré de type contraint	normal
	timeConstrainedEnabled()		succès de la déclaration du fédéré en tant que contraint	callback
	timeAdvanceRequest()	TAR	demande d'avancée à une date déterminée	normal
	nextEventRequest()	NER	demande d'avancée à la date du prochain événement	normal
	timeAdvanceGrant()	TAG	notification que la demande d'avance dans le temps est accordée	callback

Les services HLA sont organisés en six classes de gestion utilisées pendant la durée de vie d'un fédéré. Le tableau 1 décrit le sous ensemble de services HLA nécessaire pour notre approche. Pour chaque service, nous donnons une description informelle de son utilisation. Le lecteur peut trouver une description complète de ces services dans (IEEE, 2010 ; Department of Defense (DoD) Specifications, 1999).

Les services considérés ne touchent que quatre classes : (1) gestion de la fédération ; (2) gestion des déclarations, c'est-à-dire la déclaration des objets et des attributs des objets que chaque fédéré publiera ou auquel chaque fédéré s'abonnera ; (3) gestion des objets, c'est-à-dire la manière dont les fédérés mettent à jour ou reçoivent les attributs depuis la fédération ; (4) gestion du temps, c'est-à-dire l'ensemble des mécanismes nécessaires à l'avancement cohérent du temps. Une partie des informations de gestion est statiquement stockée dans le fichier FED (*Federation Execution Data*) nécessaire au RTI.

2.2. Gestion du temps

Les services de gestion du temps d'HLA permettent de réaliser des simulations distribuées déterministes et reproductibles. Chaque fédéré possède un temps logique et le RTI s'assure de la coordination des fédérés en avançant de manière cohérente les temps logiques de chaque fédéré. Une mise en œuvre possible des mécanismes de synchronisation est basée sur l'algorithme de Chandy et Misra (Chandy, Misra, 1979). Selon les besoins de performances, d'autres algorithmes peuvent être utilisés. Le temps logique est utilisé pour assurer que les fédérés observent les événements dans le même ordre (Fujimoto, 1996). Le temps logique est équivalent au *temps de simulation* dans la littérature classique sur la simulation à événement discret.

2.2.1. Politiques temporelles des fédérés

L'utilisation des services de gestion du temps est optionnelle, mais quand ils sont utilisés il faut préciser dans le fichier FED les politiques temporelles des fédérés. Il peut être important dans certains cas de contraindre l'avancement du temps d'un fédéré en fonction du progrès temporel d'un autre fédéré. Un fédéré *régulateur* participe à l'avancement du temps. Un fédéré *contraint*, à l'inverse, suit l'avancement du temps imposé par les autres fédérés. Il est également possible d'être *régulateur et contraint* ou *ni régulateur ni contraint*.

Le standard HLA distingue deux types d'événements : ceux enrichis avec une estampille temporelle (TSO - Time-Stamp Ordered) et ceux qui ne le sont pas. Les événements qui ne sont pas délivrés avec un TSO sont immédiatement disponibles pour la fédération. Pour les autres, le RTI ne les délivre qu'en fonction de l'avancée du temps, assurant ainsi la cohérence et la causalité. Ainsi, un fédéré récepteur ne peut recevoir un événement avec une estampille temporelle que si la valeur est entre le temps courant et la date de la prochaine avancée.

Un événement est délivré avec un TSO par le RTI à trois conditions : (1) le fédéré émetteur est régulateur ; (2) le fédéré récepteur est contraint ; (3) la politique temporelle de l'événement (attribute update) est *TIMESTAMP*. Les fédérés régulateurs génèrent donc des événements enrichis avec des estampilles temporelles (éventuellement dans le désordre). A un instant logique t, les événements produits ne peuvent être estampillés qu'avec une valeur supérieure ou égale à t + *lookahead*, où *lookahead* est une durée prédéterminée.

2.2.2. *Progression temporelle*

Les demandes d'avancée temporelle par les fédérés se font par deux services particuliers: *TAR* et *NER*. *TAR*, pour *timeAdvanceRequest* (voir le tableau 1), est utilisé pour implanter des progressions dirigées par le temps. *NER*, *nextEventRequest*, est utilisé pour implanter des progressions dirigées par les événements. La sémantique de ces deux services est partiellement montrée dans la figure 2.

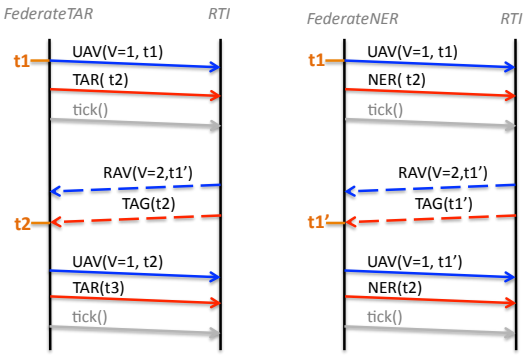


Figure 2. Services d'avancée temporelle : a. TAR b. NER

Dans la figure 2a, FederateTAR produit un événement à la date t_1, en appelant le service *updateAttributeValue* (*UAV*). Pendant le même instant logique, il calcule le prochain instant logique t_2 où il aura quelque chose à faire. Il demande une avancée temporelle au RTI jusqu'à t_2 en invoquant *TAR*(t_2). Jusqu'à la réception du callback *TAG*(t_2), son temps logique est bloqué à t_1. FederateTAR reçoit entre temps *RAV(v,*t_1*')* avec $t'_1 \in [t_1, t_2[$. Il peut s'il le souhaite réaliser des calculs avec les valeurs reçues avec *RAV*. Le fédéré reçoit enfin *TAG*(t_2) et peut avancer son temps logique à t_2.

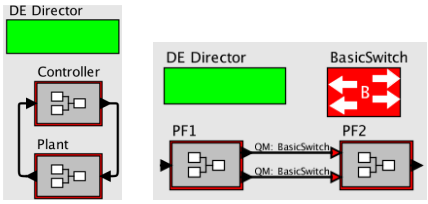
Dans la figure 2b, FederateNER produit également un événement à t_1 et demande une avancée temporelle à t_2 mais en invoquant *NER*(t_2). La réception de *RAV(v,*t_1*')* est immédiatement suivie par *TAG*(t'_1) permettant ainsi à FederateNER d'avancer son temps logique à t'_1. De nouveaux événements peuvent être produits à partir de t'_1, ce qui n'était pas le cas précédemment. Dans l'exemple, la fédération publie une nouvelle valeur pour V et réémet ensuite *NER*(t_2).

3. Environnement Ptolemy II

3.1. Présentation générale de Ptolemy II

Ptolemy II (Ptolemaeus, 2014) est un environnement de modélisation et de simulation open source de systèmes *hétérogènes*. La modélisation est orientée acteurs. Un acteur est une entité exécutable munie d'une interface contenant des *ports*, représentant les points de communication, et des paramètres, utilisés pour configurer l'acteur. La modélisation est hiérarchique : un acteur peut donc être *atomique* (élémentaire)

ou *composite* (hiérarchique). Pour chaque niveau de hiérarchie, un acteur particulier appelé le *directeur* impose la sémantique du modèle, imposant ainsi les règles d'exécution, de communication et de mise à jour de l'état interne. Plusieurs directeurs sont disponibles en Ptolemy, chacun correspondant à un modèle de calcul (model of computation - MoC): événement discret (DE), temps continu (CT), synchronous reactive (SR), synchronous data-flow (SDF), etc. Grâce à la composition hiérarchique, il est possible de combiner des acteurs gouvernés par des directeurs différents et d'obtenir ainsi des modèles hétérogènes.



a) CPS b) Contrôleur distribué

Figure 3. Modèles Ptolemy

Un exemple de modèle Ptolemy composé de deux acteurs composites est montré dans la figure 3a. `Plant` contient un directeur continu et `Controller` contient un directeur discret. Le directeur du niveau supérieur est DE ainsi toutes les données transitant à ce niveau sont discrètes. Les interfaces d'un modèle de calcul vers un autre sont réalisées par des acteurs spécifiques, par exemple, `Sampler` (de Continuous vers DE) et `ZeroOrderHold` (de DE vers Continuous).

La figure 3b montre un modèle permettant de simuler le comportement d'une plateforme distribuée où deux acteurs composites, `PF1` et `PF2`, échangent leurs données à travers un réseau, réalisé par un troisième acteur composite `BasicSwitch`. Pour coder `BasicSwitch`, il faut utiliser l'acteur Quantity Manager (QM) (Derler *et al.*, 2012) permettant de modéliser le comportement de protocoles réseau tels que CAN ou AFDX.

3.2. Sémantique

3.2.1. Au niveau acteur

Un acteur s'exécute en trois phases illustrées dans la figure 4 : une phase d'initialisation *setup*, une séquence d'*iterations*, et une terminaison *wrapup*.

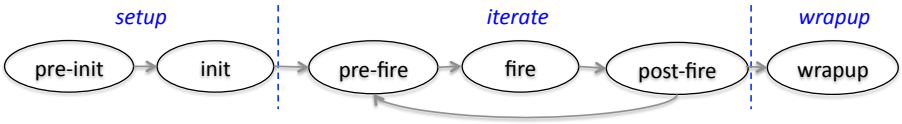


Figure 4. Cycle de vie d'un acteur Ptolemy

La phase de *setup* est décomposée en 2 étapes. La *pré-initialisation* est réalisée une unique fois au tout début de l'exécution de l'acteur. On y précise les actions qui modifieront ou influenceront l'architecture (instanciation d'acteurs internes, résolution de typage). L'*initialisation* se fait juste après : il faut initialiser les paramètres, se placer dans un état initial et éventuellement envoyer des messages d'initialisation dans les ports de sortie.

Le *prefire* est optionnel et consiste à vérifier des pré-conditions pour assurer la terminaison de l'itération. Le *fire* est l'exécution proprement dite consistant à lire les entrées, réaliser les calculs et produire les sorties. L'envoi et la lecture de données sur les ports se fait à l'aide des primitives *get* et *put*. La sémantique de ces méthodes dépend du directeur utilisé, ainsi la communication d'un acteur composite vers l'extérieur doit obéir à la sémantique du directeur du niveau supérieur. Le *postfire* consiste à mettre à jour les états persistants du modèle. Ce calcul peut nécessiter de calculer un point fixe.

L'objet *attribut* est une entité intermédiaire entre acteur et directeur qui permet de paramétrer statiquement ou dynamiquement les objets d'un modèle. Le cycle de vie d'un attribut ne contient que les deux phases *setup* et *wrappup*. L'introduction d'un *attribut* facilitera la synchronisation des temps entre Ptolemy II et HLA/CERTI sans modifier l'implantation d'un Directeur.

3.2.2. Au niveau d'un modèle

Le directeur est l'élément central qui gère les pas de simulation et contrôle l'avancement du temps. Dans le cas d'un modèle hiérarchique, c'est le directeur de plus haut niveau qui dicte les avancées. Les règles d'avancement varient selon le type de directeur. Tous les directeurs partagent néanmoins le même modèle de temps appelé *temps super dense*. Une date (encore appelée estampille temporelle, time stamp) est de la forme (t, n), où $t \in \mathbb{R}$ est le temps logique (ou temps du modèle ou temps de simulation) et $n \in \mathbb{N}$ est un micro-step. Le temps logique représente la date courante, tandis que le micro-step sert à ordonnancer des événements arrivant au même instant logique. La sémantique est complètement définie dans (Lee, 1999) et nous en donnons un aperçu ci-dessous.

Pour un directeur continu, à l'instant logique t_k, tous les acteurs sont appelés dans un ordre déterminé et une prochaine date t_{k+1} est calculée par le solveur. On recommence ainsi en faisant un nombre fini de pas pour atteindre une borne prédéfinie par l'utilisateur.

Pour un directeur discret, un acteur est réveillé à la date t s'il a un élément dans un de ses ports d'entrée ou s'il a demandé précédemment à être réveillé à cette date en utilisant la méthode `fireAt()`. Pour assurer le déterminisme, l'ordre d'appel des *fire* est crucial. Le directeur DE maintient une queue globale triée des événements : un événement est une paire (*valeur*, *time stamp*) où le time stamp est une date super-dense (t, n). Considérons deux événements $e_1 = (v_1, (t_1, n_1))$ et $e_2 = (v_2, (t_2, n_2))$. L'ordre dans la queue est réalisée en appliquant successivement un tri sur :

1. le time stamp : si $t_1 < t_2$, ou si $t_1 = t_2$ et $n_1 < n_2$, alors e_1 s'exécute avant e_2 ;

2. le rang : si $t_1 = t_2$ et $n_1 = n_2$, le directeur regarde le rang associé aux acteurs qui produisent e_1 et e_2. Le rang des acteurs est déterminé selon un tri topologique respectant l'ordre imposé par les précédence de données. L'utilisateur peut forcer les rangs en utilisant des priorités. Si le rang de l'acteur produisant e_1 est plus petit que celui produisant e_2 alors e_1 s'exécute avant e_2.

4. Coopération de simulation

L'objectif de ce travail est d'intégrer un modèle hiérarchique Ptolemy dans une fédération HLA. Dans la version actuelle de nos travaux, le modèle Ptolemy doit contenir comme directeur de haut niveau un DE. La hiérarchie peut ensuite contenir des acteurs composites de natures totalement différentes.

4.1. Gestion du temps

La coopération des deux outils passe par la mise en corrélation des estampilles temporelles des deux approches. Pour ce faire, un nouvel attribut Ptolemy, appelé *HlaManager*, et deux acteurs *HlaPublisher* et *HlaSubscriber* ont été développés. Ces trois composants permettent de réaliser l'interface entre les deux outils de simulation. Ces composants sont implémentés dans le langage de programmation Java et intégrés à l'outil Ptolemy dans le paquetage *ptolemy.apps.hlacerti*. La classe java implémentant l'attribut *HlaManager* est l'entité la plus complexe (env. 2300 lignes de code). Celle-ci s'appuie sur l'API fournie par *JCERTI* pour invoquer les services HLA, elle encapsule toute la logique de gestion d'avancée du temps ainsi que les échanges de données entre les deux outils. Les classes *HlaPublisher* (200 lignes) et *HlaSubscriber* (140 lignes) ont une logique moins complexe car elles sont uniquement en charge des opérations respectivement d'envoi d'événements (au *HlaManager*) et de délivrance (d'événements reçus du *HlaManager*) aux acteurs concernés du modèle Ptolemy.

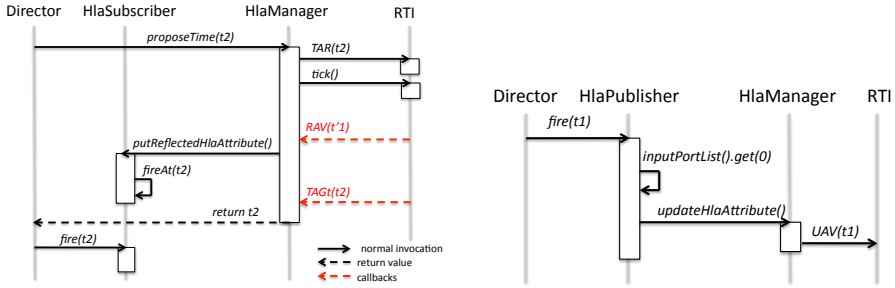


Figure 5. a. Avancement du temps *b. Exécution d'une publication*

La figure 5a montre une avancée dans le temps et reprend l'exemple 2a. Avant que le directeur DE n'avance dans le temps, il doit interagir avec le RTI au travers de l'at-

tribut *HlaManager*. Il envoie un signal particulier *proposeTime(t_2)* qui est transformé par *HlaManager* soit en *TAR(t_2)* ou *NER(t_2)*. Cela dépend de la configuration de l'attribut. Sur la figure il s'agit d'un *TAR* donc l'attribut ne redonnera la main au directeur qu'à réception du *TAG(t_2)*. A la réception de *RAV(t_1')*, *HlaManager* réveille l'acteur *HlaSubscriber* qui stocke la valeur reçue et rajoute un événement dans la file du directeur pour que ce dernier le réveille ultérieurement pour traiter les données reçues (*FireAt(t_2)*). A la réception de *TAG(t_2)*, *HlaManager* rend la main au directeur. Ce dernier réveille en premier *HlaSubscriber* afin que celui-ci mette les données à jour.

La figure 5b décrit la publication d'une donnée. Cela correspond au début de l'exemple 2a. Cette fois, l'acteur *HlaPublisher* entre en jeu et interagit avec l'attribut.

Les échanges entre le directeur DE et *HlaManager* se font au travers d'une interface, appelée *TimeRegulator*, développée conjointement avec UCB/EECS. Le directeur peut ainsi appeler la méthode *proposeTime*.

4.2. Participation à une fédération

Afin qu'un modèle Ptolemy devienne un fédéré, il faut gérer son intégration dans la fédération. Le tableau 1 résume la manière dont les services ont été intégrés dans *HlaManager*.

HlaManager	services	HLA
• pre-initialize()	• *instantiate rti'sproxy and federate's proxy*	
	• createFederationExecution • joinFederation	*Federation*
	• subscribeObjectClassAttributes • publishObjectClass	*Declaration*
	• registerObjectInstance	*Object*
• initialize()	• enableTimeRegulation • timeRegulationEnabled (callback) • enableTimeConstrained • timeConstraintedEnabled (callback)	*Time*
	• registerFederateSynchronizationPoint • Synch...PointReg...Succeeded (callback) • announceSynchronizationPoint (callback) • synchronizationPointAchieved • federationSynchronized (callback)	*Federation*
• wrapup()	• unpublishObjectClass • unSubscribeObjectClass	*Declaration*
	• resignFederationExecution • destroyFederationExecution	*Federation*

Figure 6. Intégration des services HLA dans l'attribut HlaManager

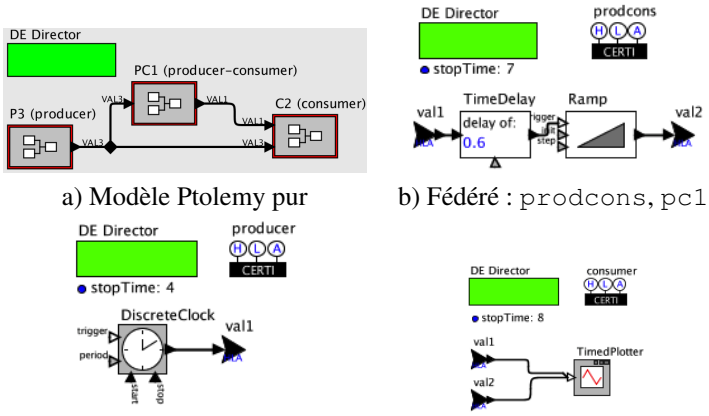
La méthode `pre-init()` est en charge de l'instantiation des objets. Il faut donc y mettre l'instantiation des objets requis par HLA/CERTI comme (1) le proxy du fédéré (implantant les callback HLA), (2) les services de gestion de la fédération (ex. création ou entrée dans une fédération) et (3) un certain nombre de services de déclaration relatifs à la publication et l'abonnement d'objets.

La méthode `init()` est en charge de l'initialisation de valeurs. On y ajoute donc la déclaration de la politique temporelle (régulateur, contraint) et la gestion des points de synchronisation. En effet, les points de synchronisation sont utilisés pour démarrer correctement la simulation dans une fédération et donc pour parfaitement synchroniser Ptolemy et le RTI.

Enfin, la méthode `wrapup()` est en charge de la terminaison de l'acteur. On y ajoute donc la sortie propre du fédéré de la fédération: arrêt de publication, fin d'abonnement, sortie (ou destruction, si l'acteur est le dernier fédéré) de la fédération.

4.3. Etude de cas

Illustrons au travers d'une étude de cas les possibilités induites par l'outil de coopération de simulation. La figure 7a montre un modèle complet Ptolemy composé de 3 acteurs composites : un producteur-consommateur `pc1`, un consommateur `c2` et producteur `p3`. Ce modèle est éclaté en 3 modèles distribués, figures 7b à 7d, qui deviendront des fédérés appartenant à une même fédération HLA.



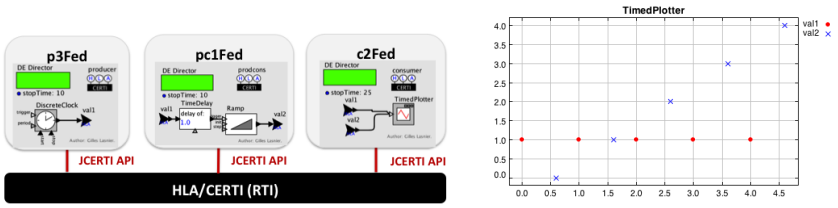
a) Modèle Ptolemy pur b) Fédéré : `prodcons`, `pc1`

c) Fédéré : `producer`, `p3` d) Fédéré : `consumer`, `c2`

Figure 7. Eclatement d'un modèle Ptolemy centralisé en 3 fédérés

Chaque acteur est encapsulé dans un modèle Ptolemy contenant un `HlaManager`. Ainsi dans la figure 7b, la déclaration de l'attribut est faite à droite du directeur et l'attribut à un nom *prodcons*, conforme au FOM. `HlaManager` est configuré par l'utilisateur et dans notre cas on ajoute le nom de la fédération `cspc` et le chemin vers le fichier .fed. L'acteur *HlaSubscriber* s'appelle *val 1* sur cette même figure et *HlaPublisher* s'appelle *val 2*.

La figure 8a montre l'architecture de simulation des 3 fédérés Ptolemy. La figure 8b affiche les valeurs de `val1` et `val2` reçues par le fédéré `c2`. `val1` est produite par *p3* qui est régulé par une horloge discrète périodique d'une unité de temps et qui s'arrête à la date 4. Sur l'affichage, `val1` est reçue à $\{0, 1, 2, 3, 4\}$ respectant ainsi la sémantique de Ptolemy. La valeur est constante et égale à 1. `val2` est produite par *pc1* à partir de la réception de `val1` : l'acteur *TimeDelay* applique un retard de 0.6 unité de temps et l'acteur *Ramp* calcule à chaque *fire()* l'entier $\sum_i 1$ en commençant à 0. Sur le diagramme, `val2` est reçue $\{0.6, 1.6, 2.6, 3.6, 4.6\}$ et les valeurs sont bien $\{0, 1, 2, 3, 4\}$.



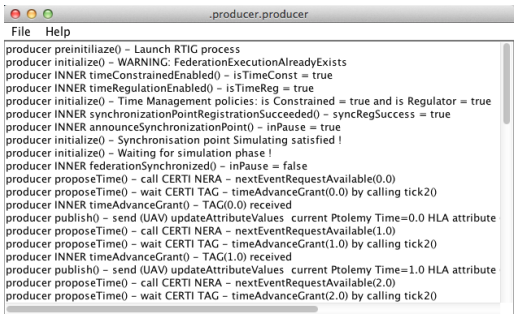
a) Co-simulation b) Résultats fonctionnels

Figure 8. Fédération composée des 3 fédérés Ptolemy

```
;; cspc
(Fed
  (Federation cspc)
  (FedVersion v1.3)
  (Federate "consumer" "Public")
  (Federate "producer" "Public")
  (Federate "prodcons" "Public")
  (Spaces
  )
  (Objects
    (Class ObjectRoot
      (Attribute privilegeToDelete reliable timestamp)
      (Class RTIprivate)
(Class myObjectClass
(Attribute val1 RELIABLE TIMESTAMP)
(Attribute val2 RELIABLE TIMESTAMP)
              (Attribute val3 RELIABLE TIMESTAMP) ) ) )

  (Interactions
    (Class InteractionRoot BEST_EFFORT RECEIVE
      (Class RTIprivate BEST_EFFORT RECEIVE) ) ) )
```

a) Fichier FED



b) Trace d'exécution du fédéré `p3`

Figure 9. Observation des échanges entre Ptolemy et HLA/CERTI

La figure 9a montre le fichier FED spécifiant le FOM de la fédération (ce fichier est fourni par l'utilisateur). On y trouve la spécification des attributs échangés entre fédérés, ainsi que leur interaction avec les mécanismes de gestion du temps. Dans cet exemple, la propriété "TIMESTAMP" attachée à un attribut renseigne que le RTIG doit délivrer les valeurs des mises à jour en suivant l'ordre de leur estampille temporelle. La figure 9b est la trace des interactions entre le fédéré Ptolemy *p3* et la fédération HLA. L'utilisateur peut ainsi visualiser les différentes étapes du cycle de vie du fédéré, notamment : création, intégration à la fédération, déclaration du comportement temporel du fédéré (*régulateur* et/ou *contraint*), synchronisation avec les autres fédérés, échange des valeurs d'attributs HLA (ex. *UAV*, *RAV*) et appels aux mécanismes d'avancée dans le temps (ex. *TAR*, *NER*). Les informations représentées sur la figure 9b sont consistantes avec le comportement attendu: le fédéré a rejoint la fédération (il ne l'a pas créée), il s'est déclaré comme *contraint et régulateur*, ce qui a été validé par le RTI, la synchronisation entre tous les fédérés a été réalisée, le fédéré a utilisé des NER. La première demande d'avance temporelle a été *NER(0)* qui a été acceptée par un *TAG(0)*, cela a été suivi par un *UAV(0)*. Ce schéma se répète ensuite pour $t = 1, 2, 3, 4$.

5. Conclusion et perspectives

Nous avons présenté un environnement de co-simulation permettant de simuler des systèmes CPS en combinant des modèles réalisés à plusieurs niveaux dans le cycle de conception. Notre approche repose sur l'utilisation des outils Ptolemy II et HLA/-CERTI : notre contribution a consisté à mélanger les deux approches et synchroniser les temps de simulation. La coopération a été illustrée sur une étude de cas simple pour des raisons de clarté et de manque de place. Une étude complète d'un système de commande de vol longitudinal a été réalisée dans (Lasnier *et al.*, 2013). A partir d'un modèle centralisé Matlab d'un avion F14, trois fédérés Ptolemy ont été implémentés: *Aircraft* (pour le modèle physique du F14), *AutoPilot* (pour le contrôleur) et *PilotStick* (pour représenter l'action du pilote sur le manche). Ces trois fédérés ont ensuite été interconnectés dans une simulation HLA: les commandes résultantes sont conformes au modèle centralisé. Plus récemment, les deux fédérés *Aircraft* et *AutoPilot* ont été connectés directement à la plateforme PRISE (Chaudron *et al.*, 2011) de façon à utiliser un vrai manche (commandé par une personne). Ainsi, selon l'action du *pilote*, le fédéré *AutoPilot* calculait la commande de la gouverne et le fédéré *Aircraft* réagissait en affichant le bon angle d'attaque.

La formalisation sémantique et la preuve de correction de l'environnement de co-simulation sont en cours d'écriture. Le formalisme exprimant la sémantique est le *tagged signal model*. Dans la suite, nous souhaitons étendre ces travaux pour prendre en compte des unités temporelles et des horloges physiques.

Les auteurs ont également développé des acteurs Ptolemy tels que des bus ou des commutateurs réseaux. Nous souhaitons donc connecter ces acteurs pour simuler des latences de bout en bout. A noter que ce mode de coopération n'existe pas dans la boîte à outil MATLAB/Simulink/HLA.

Remerciements

Ce travail est soutenu par le projet TORRENTS/TOAST du RTRA STAE. Les auteurs remercient E. Lee, P. Derler et C. Brooks de l'équipe CHESS de l'Université de Californie, Berkeley pour les discussions sur Ptolemy II. Les auteurs remercient également E. Noulard ONERA pour les discussions sur HLA/CERTI.

Bibliographie

Awais M. U., Palensky P., Elsheikh A., Widl E., Stifter M. (2013). The high level architecture rti as a master to the functional mock-up interface components. In *Proceeedings of international conference on computing, networking and communications (accepted).*

Chandy K. M., Misra J. (1979). Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Trans. Softw. Eng.*, vol. 5, p. 440–452. http://dx.doi.org/10.1109/TSE.1979.230182

Chaudron J.-B., Saussié D., Siron P., Adelantado M. (2011, juin). Real-time aircraft simulation using hla standard. *IEEE AESS Simulation in Aerospace*.

Department of Defense (DoD) Specifications. (1998a, Apr). *High Level Architecture Interface Specification, Version 1.3*. Rapport technique. DOD/DMSO HLA IF 1.3.

Department of Defense (DoD) Specifications. (1998b, Feb). *High Level Architecture Object Model Template, Version 1.3*. Rapport technique. DOD/DMSO OMT 1.3.

Department of Defense (DoD) Specifications. (1999, Mar). *High Level Architecture Run-Time Infrastructure Programmer's Guide*. Rapport technique. DOD.

Derler P., Lee E. A., Sangiovanni-Vincentelli A. (2012, Jan). Modeling cyber-physical systems. *Proceedings of the IEEE (special issue on CPS)*, vol. 100, p. 13 - 28. http://chess.eecs .berkeley.edu/pubs/843.html

FMI Development Group. (2012). *Functional mock-up interface (fmi).* https://www.fmi -standard.org/

ForwardSim Inc. Simulation and Technologies. (2013). *HLA Toolbox for MATLAB - HLA Blockset for Simulink.* http://www.forwardsim.com

Fujimoto R. M. (1996, Aug). *HLA time management: Design document*. Rapport technique. Georgia Tech College of Computing.

Hadj-Amor H., Soriano T. (2012). A contribution for virtual prototyping of mechatronic systems based on real-time distributed high level architecture. *Journal of computing and information science in engineering*, vol. 12, n° 1.

IEEE. (2010). IEEE standard for modeling and simulation High Level Architecture (HLA). *IEEE Std 1516-2010*, vol. 18, p. 1-38.

Lasnier G., Cardoso J., Siron P., Pagetti C., Derler P. (2013). Distributed simulation of heterogeneous and real-time systems. In *Distributed simulation and real time applications (ds-rt), 2013 ieee/acm 17th international symposium on.*

Lee E. A. (1999). Modeling concurrent real-time processes using discrete events. *Ann. Software Eng.*, vol. 7, p. 25-45.

MathWorks. (2013). *MATLAB/Simulink.* http://www.mathworks.com

Microsystems S. (1988, June). *RPC: Remote Procedure Call Protocol specification: Version 2* n° 1057. RFC 1057 (Informational). IETF. http://www.ietf.org/rfc/rfc1057.txt

Modelica Association. (2012). *Modelica: A unified object-oriented language for physical systems modeling, language specification version 3.3.* http://www.modelica.org/

Modelisar. (2012, Aug). *Functional mock-up interface for model exchange and co-simulation, Version 2.0 beta 4*. Rapport technique. Information Tech for European Advancement.

Noulard E., Rousselot J.-Y., Siron P. (2009, Mar). CERTI, an open source RTI, why and how ? *Spring Simulation Interoperability Workshop*, p. 23-27.

OMG. (2004, mar). *Common object request broker architecture : Core specification, version 3.0.3*. OMG Technical Document formal/04-03-12. Auteur. OMG.

ONERA. (2013). *The Open Source middleware CERTI.* http://www.onera.fr/dtim-en/hla -distributed-simulation/index.php

Ptolemaeus C. (Ed.). (2014). *System design, modeling, and simulation using ptolemy ii*. Ptolemy.org. http://ptolemy.org/books/Systems