Ingénierie Dirigée par les Modèles pour la Simulation, le cas de PRISE

Jérôme Hugues et Pierre Siron

Dans cet article, nous présentons une approche dirigée par les modèles permettant de fournir une vision haut niveau d'une simulation basée sur le standard HLA « High Level Architecture » sous forme d'un modèle AADL. Cette modélisation permet d'abstraire les paramètres clés d'une fédération HLA, de générer le code utile en lieu et place d'une écriture manuelle et de vérifier certains paramètres clés.

systèmes embarqués, simulation distribuée, modélisation, validation, contraintes temps réel, HLA, AADL

1. Introduction

La complexité des systèmes embarqués requiert la définition de stratégies visant à acquérir un niveau de qualité satisfaisant au plus tôt. Les étapes de tests d'intégration doivent être complétées par des étapes de vérification amont permettant de conforter les différentes étapes de conception et de réalisation. À ce titre, la simulation permet de réaliser une première étape d'intégration dans un environnement contrôlé.

Parmi les standards de simulation existants, le standard HLA [1] fournit une couverture complète des besoins en simulation. Pour autant, HLA repose sur un modèle objet implanté en C++ rendant complexe la mise en œuvre d'une simulation pour un système complexe. Le standard AADL fournit une représentation sous forme de modèle de l'architecture d'un système embarqué, en se focalisant sur les entités de premier ordre : processus, bus, tâches, etc. AADL est riche d'un écosystème couvrant aussi bien les étapes de modélisation, d'analyse (dimensionnement, performances, etc.) et de génération de code.

Dans cet article, nous présentons une évolution de la plateforme PRISE, Plate-forme pour la Recherche en Ingénierie des Systèmes Embarqués, développée à l›ISAE, l›Institut Supérieur de l›Aéronautique et de l›Espace. Cette évolution vise à remplacer la mise en œuvre manuelle de la simulation basée sur le standard AADL par une étape de génération de code depuis un modèle AADL représentant le système simulé. Cet apport de l'Ingénierie Dirigée par les Modèles permet en outre l'application de nombreuses analyses complémentaires visant à garantir l'exactitude de la plate-forme simulée.

2. Contexte et objectif

Cette partie 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.

2-1 Présentation du standard HLA

Le standard HLA (High-Level Architecture) 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, c.-à-d. 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 patron de modèle objet (basé sur le standard OMT 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.

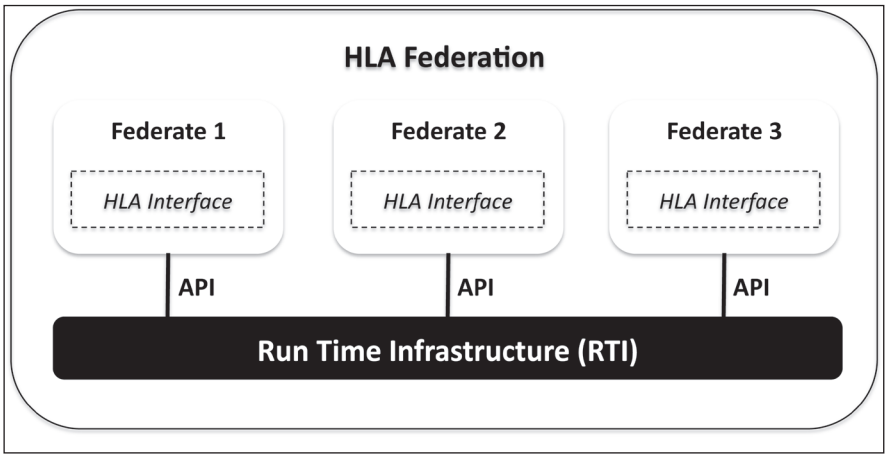


Figure 1 : Architecture de HLA

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 [2]. 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 [3]. Le temps logique est équivalent

Classes	Services	Acronyme	Description(informelle)	Invocation
Federation	createFederationExecution()		créationd' une fédération	normal
	joinFederationExecution()		rejoindreune fédération	normal
	resignFederationExecution()		quitterune fédération	normal
	destroyFederationExecution()		détruireune fédération	normal
	registerFed...Sync...Point()		définir un point de synchronisation	normal
	sync...PointReg...Succeeded()		acquittementdéfinitiondu point de synchronisation	callback
	announceSynchronizationPoint()		attendreun point de synchronisation	callback
	synchronizationPointAchieved()		fin du point de synchronisation	normal
	federationSynchronized()		fédérationsynchronisée	callback
	tick()	TICK	récupérationdes callbacksdepuis le RTI	normal
Declaration	publishObjectClass()		publicationd' un objet	normal
	subscribeObjectClassAttributes()		abonnementà un objet	normal
	unsubscribeObjectClass()		désabonnementà un objet	normal
	unpublishObjectClass()		arrêtde la publicationd' un objet	normal
Object	registerObjectInstance()		définir une instanced' objet	normal
	discoverObjectInstance()		découverted' instanced' objet	callback
	updateAttributeValues()	UAV	mise à jourd' une valeur à la fédération	normal
	reflectAttributeValues()	RAV	réceptiond' une mise à jour de valeur par la fédération	callback
Time	enableTimeRegulation()		déclarationd' un fédéréde typerégulateur	normal
	timeRegulationEnabled()		succès de la déclarationdu fédéréen tant que régulateur	callback
	enableTimeConstrained()		déclarationd' un fédéréde typecontraint	normal
	timeConstrainedEnabled()		succès de la déclarationdu fédéréen tant que contraint	callback
	timeAdvanceRequest()	TAR	demanded' avancée à une datedéterminée	normal
	nextEventRequest()	NER	demanded' avancée à la datedu prochainévénement	normal
	timeAdvanceGrant()	TAG	notificationque la demanded' avance dans le temps est accordée	callback

Figure 2 : Services HLA

Les services HLA sont organisés en six classes de gestion utilisées pendant la durée de vie d'un fédéré. La table 1 décrit le sous-ensemble de services HLA nécessaire pour notre approche. Pour chaque service, nous donnons une description informelle de son utilisation.

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.

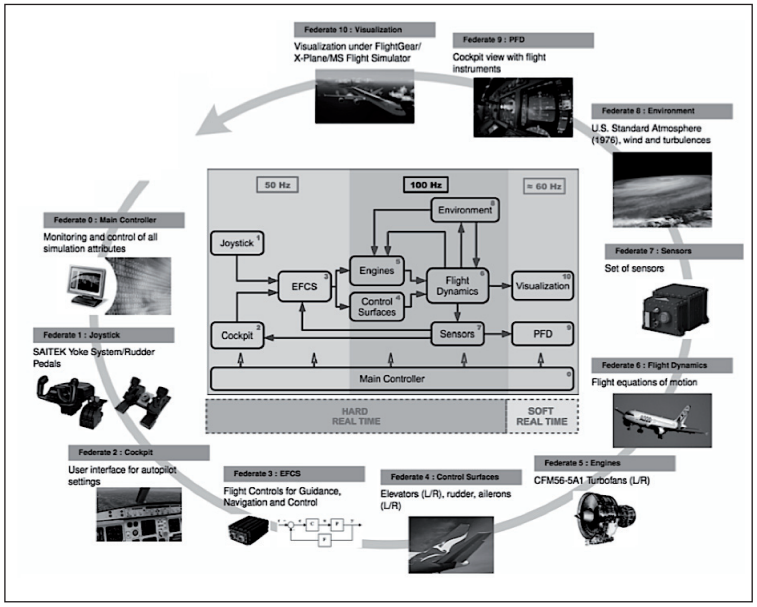
au temps de simulation dans la littérature classique sur la simulation à événement discret.

2-2 Plateforme PRISE/SDSE

La plateforme PRISE est un projet mis en œuvre à l'ISAE depuis 2007, destiné à servir de support aux bonnes pratiques d'ingénierie dans le domaine spatial et le domaine aéronautique. Le volet « Simulation Distribuée de Systèmes Embarqués » vise à fournir une plate-forme virtuelle représentative d'un système avion complet, bâti autour du standard HLA. SDSE est implanté au dessus de l'intergiciel CERTI [4], intergiciel multiplate-forme (Linux et Windows) implémentant le standard HLA, développé conjointement à l'ISAE et l'ONERA. Étant un logiciel libre, il permet de maîtriser l'implémentation du RTI et ainsi de faciliter l'intégration de modifications du code source. PRISE/SDSE fournit un ensemble de blocs logiciels et matériels

représentatif des différents blocs d'un système avion. Onze fédérés représentent chacun une partie de l'avion (avion naturel, moteur, gouvernes) et communiquent en utilisant CERTI pour assurer l'interopérabilité (figure 3) :

- Fédéré 0 : Surveille et contrôle tous les paramètres de la simulation
- Fédéré 1 ***Joystick*** : représente les différents organes de pilotage avec un ensemble volant/ manette de gaz/ palonnier
- Fédéré 2 ***Cockpit*** : interface utilisateur via un écran tactile
- Fédéré 3 EFCS (***Electronic Flight Control System***) : contrôle de l'aéronef et pilote automatique
- Fédéré 4 ***Control Surfaces :*** ailerons à gauche et à droite, des gouvernes de profondeur à gauche et à droite et une gouverne de direction.
- Fédéré 5 ***Engines*** : deux turboréacteurs (gauche et droite) CFM56-5A1
- Fédéré 6 ***Flight Dynamics :*** cœur du simulateur de vol (équations de mécanique de vol)
- Fédéré 7 ***Sensors*** : vingt capteurs tels que IRU, IMU, avec des phénomènes tels que retard put, bruit, quantification
- Fédéré 8 ***Environnement :*** modèle de l›atmosphère US (1976) avec différentes turbulences (Dryden, Von Karman, windshear)
- Fédéré 9 (***Primary Flight Display***) : cockpit avec les instruments de vol
- Fédéré 10 ***Visualization*** : environnement virtuel (FlightGear, Microsoft Flight Simulator et X-Plane)



▲ *Figure 3 : PRISE/SDSE*

2-3 Apport de l'IDM à PRISE/SDSE

Dans sa version actuelle, PRISE/SDSE repose intégralement sur une implantation manuelle basée sur le langage C++, l'ensemble des fédérés sont instanciés, configurés, déployés manuellement. Par ailleurs, les fédérés partagent une notion commune du temps, couvrant aussi bien le temps d'exécution des différents blocs que l'horloge globale servant à la synchronisation des échanges. Il est important de garantir que l'ensemble des échéances temporelles pourra être respectée. Pour autant, la mise en œuvre de HLA mélange code pour la simulation et opérations de synchronisation, rendant difficile toute analyse précise de la simulation sans une étape d'abstraction.

Afin de traiter ces problématiques, nous avons évalué l'apport du langage de description AADL [5] afin de 1) fournir une abstraction du système simulé, 2) l'analyser et 3) pouvoir générer à nouveau le code de synchronisation HLA utilisé lors des phases de simulation. Nous présentons cette étude dans la section suivante.

3. Modélisation de PRISE/SDSE

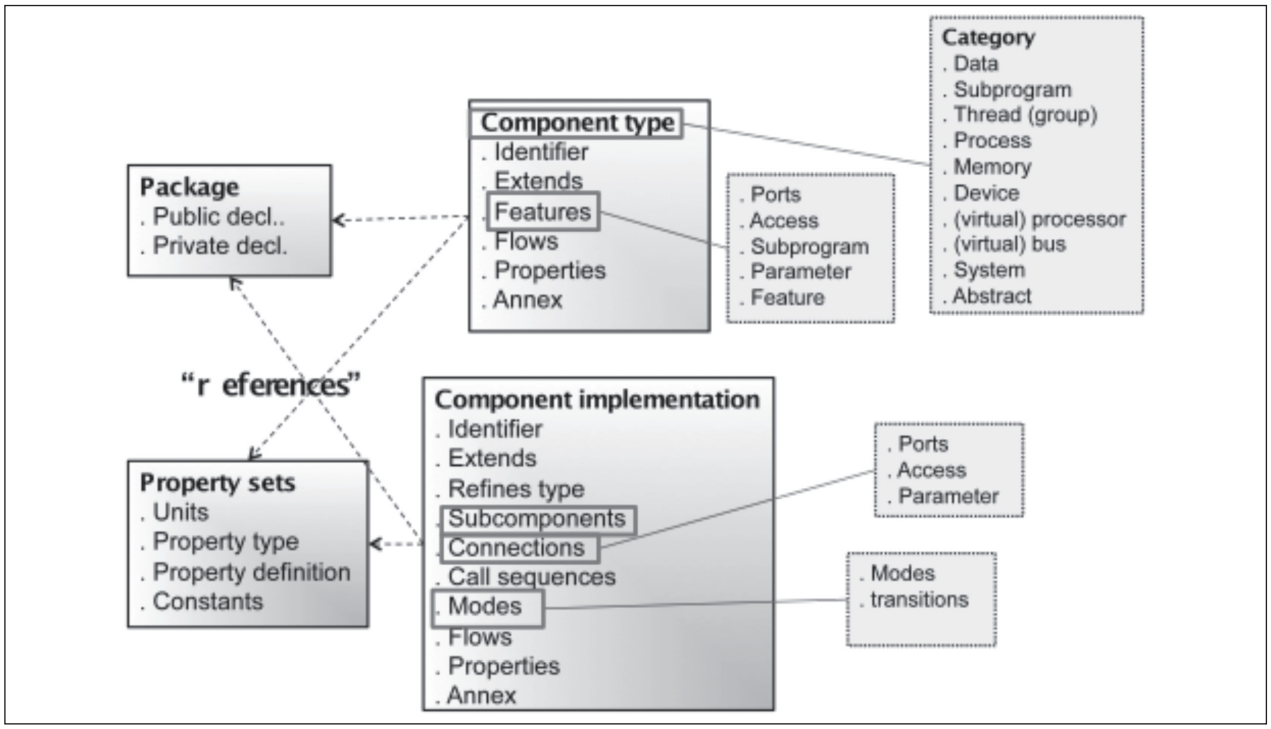
3-1 Présentation de AADL

Le langage de description d'architecture AADL, basé sur la description de composants formant l'architecture d'un système embarqué, est standardisé par SAE International [7]. AADL permet de modéliser à la fois les aspects logiciels et matériels d'un système temps réel en assistant une démarche de modélisation par raffinement (héritage simple), avec une sémantique clairement définie conforme aux systèmes avioniques et spatiaux, et un mécanisme avancé de gestion de propriétés non fonctionnelles.

La figure 4 fournit une vision haut niveau des différents blocs formant un modèle AADL. Ceux-ci sont formés d'ensemble de propriétés (***property sets***) et de paquetages. Les ensembles de propriétés définissent les attributs pouvant être attachés à des éléments du modèle : nom, type, unité. Les paquetages rassemblent les éléments du modèle : types des composants et implantations de composants.

- Les types de composants définissent les signatures des composants formant les briques de base de l'architecture : son identifiant, le composant qu'il étend par un mécanisme d'héritage simple, son interface et ses propriétés. Les types correspondent à une catégorie prédéfinie (tâches, processeurs, etc.) ; les interfaces à des mécanismes d'interaction connus : échange de messages, d'événements, de données, etc.
- Les implantations de composants définissent les réalisations concrètes des types ; ses sous-composants et les connexions entre ces derniers et l'interface du composant.

Au sein d'un composant, les modes permettent de modéliser les différentes configurations



▲ *Figure 4 : Vue simplifiée d›AADL*

d'un système, les décisions de reconfiguration en cas d'événement prévu (par exemple un cas d'erreur ou un ordre de démarrage). Chaque configuration peut spécifier des valeurs différentes pour les propriétés, mais aussi activer/désactiver certains composants. Ainsi, les modes définissent l'ensemble statique de configurations du système.

AADL définit un jeu de règles syntaxiques et sémantiques permettant de garantir la cohérence des modèles et leur bonne composition. Ces règles garantissent que le modèle est formé d'un assemblage valide de composants, mais ne libère pas des activités de v**érification et validation assis**tées par des outils ou une relecture du modèle.

3-2 Utilisation de AADL pour modéliser une simulation HLA

HLA et AADL reposent sur des entités différentes, mais la base sémantique est proche : il s'agit de faire dialoguer des composants définis par une interface, suivant une sémantique dirigée par le temps. Dès lors, il est possible de proposer une projection des concepts HLA vers ceux proposés par AADL de sorte qu'une fédération HLA puisse être exprimée à l'aide d'AADL.

- Un fédéré HLA représente une unité d'exécution, disposant de sa propre zone mémoire, on peut l'assimiler à un processus et un thread AADL ;
- Le code fonctionnel exécuté par une fédéré HLA correspond aux points d'entrées d'un thread AADL, suivant qu'il s'agisse du code exécuté lors de l'activation de la simulation, son initialisation ou son exécution. Nous montrons dans le figure 6 le lien entre points d'entrée d'un thread AADL et services HLA devant être exécutés à cette étape ;
- Les paramètres de configuration d'une simulation HLA peuvent être capturés par un jeu de propriétés dédié du langage AADL ;
- Le RTI HLA est équivalent à l'exécutif AADL, sous réserve que politiques d'activation des fédérés HLA et des threads AADL soient compatibles. Nous rappelons ces politiques dans la figure 5.

Dispatch_Protocol	Enabled	Wait_For_Dispatch Invariant
Periodic	$t = Period$	$t \leq Period \wedge \delta t = 1$
Aperiodic	$\exists p \in E : p \neq \emptyset$	$\forall p \in E : p = \emptyset$
Sporadic	$t \geq Period \wedge \exists p \in E : p \neq \emptyset$	$t < Period$ $\vee (t > Period \wedge \forall p \in E : p = \emptyset)$
Timed	$\exists p \in E : p \neq \emptyset \vee t = Period$	$\forall p \in E : p = \emptyset \wedge t < Period$
Hybrid	$t = Period \wedge \exists p \in E : p \neq \emptyset$	$\forall t \in E : p = \emptyset \wedge t \leq Period$
Background	*true*	$t = 0$

▲ *Figure 5 : Politiques d'activation des threads AADL*

3-3 Modélisation de SDSE à l'aide d' AADL et génération de code

Sur la base des définitions fournies précédemment, nous avons réalisé un premier prototype de SDSE en AADL. Ce modèle reproduit type de données échangées, fédérés et paramètres de configurations associés extraits à partir du code initial de SDSE et ses documents de conception.

Cette première étape nous permet de réaliser une analyse d'ordonnançabilité du système. SDSE étant un simulateur d'une plate-forme avion, l'ensemble de ses composants est implanté avec un souci de déterminisme. Par ailleurs, les différents composants ont été ramenés à des tâches périodiques implantant soit un solveur d'équation discret,

AADL	HLA Services
Activate_Entrypoint	Instantiate proxies createFederationExecution joinFederation subscribeObjectClassAttributes publishObjectClass registerObjectInstance
Initialize_Entrypoint	enableTimeRegulation timeRegulationEnabled registerFederateSynchronizationPoint announceSynchronizationPoint synchronizationPointAchieved
Compute_Entrypoint	updateAttributeValues reflectAttributeValues timeAdvanceRequest/nextEventRequest

▲ *Figure 6 : Correspondance HLA <-> AADL*

soit un système à événements discrets. Ces considérations permettent l'utilisation immédiate d'outils tels que Cheddar pour s'assurer que la simulation peut s'exécuter suivant les contraintes énoncées dans le modèle.

La seconde étape concerne la phase de génération de code. Elle nécessite d'une part de revisiter le code des différents fédérés afin de suivre le patron proposé figure 6, d'autre part d'adapter le générateur de code Ocarina pour utiliser l'API HLA en lieu et place des API POSIX et Socket utilisées sur cible native. Ces étapes sont en cours de réalisation, elles ont pour le moment été validées sur des simulations simples.

4. Conclusion

Dans cet article, nous avons présenté le cadre général pour utiliser AADL comme langage décrivant une simulation HLA. Ce faisant, nous facilitons un certain nombre d'analyses sur le système simulé, et permettons de regénérer le code de simulation.

Nous envisageons deux extensions à ces travaux 1) extension à des simulations utilisant des interactions plus complexes, notamment sans attente, 2) utilisation du standard FMI comme moyen d'implanter un fédéré.

5. Bibliographie

[1] The Institute of Electrical and Electronics Engineers (IEEE) Computer Society **: *IEEE Standard for Modeling and Simulation High Level Architecture (HLA) - Federate Interface Specification*** ; septembre 2000.

[2] K. M. Chandy et J. Misra : ***Distributed simulation: A case study in design and verification of distributed programs*** **;** IEEE Trans. Softw. Eng., vol. 5, p. 440–452, 1979. Consulté sur http://dx.doi.org/10.1109/TSE.1979.230182.

[3] R. M. Fujimoto : ***HLA time management: Design document*** ; Rapport technique, Georgia Tech College of Computing, août 1996.

[4] J.-B. Chaudron, D. Saussié, P. Siron et M. Adelantado : ***Real-time aircraft simulation using HLA standard*** ; IEEE AESS Simulation in Aerospace 2011, Toulouse, 8 juin 2011.

[5] P. H. Feiler et D. P. Gluch : ***Model-based engineering with AADL - An introduction to the SAE architecture analysis and design language*** : SEI series in Software Engineering, Addison-Wesley 2012, ISBN 978-0-321-88894-5, pp. I-XX, 1-468.

[6] J. Hugues, B. Zalila, L. Pautet et F. Kordon : ***From the prototype to the final embedded system using the Ocarina AADL tool suite*** ; ACM Trans. Embedded Comput. Syst. 7(4), 2008.

[7] SAE. ***Architecture Analysis and Design Language (AADL) AS-5506A***. Technical report, The Engineering Society For Advancing Mobility Land Sea Air and Space, Aerospace Information Report, Version 2.0, January 2009.