Environnement de simulation GDEVS compatible HLA

Gregory Zacharewicz

Environnement de simulation GDEVS compatible HLA

Grégory Zacharewicz
LSIS
Université d'Aix-Marseille III
Avenue Escadrille Normandie Niemen
13397 - Marseille cedex 20
gregory.zacharewicz@lsis.org

Résumé : Deux des enjeux actuels de la simulation sont la réutilisation de modèles et la composition de simulations globales distribuées à partir de simulateurs répartis. Cette publication présente un environnement contribuant à modéliser et simuler de façon distribuée un système. Cet environnement utilise les concepts de spécification DEVS développés par B.P. Zeigler et la généralisation GDEVS définie par N. Giambiasi permettant ainsi d'utiliser les performances de la simulation à événements discrets. Il permet également la composition de modèles à partir d'éléments stockés en bibliothèques, évitant le redéveloppement de simulations existantes. De plus, la structure de simulation choisie est « mise à plat », réduisant ainsi l'échange de messages entre les composants par rapport à la structure des simulateurs DEVS existants. Enfin, sa compatibilité avec la norme HLA, grâce à l'utilisation de mécanismes de synchronisation d'échange des messages, lui permet de s'intégrer dans des simulations globales hétérogènes compatibles HLA.
Mots Clés : Modélisation et simulation à événements discrets, DEVS, GDEVS, Simulation distribuée, Synchronisation, HLA

1 INTRODUCTION

Les problèmes de performance et de répartition des moyens informatiques nécessitent le développement de protocoles de simulation distribuée et de structures hiérarchiques de modélisation et simulations efficientes.
La modélisation et la simulation à événements discrets ont pour atout une rapidité d'exécution due à l'évolution dirigée par les événements, évitant les traitements par pas temporel. D'autre part, la notion de couplage de modèles permet la réutilisation de modèles dans un nouveau modèle composite sans recoder. En revanche, les différents composants pouvant se situer sur divers ordinateurs avec des systèmes d'exploitation hétérogènes, il est nécessaire de définir un protocole de communication permettant d'intégrer les différents éléments dans une simulation globale distribuée. HLA répond à cette question par l'échange de messages entre les simulations qui permettent de diffuser des informations et de synchroniser les différentes entités.
Nous proposons une réponse efficace à ce double enjeu qui consiste à créer des simulations distribuées à événements discrets communiquant au travers de l'interface spécifiée par HLA. Cette solution permet ainsi l'exécution d'une simulation de façon repartie sur plusieurs ordinateurs. Il est à noter que dans cet environnement, les respects du déterminisme et de la causalité sont assurés par les algorithmes de synchronisation mis en œuvre par l'implémentation d'HLA.
Nous présentons dans une première section un rappel de la spécification de systèmes à événements discrets DEVS et GDEVS, puis un rappel des concepts généraux de simulation distribuée et de la norme HLA, enfin nous exposons le fonctionnement des composants de notre environnement GDEVS compatible HLA.

2 MODELISATION ET SIMULATION A EVENEMENTS DISCRETS

Nous proposons ici un bref rappel du formalisme DEVS et GDEVS.

2.1 Discrete Event System Specification (DEVS) [Zeigler 76]

B.P. Ziegler définit à partir de 1976 une spécification formelle de modèles à événements discrets DEVS (Discrete Event System Specification). Il introduit notamment la possibilité d'évolution autonome du modèle grâce à la durée de vie des états, et à la fonction de transition interne.
Ce formalisme a été introduit comme un formalisme abstrait universel indépendant de l'implémentation. Le concept de modèles basiques-couplés, introduit par [Zeigler 90], fournit un moyen de construire des modèles composés, en réutilisant des descriptions stockées en librairie.

2.2 Modèle atomique

Le modèle atomique est représenté par une « boite » comportant des entrées, des sorties continues par morceaux. Cette « boite » contient un modèle à événements discrets de type DEVS définissant le comportement du modèle. Les signaux d'entrée sont

abstraits par une forme constante par morceau où les seuils sont considérés comme des événements discrets. Les signaux de sortie sont déterminés par les événements de sortie considérés à leur tour comme seuil d'une plage constante.
Formellement, un modèle atomique M est spécifié par un septuplet :

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

X : ensemble des types des événements externes,
S : ensemble des états séquentiels,
Y : ensemble des types d'événements de sortie où **e** représente le temps écoulé dans l'état s,
δ_{int} : **S** $\rightarrow$ **S** fonction de transition interne définissant les changements d'état dus à des événements internes,
δ_{ext} : **ST** $\times$ **X** $\rightarrow$ **S** fonction de transition externe définissant les changements d'état dus à des événements externes,
L'ensemble **ST** des états totaux du système est :
ST = {(s, e) s $\in$ S, o $\leq$ e $\leq$ D (s)}
λ : **S** $\rightarrow$ **Y** fonction de sortie,
D : **S** $\rightarrow$ **R+** $\cup \infty$ fonction durée de vie des états.
Les quatre éléments dans le septuplet nommés δ_{int}, δ_{ext}, λ, D sont les fonctions caractéristiques, S est l'ensemble des variables d'état et X,Y sont respectivement les ensembles d'événements d'entrée et de sortie.

2.3 Modèle couplé

Un modèle couplé est un modèle structurel. Il décrit une structure par interconnexion de modèles de base. Chaque modèle de base du modèle couplé interagit avec d'autres modèles pour produire le comportement global. Les modèles de base sont soit des modèles atomiques soit d'autres modèles couplés figurant dans une bibliothèque, le couplage de ces modèle se réalise de façon hiérarchique comme indiqué Figure 1 a).
Un modèle couplé à événements discrets est défini par la structure suivante :

$$MC = < X, Y, D, \{M_d / d \in D\}, EIC, EOC, IC, Select >$$

X : ensemble des événements externes.
Y : ensemble des événements de sortie.
D : ensemble des noms de composants.
M_d : modèle DEVS.
EIC : couplages d'entrées externes.
EOC : couplages de sorties externes.
IC : couplages internes.
Select : définit une priorité entre événements simultanés destinés à des composants différents.

2.4 Generalised Discrete Event System Specification (GDEVS) [Giambiasi 00]

Norbert Giambiasi en 2000 généralise la notion de modélisation à événements discrets. L'abstraction traditionnelle à événements discrets approxime les signaux d'entrée, de sortie et d'état par des segments constants par morceau. GDEVS définit les abstractions par des trajectoires polynomiales par morceau. DEVS est donc un cas particulier de GDEVS, c'est-à-dire un GDEVS d'ordre 0.
En ce sens, le modèle GDEVS accepte des événements d'entrée sous la forme de vecteurs contenant les coefficients d'un polynôme de degré déterminé approchant le signal analogique observé.
La représentation d'un signal original est donc plus précise pour la modélisation de processus continus par abstraction à événements discrets.
Formellement un modèle à événements discrets d'ordre n [Giambiasi 00] :

$$SEDN = <XM, YM, S, \delta_{int}, \delta_{ext}, \lambda, D>$$

représente un système dynamique :

$$SD = <T, X, Y, \Omega, Q, \Delta, \Lambda>,$$

de la manière suivante :
XM = A^{n+1} (avec A : sous ensemble des réels)
YM = A^{n+1}
S = Q x (A^{n+1})
Pour tout état total (q, (a_n, a_{n-1},....., a_0), 0) et un segment d'entrée polynomial continu w : $<t_1, t_2> \rightarrow X$,
sont définis :
la fonction de transition interne :
δ_{int} (q, (a_n, a_{n-1},....., a_0)) =
Straj q, x ((t1+D((q, (a_n, a_{n-1},....., a_0)), x)
avec x = $a_n t^n + a_{n-1} t^{n-1} + ... + a_1 t + a_0$
et Straj trajectoire d'état du modèle
$\forall$ q de Q et $\forall$ w :<t1, t2> $\rightarrow$ X,
Straj q,w:<t1, t2> $\rightarrow$ Q
la fonction de transition externe:
δ_{ext} (q, (a_n, a_{n-1},....., a_0), e, (a_n', a_{n-1}',....., a'_0)) =
(Straj q, x(t_1+e), x')
avec : Coef (x) = (a_n, a_{n-1},....., a_0)
et Coef (x') = (a_n', a_{n-1}',....., a'_0)
Coef : fonction qui associe à tout polynôme continu sur un intervalle <ti, tj>, la valeur du (n+1)-Uplet de constante (a_n, a_{n-1},....., a_0) tel que :
w(t) = $a_n t^n + a_{n-1} t^{n-1} + ... + a_1 t + a_0$
InCoef : **λ** (q, (a_n, a_{n-1},....., a_0)) = Λ(q, x)
la fonction de sortie :
λ : S $\rightarrow$ A^{n+1}
la fonction définissant la durée de vie des états
D(q, (a_n, a_{n-1},....., a_0)) =
MIN(e/Coef (Otraj q, x(t_1))
$\neq$ Coef (Otraj q, x(t_1 + e))
avec Otraj trajectoire de sortie du modèle
Otraj q,w:<t1, t2> $\rightarrow$ Y

2.5 Simulateur DEVS [Zeigler 00]

2.5.1 Simulation de Modèles couplés

Parallèlement à l'élaboration des différents modèles DEVS présentés précédemment, B.P. Zeigler a développé le concept de simulateur abstrait [Zeigler 00]. Cette architecture est dérivée du formalisme DEVS. Un simulateur abstrait représente une description algorithmique permettant de mettre en oeuvre les instructions implicites des modèles issus du formalisme DEVS, afin de générer leur comportement. Un tel simulateur est obtenu en faisant correspondre à chaque élément du modèle un composant du simulateur. L'originalité de cette démarche réside dans la construction d'un simulateur indépendant du modèle. Ceci se traduit par une séparation, au niveau de la

réalisation, de la partie modélisation et de la partie simulation.
Pour effectuer une simulation, une hiérarchie de processeurs, équivalente à la hiérarchie de modèles, est construite (Figure 1 b). A chaque modèle est associé un « simulateur ». Chaque processeur effectue la simulation en exécutant les fonctions qui expriment la dynamique du modèle. Les composants sont :
le Simulateur qui assure la simulation des modèles atomiques en utilisant les fonctions définies par DEVS,
le Coordinateur qui assure le routage des messages entre les modèles couplés en fonction des définitions de couplage,
le Coordinateur Racine qui assure la gestion globale de la simulation. Il ordonne le début et la fin de la simulation et gère l'horloge globale.

2.5.2 Types de messages échangés
La simulation s'effectue grâce à l'envoi de messages spécifiques entre les différents processeurs comme décrit dans la figure 1. Les trois premiers types de messages ci-dessous représentent les différents événements définis dans DEVS.
Xmessage : utilisé lorsqu'un événement externe arrive sur un modèle,
***message** : utilisé pour signaler un changement d'état dû à un événement interne,
Ymessage : utilisé pour signaler l'émission d'un événement de sortie sur le modèle,
Imessage : utilisé pour initialiser le modèle avec l'ensemble des valeurs par défaut choisies par l'utilisateur. (Ce message n'a qu'un intérêt informatique)

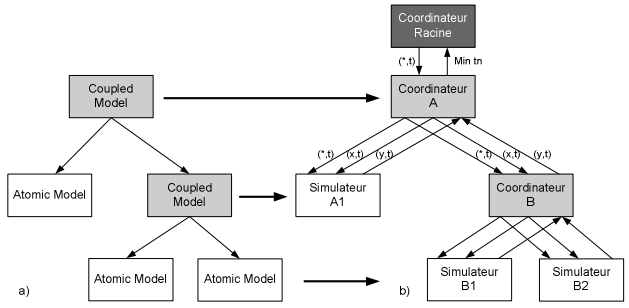

Figure 1 : Correspondance entre modèle et simulateur

3 SIMULATION DISTRIBUEE

L'objectif est d'utiliser au maximum la puissance des ordinateurs, de travailler sur des ordinateurs distants et de réutiliser des simulations existantes en les interconnectant.

3.1 Solutions architecturales disponibles
Plusieurs solutions architecturales peuvent être étudiées pour effectuer une simulation distribuée.
Parmi celles-ci, nous pouvons retenir l'utilisation d'un ordinateur multiprocesseur qui partage sa mémoire. Il est également possible d'interconnecter plusieurs ordinateurs éventuellement distants (clustering). Cette possibilité offre le choix d'utiliser ou non une mémoire partagée, et fonctionne principalement par envoi de messages permettant aux différentes simulations d'échanger des données et de se synchroniser.

3.2 Méthodes de distribution possibles
Une simulation peut distribuer différents éléments. La première solution est la distribution des fonctions du simulateur. Une autre solution consiste à distribuer les événements. Enfin, la dernière solution consiste à distribuer les éléments du modèle ; on utilise ainsi au mieux le parallélisme du modèle. Dans ce cas, on procèdera par échange de messages datés entre les processus. Il sera également obligatoire d'utiliser une synchronisation des processus.

3.3 Evolutions
3.3.1 Evolution Synchrone
Le temps simulé correspond à une horloge globale. Toutes les horloges logiques locales (horloges de chaque processeur) sont donc dirigées par cette horloge globale. A chaque pas temporel, les différents processeurs exécutent des actions.

3.3.2 Evolution Asynchrone
Pour ce type d'évolution, un processeur n'a pas de vision globale (dans le cas classique il n'a pas de mémoire partagée), les prises de décision sont donc faites en fonction des informations locales.
Dans ce cas, le temps logique de chaque processeur évolue d'événement en événement. Mais les processeurs ayant entre eux une relation « d'influencés-influenceurs » doivent, avant toute exécution d'un événement daté au temps T, être sûrs de ne pas recevoir dans le futur un message avec une date T'<T. Le problème de cette évolution est le respect de la causalité.

3.4 Exécution distribuée : respect de la causalité
Un traitement distribué doit s'assurer, en simulation distribuée, de reproduire les relations de causalité existantes dans l'exécution séquentielle équivalente.
La causalité impose donc un ordre partiel entre les événements. [Lamport 78] définit une méthode basée sur les horloges logiques locales de telle façon que, pour tout événement a et b, si a est arrivé avant b (a→ b), implique que le temps logique local C(a) doit être strictement inférieur à C(b).

3.5 Types de synchronisation :
Pour respecter la causalité, deux types de synchronisations entre les processeurs sont possibles :

3.5.1 Approche pessimiste (ou conservative)
Un processeur traite un événement local ou reçu d'un influenceur de date T (et incrémente sa date actuelle) lorsqu'il est sûr de ne pas recevoir d'autres événements de date T'<T, et donc respecter le principe de causalité. Ces premiers algorithmes ont été proposés par [Bryant 77] et [Chandy 79].

Le problème de la synchronisation conservative est la situation d'interblocage (DeadLock). Pour supprimer ces situations d'interblocage, [Chandy 79, 81] et parallèlement [Bryant 77] ont introduit la notion de **Null message**. Un Null message n'a pas de contenu autre qu'une date. Lorsqu'un processeur transmet un message daté sur une de ses sorties, il transmet également un Null message de même date sur ses autres sorties. Dans le cas de composants s'influençant en boucle, l'un d'eux devra également avoir un temps de traitement non nul.

3.5.2 Approche optimiste

La simulation traite les événements à sa connaissance (pas de vision globale de la simulation), cette connaissance partielle des événements à traiter peut induire l'omission de certains événements externes et le non respect de la causalité [Samadi 85]. Si une contrainte de causalité est enfreinte (réception d'un événement dont la date est antérieure à la date actuelle locale du processeur), la simulation « revient alors dans son passé ». Le mécanisme de « Rollback » est alors déclenché [Jefferson 85]. Une lacune de l'approche optimiste est le temps d'exécution passé à faire des Rollbacks souvent important.

4 SYSTEMES DE SIMULATION DISTRIBUEE

4.1 HLA (High Level Architecture)

4.1.1 Définition et but de HLA [DMSO 98]

HLA a été développé en 1995 par le Bureau de Modélisation et de Simulation de Défense (DMSO) du Ministère de la Défense nationale américain (DoD) pour satisfaire les besoins de projets concernant la défense.
L'Architecture de Haut Niveau (High Level Architecture HLA) est une spécification d'architecture logicielle permettant de créer des simulations comprenant différents composants de simulation. Ces composants doivent pouvoir être réutilisables et inter-fonctionner sans recodage.
Dans HLA, chaque simulation participante est appelée **fédéré** ; elle interagit avec d'autres simulations fédérées au sein de ce qui est nommé dans HLA une **fédération**, qui est en fait un groupe de fédérés.

HLA peut être défini selon trois composants [DMSO 98] :
Les **Règles HLA** assurent l'interaction appropriée des simulations dans une fédération. Elles décrivent également les responsabilités des fédérations et des fédérés.
Le **« Template » de Modèle d'Objet** (OMT) fournit une structure commune pour la documentation de modèle d'objet HLA et favorise l'inter-fonctionnement et la réutilisation de simulations et de leurs composants. Il contient le SOM (Simulation Object Model) qui définit les objets et les interactions qui peuvent être utilisés par une simulation lorsqu'elle participe à une fédération HLA. Il contient aussi le FOM (Federation Object Model) qui définit les éléments effectivement utilisés entre les simulations dans cette fédération.
La **Spécification d'interface** définit les interfaces fonctionnelles entre les fédérés et l'infrastructure de temps d'exécution (RTI) qui devront être respectées pendant l'exécution pour obtenir une simulation compatible HLA. La norme IEEE 1516 définie cela dans [DMSO 1998]. Le RTI est l'implémentation de cette spécification. Il fournit les services logiciels nécessaires pour une simulation « HLA-compliant ».

4.1.2 Eléments de l'implémentation

Un fédéré est un composant compatible HLA, pour cela le code du fédéré conserve ses fonctionnalités originales mais il doit être complété par un certain nombre de fonctions pour permettre la communication avec les autres membres de la fédération. Ces fonctions sont contenues dans le code de la classe « Federate Ambassador » qui permet de rendre interprétables par un processus local les informations reçues provenant de la fédération.
Le « Local RTI Components code » (LRC) fournit des fonctionnalités externes au fédéré pour l'enregistrement et l'utilisation des services du RTI tels que l'enregistrement d'objets et la gestion du temps.
Enfin, le composant central du RTI gère la fédération en s'appuyant notamment sur les informations fournies par le FOM pour définir les objets et les interactions participant à la fédération. Cette configuration est illustrée par la Figure 2.
Un fédéré peut, au travers des services proposés par le RTI, publier « Publishing » et souscrire « Subscribing » à une classe. "Publish" signifie que le fédéré a l'intention de diffuser la création d'instances de classe et la mise à jour des attributs de ces instances. "Subscribe" signifie l'intention des fédérés de refléter certains attributs de certaines classes à d'autre fédérés.

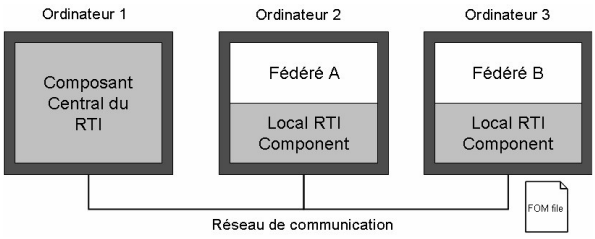

Figure 2 : Composants du RTI

4.1.3 Management du temps dans la spécification HLA [R.M. Fujimoto 98]

HLA utilise les algorithmes de synchronisation pessimiste et/ou optimiste vus précédemment, le RTI implémente donc les notions suivantes :
Lookahead : date donnée par un processeur influenceur attestant qu'il n'émettra pas de message avant celle-ci.
LBTS (Lower Bound on Time Stamp) : date jusqu'à laquelle un processeur ne sera influencé par des messages provenant de processeurs influenceurs.
NextEvent Request () : interrogation par un processeur du RTI pour obtenir l'autorisation de sélectionner un événement dans son échéancier.
Il est important de retenir que la norme HLA n'est pas une implémentation, ce n'est qu'une spécification.

5 ENVIRONNEMENT DE SIMULATION GDEVS COMPATIBLE HLA

5.1 Mise à plat du simulateur de modèles couplés GDEVS local

A partir de la structure hiérarchique de simulation abstraite définie par [Zeigler 00], nous proposons une structure hiérarchique « compacte ».
Pour cela nous définissons tout d'abord la structure des messages par un vecteur de 6 valeurs : (type de message, émetteur ou récepteur, date événement, port concerné, valeur de l'événement).
Ensuite, nous ajoutons aux types de messages définis par B.P. Zeigler : le **Dmessage,** utilisé pour indiquer que le modèle simulé a terminé sa tâche. Ce message d'acquittement, émis par les simulateurs atomiques en retour d'un message reçu d'un coordinateur, véhicule l'information de durée de vie de l'état actuel du modèle.
Enfin, pour représenter les trajectoires polynomiales des modèles GDEVS, les états du modèle et la valeur des Xmessages seront définis par un vecteur de valeurs.
Outre ces modifications d'autres points diffèrent de la structure initiale de B.P. Zeigler :
A partir des travaux de [Kim 00] et dans le but de diminuer l'échange local de messages entre les coordinateurs et les simulateurs, nous avons réduit la structure arborescente de coordinateurs intermédiaires entre le coordinateur racine et les simulateurs, à deux niveaux hiérarchiques. Les éléments subsistant localement sont donc un coordinateur local et un ensemble de simulateurs atomiques connectés en successeurs directs. Cette nouvelle configuration est représentée Figure 3 par les groupes Ordinateur 2 et 3.
Les différents modèles pouvant être exécutés sur différents ordinateurs, nous proposons une méthode permettant d'interconnecter des modèles repartis avec l'ajout d'un composant racine décrit par la suite.

5.2 Composants de simulation distribuée

5.2.1 Coordinateur local - simulateurs

Le coordinateur local manipule un échéancier (Eq) contenant les Xmessages insérés à son niveau et les *messages déduits des durées de vie de ses successeurs et gère une horloge locale (LocalTime). Il dirige donc la simulation locale en sélectionnant le prochain message chronologique de son échéancier par rapport à sa date actuelle et en le transmettant au simulateur successeur concerné. Il comporte également une liste d'attente (WaitList) et une liste d'acquittement (StopList) permettant de stocker les messages transférés et vérifier leur traitement par les successeurs. Enfin, il conserve les relations de couplage avec ses successeurs dans les listes (EOCList, ICList, EICList). Les simulateurs conservent la structure décrite par [Zeigler 00].
Ce groupe (Coordinateur Local - Simulateur(s)) peut donc simuler localement un modèle couplé de façon autonome. Il est à noter que, dans le cas d'une intégration de ce coordinateur local à une simulation distribuée, ce composant devra s'assurer de la gestion des messages provenant d'autres coordinateurs.
Notre objectif est d'obtenir un environnement de simulation distribuée. Un certain nombre de structures locales définies précédemment doivent donc communiquer entre elles (Groupes Ordinateur 2 & 3 figure 3) afin d'obtenir une simulation globale repartie sans les recoder. Pour cela, il est nécessaire de gérer les messages échangés entre les composants distribués.

5.2.2 Coordinateur distribué

Pour assurer la synchronisation globale de la simulation, nous avons ajouté un composant coordinateur racine distribué correspondant au groupe Ordinateur 1 figure 3.
Ce coordinateur distribué a pour fonction l'échange de messages entre les composants locaux ; il doit donc synchroniser les envois-réceptions de messages en respectant la causalité des événements. Il utilise un échéancier contenant les messages échangés dans la simulation globale (Eq), une liste d'attente (WaitList) et une liste d'acquittement (StopList) fonctionnant sur le même principe que le coordinateur local. Il comporte également une liste contenant les valeurs de lookahead (Lq) des différents successeurs, une liste des coordinateurs en attente (Wq) et un ensemble de tables contenant les relations de couplage entre les modèles couplés distants (EICList, EOCList, ICList).

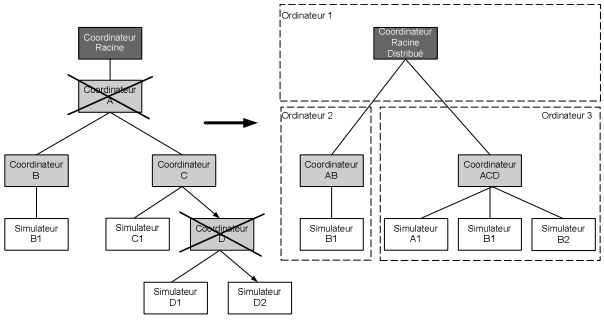

Figure 3 : Mise à plat et distribution de la structure de simulation hiérarchique

Pour assurer la synchronisation globale de nos simulateurs, nous avons développé une méthode de synchronisation conservative basée sur les travaux de [Bryant 77] et [Chandy 81]. A ce titre, nous introduisons dans notre environnent les notions de **Null message** et de **lookahead Message** associées à la synchronisation conservative.
Cette méthode présente l'avantage d'éviter un échange trop important de messages contrairement aux situations de Rollback de l'approche optimiste.

5.3 Algorithme de traitement des événements des composants

Chaque composant comporte un certain nombre de fonctions pour traiter localement les messages et interagir avec son entourage. Nous présentons ici sous forme de pseudo code les principales méthodes des trois types de composants.

5.3.1 Coordinateur Local

Le coordinateur local doit, dans un premier temps, s'enregistrer au niveau du coordinateur racine en lui précisant ses relations de couplage. Il lui envoie donc un message contenant son nom et ses EOC, EIC et IC.
Ensuite, le coordinateur local utilise une boucle de traitement des événements de son échéancier dans laquelle il sélectionne le (ou les) prochain(s) événement(s) de son échéancier.

```
Event-Loop:
Ask for Join-Federation-Request (Coordinator, EIC,
EOC, IC)
Do      Select first in Eq ('type', Simulator, t,-,-)
        // t = infinite if Eq is empty
        Next-Event-Request (Coordinator,
        InfluencedPort, LocalTime, t)
While true
Endwhile
End Event-Loop
```

Figure 4 : Algorithme boucle de traitement du coordinateur local

Ce coordinateur faisant partie d'une simulation distribuée doit s'assurer du respect du principe de causalité, en d'autres termes, il doit être certain de ne pas recevoir par la suite un événement ayant une date inférieure à la date de l'événement qu'il a sélectionné. L'algorithme du coordinateur local possède donc un appel au coordinateur racine distribué (NextEventRequest) permettant de s'assurer de la possibilité de traiter des événements locaux sans omettre de prendre en compte les influences d'autres simulateurs répartis.
La réponse de l'appel au coordinateur racine se présente sous la forme d'une autorisation de traitement, d'un Xmessage ou d'un Null message de date inférieure ou égale à la date de l'événement local sélectionné. Nous illustrons ci-dessous le traitement d'un Xmessage.

```
When receive Xmessage ('x', Coordinator, t,
InfluencedPort, Value){
if (tl<= t <= tn)
then    Get-External-Influenced-By (Coordinator,
        nfluencedPort)
        // consult external input coupling to get
           children influenced by the input
        for each Simulator influenced by input do
                add Xmessage ('x', Simulator, t,
                InfluencedPort, Value) to WaitList
                // according to priority definition
        Send first message(s) (addressed to same
         Simulator & with timestamp == t) of WaitList
        & mark it(them) "send"
        tl = t
        tn = t first Event of the EventList
else error : bad Synchronization}
```

Figure 5 : Algorithme réception Xmessage du coordinateur local

En fonction de la réponse du coordinateur racine, le coordinateur local sélectionne l'événement extrait de son échéancier ou l'événement reçu. Dans le cas d'événements simultanés, si les événements sont dirigés vers différents simulateurs, le coordinateur local devra définir des règles de priorité entre les simulateurs ; en revanche s'ils sont dirigés vers un même simulateur, il lui fera parvenir un paquet « d'événements » simultanés.

```
When receive Dmessage ('d', Simulator, t, - , Value){
// inform when an Event as been processed by a
   hierarchical children
add ('d', Simulator, t, - , newTimelifeValue) in
StopList
if (WaitList != StopList)
then    Mark "send" first "non send" message of
        WaitList
        Send this message of WaitList
else    Remove these messages from WaitList and
        StopList
        Send Dmessage to Parent
Remove ('*', Simulator, next_internal_EventTime, - ,
- ) from Eq
// last internal event associated to this Simulator
if (newTimelifeValue != null) // state is transient
then    create next internal event // value given by
        Dmessage
        add ('*', Simulator, newTimelifeValue, - , -
        ) in Eq
tl = t
tn = t first Event of the EventList}
```

Figure 6 : Algorithme réception Dmessage du coordinateur local

Lorsque le coordinateur local reçoit un message d'acquittement (Dmessage) d'un successeur, il l'insère dans sa liste d'acquittement et compare sa liste d'attente à sa liste d'acquittement. Si le contenu des deux listes est différent, alors il existe des messages en attente d'émission, il les envoie donc aux successeurs. Si le contenu est identique, il vide le contenu des deux listes.
Il extrait ensuite du Dmessage le prochain événement interne prévu pour le simulateur émetteur, l'ordonne dans son échéancier et supprime l'éventuel ancien événement interne planifié subsistant en EventList.
Enfin, il reçoit simultanément la valeur du Lookahead du simulateur et calcule la nouvelle valeur de Lookahead pour le couple local modèle-port à faire parvenir au coordinateur racine distribué.

Lorsqu'un coordinateur reçoit de la part d'un successeur un événement de sortie (sous la forme d'un Ymessage), il vérifie, en fonction des relations de couplage, si un de ses successeurs est influencé par cet événement ; dans ce cas, il le transforme en événement d'entrée (Xmessage), le stocke en liste d'attente et le fait parvenir au (ou aux) successeur(s) concerné(s). Si le modèle local possède un port de sortie, un Ymessage est généré à destination du coordinateur racine.

```
When receive Ymessage ('y', Simulator, t,
Simulator_Port, Value){
Get-Internal-Influenced-By (Simulator,
Simulator_Port)
// check for internal coupling to get children
   influenced by output
for each children influenced by output do
        Add Xmessage ('x', Simulator_Influenced, t,
         Port_influenced, Value) in WaitList
        // according to priority definition

Send first message(s) (addressed to
Simulator_Influenced & with timestamp == t) of
WaitList & mark it(them) "send"
Get-External-Influenced-By (Simulator,
Simulator_Port)
// check for external coupling to see if there is an
   external output event
if (existing influenced model)
then    send Ymessage ('y', Coordinator, t,
        Coordinator_Port, Value) to Parent
tl = t
tn = t first Event of the EventList}
```

Figure 7 : Algorithme réception Ymessage du coordinateur local

5.3.2 Simulateur

Le simulateur calcule selon les fonctions définies par [Zeigler 00] le changement d'état et éventuellement l'événement de sortie conséquent à la réception d'un événement. Nous avons complété le code du simulateur par une fonction déterminant en outre une nouvelle valeur de Lookahead dépendante du modèle.

lookahead = min (DDV_s | s ∈ S) Avec S : ensemble des phases du modèle atomique. Le simulateur prend également en compte les Null message pour actualiser sa date et recalculer la valeur du lookahead.

Cette information est jointe au message de retour.

5.3.3 Coordinateur Racine Distribué

Initialement, les modèles couplés locaux s'enregistrent auprès du coordinateur racine en lui fournissant des informations sur leurs relations de couplages. Le coordinateur traite ensuite les demandes d'autorisation de traitement des messages locaux provenant des successeurs comme suit :

```
When receive for Next-Event-Request
(Children_Coordinator, InfluencedPort, LocalTime,
Advance_Requested_Time){
Get-Internal-Influencer-Of (Children_Coordinator,
InfluencedPort)
Compute-LBTS (Children_Coordinator, InfluencedPort)
// using the lookahead min of the influences of the
   model that is simulated by Children_Coordinator

if (there is a Message for Children_Coordinator in Eq
with timestamp t < LBTS & t < Advance_Requested_Time)
then     Add in WaitList (Message) & mark it "send"
         send (Message)
         // as an answer to the request
else     if (Advance_Requested_Time < LBTS)
         then     Add in WaitList Autorisation
         (Advance_Requested_Time) & mark it "send"
         send Autorisation (Advance_Requested_Time)
         // as an answer to the request
         else     if ((Advance_Requested_Time >= LBTS)
                  or (Advance_Requested_Time == null))
                  & (LocalTime =! LBTS)
                  then     Add in WaitList
                  nullmessage("null", coordinator,
                  LBTS, InputPort,-)
                  mark it "send
                  send nullmessage("null", coordinator,
                  LBTS, InputPort,-)"
                  // as an answer to the request
                  else     if (LocalTime == LBTS)
                           then     add
                                    (Children_Coordinato
                                    r, LocalTime,
                                    Advance_Requested_Ti
                                    me) in Wq}
```

Figure 8 : Algorithme NextEventRequest du coordinateur racine

Lorsque le coordinateur racine distribué reçoit une demande d'autorisation de traitement d'un événement local d'un coordinateur local dont la date actuelle est LocalTime qui souhaite traiter un événement de date Advance_Requested_Time. Il doit déterminer si le coordinateur local demandeur ne recevra pas d'influence avant Advance_Requested_Time.

L'algorithme du coordinateur racine définit donc, à partir des valeurs de Lookahead des modèles influenceurs, une date minimale (LBTS) jusqu'à laquelle le coordinateur influencé ne recevra pas d'événement. Finalement, en fonction de ces données, le coordinateur racine émet une réponse de type : Autorisation, Xmessage ou Nullmessage.

Lorsque le coordinateur racine reçoit un message d'acquittement, il l'insère dans sa liste de message d'acquittement, qu'il compare à sa liste d'attente et supprime les messages identiques dans les deux listes. Il extrait également le prochain événement interne de son successeur et met à jour son échéancier. Enfin, il récupère la nouvelle valeur de Lookahead de ce successeur qu'il utilise pour déterminer un nouveau LBTS pour les influencés du successeur émetteur de l'acquittement.

Lorsque le coordinateur racine reçoit un événement de sortie sous la forme d'un Ymessage, il le transforme en événement d'entrée (Xmessage) en fonction des relations de couplage et le stocke dans son échéancier.

5.4 Intégration de modèles couplés distribués dans un environnement HLA

Pour la mise en œuvre de l'environnement présenté ci-dessus, la spécification de simulation distribuée HLA paraît particulièrement adaptée. En effet, cette norme permet la réutilisation et l'interopérabilité de composants de simulation sans la nécessité de les recoder. De plus, l'implémentation de la spécification d'interface (RTI) propose des services qui peuvent inscrire ceux définis pour le coordinateur racine distribué. Enfin, les ensembles coordinateurs locaux et simulateurs semblent pouvoir être vus comme des fédérés agissant entre eux pour réaliser la simulation du modèle structurel global (Fédération). Voir Figure 9.

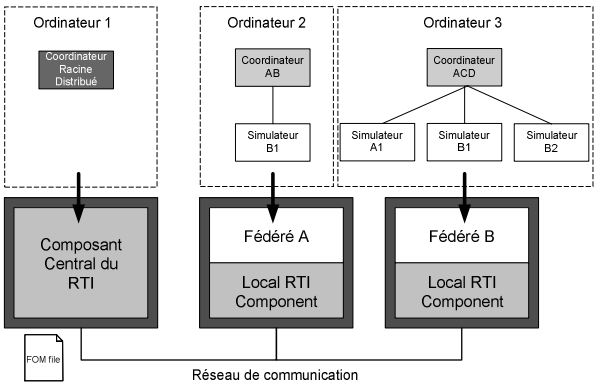

Figure 9 : Intégration du simulateur GDEVS distribué avec le RTI

5.5 Création d'une fédération en accord avec le FEDEP

Le FEDEP (HLA Federation Development and Execution Process) n'est pas une obligation de HLA mais il préconise six étapes dans le processus de création d'une fédération formalisant et aidant à réutiliser les informations de développement.

Le premier point consiste à définir les objectifs de la fédération ; dans notre cas il s'agit d'échanger des messages représentant des événements entre des coordinateurs locaux en respectant la causalité globale.

Il est donc nécessaire d'utiliser les services de synchronisation fournis par le RTI pour assurer l'échange cohérent des messages.
Les fédérés sont les simulateurs locaux composés chacun d'un coordinateur local et d'un certain nombre de simulateurs de modèles atomiques. Le coordinateur racine distribué présenté précédemment a pour fonction la récupération, l'organisation et la distribution des messages échangés au sein de la simulation globale. Dans HLA cette fonction peut être assurée par le RTI.
Le FOM de notre fédération contient donc un objet qui définit la partie saillante des simulations locales, en l'occurrence un objet « coordinateur local » avec pour attribut son nom et ses ports. Tous les fédérés publient « subscribeObjectClassAttributes » et souscrivent « publishObjectClassAttributes » à cet objet afin de connaître et de se faire connaître d'autres fédérés.
Le FOM contient également les interactions nommées « couplage » associées aux liens représentant la relation influenceur-influencé entre les modèles participant à la fédération. Ces interactions établissent la communication entre les fédérés et transportent donc les messages contenant les événements de sortie qui influencent d'autres modèles. Les modèles qui comportent un port de sortie publient donc ces interactions « publishInteractionClass » et ceux qui sont influencés par ces ports y souscrivent « subscribeInteractionClass ».
Pour respecter la causalité, les interactions entre fédérés sont définies « TimeStamped Order ». Elles sont donc émises avec une date associée au temps logique local du fédéré émetteur et sont stockées dans un échéancier du LRC avant d'être délivrées au souscripteur lorsque ce dernier sera temporellement en mesure de traiter ce message. Au moment de l'implémentation des concepts HLA dans chaque fédéré, le code héritant de la classe « Federate Ambassador » ajouté dans chaque fédéré, devra donc comporter un appel au service de demande d'autorisation de traitement du prochain événement. Cet appel « NextEventRequest » auprès du RTI permet d'obtenir l'autorisation de traiter le prochain événement local planifié dans le cas d'une simulation dirigée par les événements. Lorsque le RTI reçoit une telle demande, il détermine en fonction du LBTS et des messages à destination de ce fédéré, s'il peut lui accorder le droit de traiter son prochain événement. S'il autorise le fédéré à traiter le message demandé, il envoie « TimeAdvanceGrant », mais si le fédéré a des messages en attente de réception avant la date de son prochain événement local, le RTI les lui délivre.
Lorsque le fédéré reçoit une autorisation ou des messages du RTI, il les fait parvenir à ses successeurs. En retour, il reçoit la nouvelle date de ses modèles successeurs et éventuellement un message de sortie qu'il envoie au RTI avec le service « sendInteraction ». Si les attributs de l'objet partagé « coordinateur local » ont changé, il en informe la fédération avec le service « sendAttributesUpdates ».
L'intérêt des fédérés « GDEVS couplés » est, en particulier, leur capacité d'interfaçage avec des programmes hétérogènes générateurs d'événements compatibles HLA en amont et des programmes exploitant les sorties du modèle GDEVS global en aval.

6 CONCLUSION & PERSPECTIVES

Cette publication a présenté un environnement de simulation utilisant la précision de GDEVS dans un contexte distribué compatible HLA. La contribution clé de cet environnement réside dans la possibilité de coupler des modèles GDEVS distants entre eux et/ou directement avec d'autres simulations hétérogènes. La compatibilité HLA permet ainsi de réaliser des simulations globales, intégrant des entités locales notamment GDEVS qui communiquent entre elles sans dévoiler toutes leurs données et sans être recodées.
Le Lookahead étant un facteur de performance [Fujimoto 98], nous pourrons affiner le calcul de cette valeur en fonction de l'état courant du modèle et généraliser ce calcul pour des modèles dont les durées de vie dépendent de variables d'état.

BIBLIOGRAPHIE

[Bryant 77] Bryant R. E., "Simulation of packet communication architecture computer systems", Technical Report MIT/LCS/TR-188, MIT, (1977).
[Chandy 79] Chandy K. M., Misra J., "Distributed simulation: A case study in design and verification of distributed programs". IEEE Transactions on Software Engineering, Vol. SE-5 No.5, 1979, pp 440-452, (1979).
[Chandy 81] Chandy K. M., Misra J., "Asynchronous distributed simulation via a sequence of parallel computations". ACM, v.24.4, p.198-206, (1981).
[DMSO 98] DMSO : "High Level Architecture" (1998)
[Fujimoto 90] Fujimoto R. M., "Parallel discrete event simulation", ACM, 33(10): 30-53, (1990).
[Fujimoto 98] Fujimoto R. M., "Time management in the high level architecture". Simulation, vol. 71, no. 6, pp. 388-400, (1998).
[Giambiasi 00] Giambiasi N., Escude B., Ghosh S., "GDEVS A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems". SCS Transactions Volume 17, 3, p.120-134 (2000).
[Jefferson 85] Jefferson D. R, "Virtual Time". ACM, Vol 7, 3 (1985).
[Kim 00] Kim K., Kang W., Sagong B., Seo H., Yeungnam University "Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One" 33rd ASS, 2000 Washington, D.C. p. 227 (2000)
[Lamport 78] Lamport L., "Time, clocks and the ordering of events in a distributed system". Communication of the ACM, Vol 21, 7 (1978).
[Samadi 85] Samadi B., "Distributed simulation, algorithms and performance analysis". Phd, UCLA, USA, (1985).
[Zeigler 76] Zeigler B. P. "Theory of Modelling and Simulation". Wiley & Sons, New York, NY, (1976).
[Zeigler 00] Zeigler B. P., Praehofer H., Kim T. G., "Theory of Modeling and Simulation". 2nd Edition, Academic Press, New York, NY (2000).