GDEVS/HLA Environment: A Time Management Improvement

Gregory Zacharewicz, Norbert Giambiasi, Claudia Frydman

GDEVS/HLA Environment: A Time Management Improvement

G. Zacharewicz, N. Giambiasi, C. Frydman
LSIS UMR CNRS 6168
Université Paul Cézanne
Avenue Escadrille Normandie Niemen
13397 - Marseille cedex 20
Phone: +334 91 05 60 30, Fax: +334 91 05 60 33
E-mail: {Gregory.Zacharewicz, Norbert.Giambiasi, Claudia.Frydman}@lsis.org

Abstract - This paper presents a distributed discrete event simulation environment based on GDEVS and HLA concepts. The chosen local simulation structure is "flatten" to reduce the exchange of messages between simulation components regarding with classical structure of DEVS simulators. Moreover, we present an integration method to create GDEVS models HLA-compliant; for that purpose, we introduce an effective algorithm of conservative synchronization using the HLA lookahead and so improving the distributed simulation time management. Finally, we detail an example of distributed GDEVS coupled models to show the improvement due to the new algorithm.

Keywords: DEVS, GDEVS, Distributed Simulation, Synchronization, HLA, Lookahead.

I. INTRODUCTION

In this paper, we propose a new DEVS-GDEVS/HLA environment. Our solution improves GDEVS simulation by reducing the simulation structure. It also improves time management in the GDEVS/HLA integration with a new algorithm using the HLA lookahead and without moving coupling information from the RTI level to the federate level, regarding previous approaches as those proposed by [ZEI, 99] and [LAK, 00].

On one hand, the use of GDEVS [GIA, 00] is particularly suited for this environment. Indeed, GDEVS preserves the notion of coupled model [ZEI, 00] and uses polynomial functions that promises higher accuracies in modelling continuous processes as discrete event abstractions. Moreover, event driven simulations permit faster execution in contrast to continuous simulations. A key contribution of GDEVS is the possibility to develop uniform simulation environment for hybrid (i.e. both continuous and discrete) systems.

On the other hand, GDEVS models components composing a GDEVS coupled model might be implemented on different computers with diverse operating systems maybe geographically distant. It is therefore necessary to define a communication protocol to integrate these components into a distributed simulation. The HLA specification [IEEE1,2,3, 00] satisfies this need by specifying message-exchange between components. Rather, HLA defines services for message-exchange within distributed components respecting determinism and causality by using synchronization mechanisms.

The remainder of this paper is organized as follows. Section 2 gives recalls on DEVS/GDEVS formalism, general concepts of distributed simulation and HLA standard. Section 3 exposes the GDEVS/HLA components behaviour of our compliant environment, and in particular, a new algorithm. Then, an example that illustrates the use of the new algorithm is presented. Finally some conclusions and future works are given.

II. RECALLS

A. *Discrete Event Specification*

1) *Generalized Discrete EVent System Specification (GDEVS)*

Traditional discrete event abstraction (e.g. DEVS) approximates observed input-output signals as piecewise constant trajectory. GDEVS defines abstraction of signal with piecewise polynomial trajectory [GIA, 00]. Thus, GDEVS defines event as a list of values. These values represent polynomial coefficients that approximate the input-output trajectory. Therefore, DEVS is a particular case of GDEVS (i.e. an order zero GDEVS). The original signals representation of a dynamic system is thus more accurately modelled with GDEVS. Formally, GDEVS represents a dynamic system as an n order discrete event model with:

$$SED_N = \langle XM, YM, S, \delta_{int}, \delta_{ext}, \lambda, D\rangle$$

The following mappings are required:

XM = A^{n+1}, where A is a subset of integers or real numbers

YM = A^{n+1}

S = $Q \times (A^{n+1})$

For all total state $(q, (a_n, a_{n-1}, \ldots\ldots, a_0), 0)$ and a continuous polynomial input segment $w : \langle t_1, t_2\rangle \rightarrow X$, are defined:

The internal transition function:

$\boldsymbol{\delta_{int}}\,(q, (a_n, a_{n-1}, \ldots\ldots, a_0)) =$

Straj $q, x\ ((t1+D((q, (a_n, a_{n-1}, \ldots\ldots, a_0)), x)$

with $x = a_n t^n + a_{n-1} t^{n-1} + \ldots\ldots + a_1 t + a_0$ and Straj model state trajectory

$\forall$ q de Q et $\forall$ w :<t1, t2> $\rightarrow$ X,

Straj q,w:<t1, t2> $\rightarrow$ Q

The external transition function:

δ_{ext} (q, (a_n, a_{n-1},......, a_0), e, (a_n', a_{n-1}',......, a'_0)) =
(Straj q, x(t_l+e), x')
with: Coef (x) = (a_n, a_{n-1},......, a_0)
and Coef (x') = (a_n', a_{n-1}',......, a'_0)

Coef:	function to associates n-coefficient of all continuous polynomial function segments over a time interval <ti, tj>, to the constants (n+1) values (a_n, a_{n-1},......, a_0) such as: $w(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots\ldots + a_1 t + a_0$
InCoef:	λ (q, (a_n, a_{n-1},......, a_0)) = Λ(q, x)

The output function:
$\lambda: S \rightarrow A^{n+1}$

The function defining the life time of the states:
D(q, (a_n, a_{n-1},......, a_0)) =
MIN(e/Coef (Otraj q, x(t_l)) $\neq$
Coef (Otraj q, x(t_l + e))
with Otraj model output trajectory:
Otraj q,w:<t1, t2> $\rightarrow$ Y

2) Coupled model

Reference [ZEI, 00] has introduced the concept of coupled model that is a structural model. It describes a structure by interconnection of basic models. Every basic model of the coupled model interacts with the other models to produce a global behaviour. The basic models are, either atomic models, or coupled models stored in a library. The models' coupling is done using a hierarchical approach. Note that GDEVS models with same order events could be as well coupled.

A discrete event coupled model is defined by the following structure:

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

X:	set of external events.
Y:	set of output events.
D:	set of components names.
M_d:	DEVS models.
EIC:	External Input Coupling relations.
EOC:	External Output Coupling relations.
IC:	Internal Coupling relations.
Select:	defines priorities between simultaneous events intended for different components.

3) DEVS Simulator

The concept of abstract simulator has been developed in [ZEI, 00]. The architecture of simulation is diverted from the DEVS hierarchical model structure. An abstract simulator executes functions to express the dynamic of the model. Such a simulator is obtained by matching each element of a model to a component of the simulator. The originality of this method is the independence of simulators from models behaviour. Moreover, this simulator can also simulate same order GDEVS coupled models, the distinction lies only in the message structure.

The Components involved in a hierarchical simulation are **Simulators**, which insures the simulation of the atomic models by using the functions defined in DEVS or GDEVS, **Coordinators**, which insures the routing of messages between coupled models according to coupling definitions, and the **Root Coordinator**, which insures the global management of the simulation. It orders beginning and end of the simulation and manages the global clock.

The simulation runs thanks to exchange of specific messages between the different processors. The first three types of messages below represent the various events defined in DEVS.

Xmessage:	Represents an external event (e.g. coefficient-event vector in GDEVS)
***message**:	Represents an internal event,
Ymessage:	Represents an output event,
Imessage:	Initializes the model with all the default values chosen by the user. (This message has only a computing interest)

B. Distributed Simulation

The purpose of distributed simulation is to optimize computers use, to work on distant computers and/or to reuse existing simulations by interconnecting them. As well, a distributed treatment has to guaranty the temporal causality relations.

The causality thus imposes a **partial order** between events treated. [LAM, 78] defines a method based on local logical clocks. For any 'a' and 'b' events, if 'a' occurs before 'b' (a → b), implies that the local logical time C(a) must be strictly lower than C(b). To respect causality, synchronization mechanisms have been created.

A first kind of synchronization is called conservative (or pessimistic). Processors treat local events or received events from an influencer time stamped T (and increments its actual time) when they are sure not to receive any more events time stamped T' < T, and thus to respect the principle of causality. The firsts algorithms were proposed by [BRY, 77] [CHA, 79].

The other main kind is called optimistic. Processors treat events received, without being sure to have received all events until this date; this incertitude can induce treatment omissions of some events and therefore a violation of the causality constraint [SAM, 85]. If a processor receives an event time stamped earlier than its local current time, it uses the "Rollback" mechanism [JEF, 85]. For that purpose memory of reached states, events received and sent is needed.

C. HLA (High Level Architecture)

1) Aim and Definition

HLA was developed in 1995 by the American Department of Defense (DoD) Defense Modelling and Simulation Office (DMSO) to suit the military projects needs.

The High Level Architecture (HLA) is a software architecture specification for creating global simulations that include a variety of simulation programs. These programs must be able to be reusable and inter-work without recoding them.

In HLA, every participating simulation is called **federate**. A group of federates, interacting together, is named in HLA terms a **federation**. The set of HLA definitions was formalized by the creation of HLA 1.3 standard in 1996, which then evolved to HLA 1516 in 2000. Reference [DMSO, 98] and [IEEE1,2,3, 00] define HLA by three constituents:

HLA Rules insure the appropriate interaction of simulations in a federation. They also describe the responsibilities of federations and federates.

Object Model Template (OMT) supplies a common structure for the documentation of HLA object model. It contains the SOM (Simulation Object Model) which defines objects and interactions which can be used by a simulation when it participates in a HLA federation. It contains also the FOM (Federation Object Model) that defines elements effectively used between the simulations in this federation.

The **Interface Specification** defines the functional interfaces between the federate and the Run Time Infrastructure (RTI) that must be respected during the execution to obtain a HLA-compliant simulation. The RTI is the implementation of this specification. It supplies services required by a simulation to be “HLA-compliant”.

2) Implementation Components [IEEE2, 00]

A federate is a HLA-compliant program, the code of that federate keeps its original features but must be extended by other functions to communicate with other members of the federation. These functions are contained in the class code of *FederateAmbassador* and make interpretable by a local process the information received resulting from the federation. Therefore, the federate program code must inherit of *FederateAmbassador* to complete abstract methods defined in this class used to receive information from the RTI.

The “Local RTI Components code” (LRC) supplies external features to the federate for using RTI call back services such as the handle of objects and the time management. The implementation is the class *RTIAmbassador*, this class is used to transform the data coming from the federate in an intelligible format for the federation. The federate program calls the functions of *RTIAmbassador* to send data to the federation or to ask information to the RTI. Each LRC contains two queues, a FIFO queue and a time stamp queue to store data before delivering to the federate.

Finally, the Central RTI Component manages the federation notably by using the information supplied by the FOM to define Objects and Interactions classes participating in the federation. Object class contains object-oriented data shared in the federation that persists during the run time; Interaction class data are just sent and received.

A federate can, through the services proposed by the RTI, "Publish" and "Subscribe" to a class. A federate that "Publishes" intends to diffuse the creation of object instances and the update of the attributes of these instances. "Subscribe" is the intention of a federate to reflect attributes of certain classes of other federates.

3) HLA time management

Reference [FUJ, 98] defines that HLA proposes conservative and/or optimistic synchronization mechanisms; the RTI thus implements the following notions, (N.B. names depend on HLA standard version [DMSO, 98] [IEEE1,2,3, 00]):

Lookahead: delay given by an influencer processor to the federation. Federates certify not to emit a message before their actual logical time plus the lookahead value.

LBTS (Lower Bound on Time Stamp 1.3) or **GALT** (Greatest Available Logical Time 1516): Time stamp until a processor will not be influenced by other processors (i.e. minimum of its influencers lookaheads).

NextEventRequest(t) 1.3 or *NextMessageRequest(t)* 1516: (*NER(t)* or *NMR(t)*) asks for grant to the RTI to treat an event time stamped t. If the RTI *TimeAdvanceGrant*(t), the federate is sure to have received all events time stamped $t' \leq t$ but events emitted must be time stamped > t.

NextEventRequestAvailable(t) 1.3 or *NextMessage-RequestAvailable(t) 1516*: (*NERA(t)* or *NMRA(t)*) differs from previous definition in *TimeAdvanceGrant*(t). As an answer to *NERA*(t), the federate can (if its lookahead is null) emit an event time stamped t, on the other hand, the federate is not sure to have received all the events of t.

MNET (Minimum Next Event Time 1.3) or **LITS** (Least Incoming Time Stamp 1516): Federate LITS is a lower bound until which the federate will receive no message, this value is calculated from its GALT and the messages in transit not received yet by the federate (messages stored in the LRC queue).

II. GDEVS HLA COMPLIANT SIMULATION ENVIRONMENT

Simulating same order GDEVS or DEVS models does not differ in the simulation concept. For this reason, we use afterward DEVS simulation structures as reference for the distributed GDEVS environment introduced in this paper.

A. Local coupled models simulator GDEVS "flattened"

We keep the original modular hierarchical structure of DEVS coupled models that allows the model composition by reusing models stored in libraries. However, we propose a hierarchical "compact" simulation structure that differs from the abstract simulation hierarchical structure defined by [ZEI, 00] (e.g. Fig 1 a)). The transformation is done by the environment before to run the simulation.

We employ the works proposed by [KIM, 00] to reduce the hierarchical structure of intermediate coordinators between the DEVS Root Coordinator and Simulators. Elements remaining locally are a Local Coordinator linked to one or many atomic Simulator(s), this group is abbreviated by LCS (e.g. Fig 1 b)). Thus, the local Coordinator component skills must be extended to manage in an autonomous fashion the local simulation. Furthermore, the component Coordinator is renamed "Local Coordinator" (LC) to differ from the one of the original DEVS structure. The LC manipulates an event list containing local internal and external events. It keeps local Logical Time (LT). It manages the local simulation by selecting the next chronological message in its event list with regard to its current LT. Then, this selected message is sent to the concerned successor Simulator. Consequently, the GDEVS simulation "flattened" exchanges locally less messages between coordinators and simulators.

This simulation structure improvement is not a necessity of distributed simulation environment but it improves the local performance of the environment.

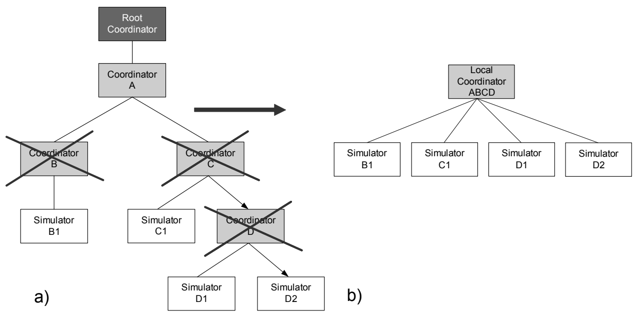

Fig. 1. Hierarchical simulation structure Flattened

B. Distributed simulation components structure

Because we are interested in executing the models composing a GDEVS coupled model on many distributed computers, the environment must be able to define a way of communicating between these distributed models.

According to classical distributed computers hardware platforms enounced in [FUJ, 00], one kind of platform is composed of simulation components linked to an interconnection network, these components communicates by messages passing. The structure of GDEVS distributed simulations generated by the proposed environment is based on this platform. The environment proposes to split a GDEVS coupled model into several distributed GDEVS models (i.e. each distributed GDEVS model is defined with the local structure defined in the above section).

The environment defines local LCS groups, which simulate distributed parts of the GDEVS coupled model split (e.g. Computer 2 and 3 in Fig 2 b)). Furthermore, LCS must intercommunicate. For that purpose, LC must also be able to manage messages resulting from others distributed LC.

For the global synchronization of the distributed simulation, the classical DEVS simulation Root Coordinator is transformed into a Distributed Root Coordinator (DRC) component, represented by the group Computer 1 Fig 2 b). The DRC is designed for routing the messages exchanged among the LCS; so it must exploit a synchronization mechanism for respecting the causality of events transmitted. It uses an Event List containing messages exchanged in the global simulation and a set of tables describing the coupling relations between distant models (EICList, EOCList, ICList).

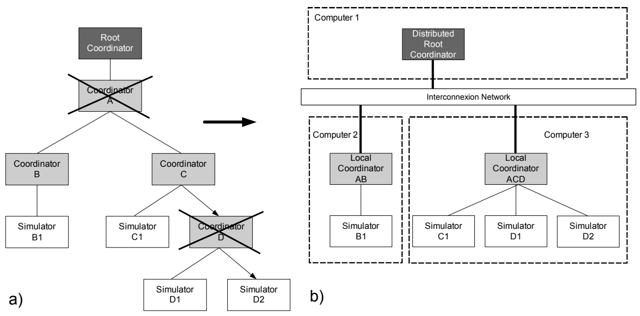

Fig. 2. Flattening and distribution of hierarchical simulation structure

C. Creating a HLA GDEVS Federation

For a compatibility and reusability purpose, the proposed environment generates HLA-compliant GDEVS simulations.

The environment follows the HLA Federation Development and Execution Process (FEDEP) [IEEE, 03] that proposes process steps in the federations creation. This formalization helps development information reuse.

First, the objectives of the federation have to be defined.

The common goal of all federation created by the environment consists in defining GDEVS coupled model.

As described in the FEDEP second step, the environment generates a conceptual model. It contains GDEVS models represented as entities and actions that represent external events exchanged between GDEVS Local Coordinators.

References [ZEI1, 98] [ZEI2, 98] [ZEI, 99] present a first integration of DEVS Coordinators in a HLA-compliant architecture. They map the local coupled models in HLA federates whose coordinators of higher level will have for responsibility to communicate with the "Time Manager" federate. As FEDEP third step, the environment conforms to [ZEI, 99] mapping of LCS into HLA federates, but does not use the "Time Manager" federate. It maps directly the DRC into the RTI. The reason of this mapping is the specification of interface (RTI) proposes services that enclose those defined in the DRC. Thus, the "global distributed" model (i.e. the federation) is constituted of federates intercommunicating (e.g. Fig. 3).

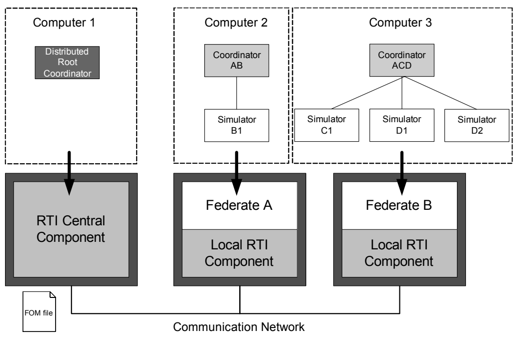

Fig. 3. DEVS integration with HLA Run Time Infrastructure

In the FEDEP fourth step, the environment integrates GDEVS models coupling relations into HLA interactions as in [ZEI, 99] and generates the associated FOM. Indeed, a GDEVS model possessing an influencer output port "publishes" on an Interaction Class defining the coupling relation by *publishInteractionClass*(). A GDEVS model possessing an influenced input port "subscribes" to the Interaction Class published by the port which influence it by using *subscribeInteractionClass*(). The environment may also generate, in the FOM, an object that defines the shared elements of local simulations; for tracking some state variables values. Tracked federate publishes using *RTIAmbassador publishObjectClassAttributes*() and tracker federate subscribes using *subscribeObjectClassAttributes*().

To respect the causality, the interactions among federates are defined "Time Stamped Order". They are emitted with a timestamp related to the local logical time of supplier federate in order to the RTI handles them respecting the causality.

In consequence, the "coupled GDEVS" federates generated by the environment can be reused and interfaced with varied programs HLA-compliant. For instance, "event generators" federates can be upstream connected to "coupled GDEVS" federates and "GDEVS outputs user" federates can be downstream connected.

D. DEVS/HLA integrating Algorithms

The first HLA/DEVS integration algorithm is presented by [ZEI, 99]. To guarantee the global synchronization of the Local Coordinators, the authors choose to exploit a conservative algorithm mechanism based on [BRY, 77] and [CHA, 81] proposed by the HLA specification. This method presents the advantage of reducing message exchange regarding to the Rollback situations of the optimistic approach proposed in [JEF, 85].

In the implementation of the HLA concepts of [ZEI, 99], DEVS federates code inherits from the *FederateAmbassador* class to be able to use HLA services. Federates contain *NER(t)* calls to the RTI to demand of treatment their next local event. When the RTI receives such a demand, it determines according to the LBTS and messages stored in LRC's queues of the concerned federate, if it can grant this federate to treat its next event. If the RTI allows the federate to treat the expected message, it sends *TimeAdvanceGrant*(t). If the RTI contains message, intended to this federate, time stamped t' earlier than the time stamp of its next local event, the RTI delivers it to the federate before to provide the *TimeAdvanceGrant*(t').

When a federate receives a message and/or a grant from the RTI, it sends the received or first of its event list message to its child. In return, it receives the new logical time and next internal event of its child model and possibly an output message that it sends to the RTI with *sendInteraction*() service. If shared objects have been defined, their attributes change, the federate informs the federation about it with *sendAttributesUpdates*() service.

The gap of this solution results from the treatment of the simultaneous events by federates that are influenced and influencer. *TimeAdvanceGrant*(t) is granted to an imminent federate (i.e. a federate that can publish Time stamped objects or interactions). Federate not imminent has to wait for all imminent federate emitted their interactions of timestamp $t' \geq t$ to receive *TimeAdvanceGrant*(t). In the case of cyclic influences between federates, it is impossible to determine which to deliver *TimeAdvanceGrant*() first.

Using works cited in the above paragraphs in reference, [LAK, 00] present two approaches of integrating DEVS models in HLA. They notably introduce a distinction between "DEVS lookahead" and "HLA lookahead". DEVS lookahead = Min $D(S)$ / $s \in S$ with S set of model states. "HLA Lookahead" is a minimum delay from the treatment of an event to the emission of the output event associated. The authors present in this solution a direct implementation of DEVS into HLA that resolves the problem found by [ZEI, 99]. They use the *NERA*(t) service. This solution uses, as the previous one, a null HLA lookahead for every federate.

Reference [LAK, 00] second solution proposes to broadcast the events messages among the federates and to give to all the federates a global awareness of the coupling relations. The local entities make their decisions regarding to their history of received messages and to their knowledge of the coupling relations. In that case, it is possible to define a positive not null HLA lookahead. We do not take into account this last solution because it transfers responsibilities of the RTI towards the simulators entities. As a result, we consider that these responsibilities migrations short-circuit some RTI function.

E. GDEVS/HLA integrating Algorithms

From the first algorithm of [LAK, 00], we propose a solution integrating the use of the HLA lookahead. This solution could be applied to GDEVS or DEVS coupled model as order zero GDEVS.

Consider a local GDEVS coupled model integrated in a HLA federate. This federate communicates with the other GDEVS models within the federation. The actual logical time of this federate is "Tact" and it possesses a next local event planned in its local event list at "TnextLocal". We set the federate lookahead value = Min D(S) / s $\in$ S where S is the set of model States. It is to note that we use, similarly as the previous solutions, GDEVS models with constant D(S) > 0.

Moreover, we state that, in the case of simultaneous events, we choose to treat first the internal event independently of the others, then, after having emitted an output event and done state changes, we process simultaneous external events using a confluent function. Note that our pseudo-code is designed for RTI 1516.

We propose a pseudo-code algorithm in Fig. 4 that uses the *queryLITS*() RTI service defined in the HLA standard. In many cases, depending on influencer data and on the use of *queryLITS*(), a federate can preserve a not null lookahead value. In consequence, it frees of constraint federates under its influence for a period equal to the lookahead. Thus, this situation increases the parallelism of the global simulation.

```
Do queryLITS()
          If (TNextLocal ≤ LITS)
                ComputeOutput() // associated to next local internal event timestamped TNextLocal
                SendInteraction(TnextLocal) // send output without reducing federate Lookahead
                NMRA(TNextLocal) // RTI 1516 NextMessageRequestAvailable(TNextLocal) and then wait for RTI answer

          Else // if (TNextLocal > LITS)
                NMRA(TnextLocal - Lookahead)
                WaitUntil(RTI responds callback)
                If (TimeAdvanceGrant (TNextLocal - Lookahead))
                      queryLITS()
                      If ((TNextLocal) > LITS)
                            ModifyLookahead(zero)
                      Else // If ((TNextLocal) ≤ LITS)
                            ComputeOutput() // associated to next local internal event timestamped TNextLocal
                            SendInteraction(TnextLocal) // send output without reducing federate Lookahead
                      NMRA(TNextLocal) // then wait for RTI answer

                Else If (ReceiveInteraction(T'≤ (TnextLocal - Lookahead)) & TimeAdvanceGrant(T'))
                              Do
                                        NMRA(T'+ε) // guaranty to have received simultaneous event timestamped T'.
                                        WaitUntil(RTI responds callback)
                                        If (TimeAdvanceGrant(T'+ε))
                                              ComputeExternalTransition() // associated to external event(s) timestamped T'
                                              Break to beginning // with new TnextLocal.
                                        Else If ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                                                      AddtoSimultaneousMessageList()
                              While (TimeAdvanceGrant(T') < T'+ε)

          WaitUntil(RTI responds callback)
          If (TimeAdvanceGrant(TNextLocal))
                  If (output not already sends with positive lookahead)
                        ComputeOutput() // associated to next local internal event timestamped TNextLocal
                        SendInteraction(TNextLocal) // send output with zero federate Lookahead
                  ComputeInternalTransition() // associated to internal event timestamped TNextLocal
                  Do
                        NMRA(TNextLocal+ε) // guaranty to have received simultaneous event timestamped TNextLocal.
                        WaitUntil(RTI responds callback)
                        If (TimeAdvanceGrant(TNextLocal+ε))
                              ComputeExternalTransition() // associated to eventual external event(s) timestamped T
                              ModifyLookahead(min of D(S))
                              Break to beginning
                        Else if ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                                      AddtoSimultaneousMessageList()
                  While (TimeAdvanceGrant(TNextLocal) < TNextLocal +ε)

          Else If ReceiveInteraction(T<TNextLocal) & TimeAdvanceGrant(T)
                        Do
                              NMRA(T+ε) // guaranty to have received simultaneous event timestamped T.
                              WaitUntil(RTI responds callback)
                              If (TimeAdvanceGrant(T+ε))
                                        ComputeExternalTransition() // associated to external event(s) timestamped T
                                        Break to beginning // with new TnextLocal.
                              Else if ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                                            AddtoSimultaneousMessageList()
                        While (TimeAdvanceGrant(T) < T +ε)

While (Simulation not end)
```

Fig. 4. Federate Algorithm

F. *GDEVS/HLA Communicating Example*

Consider an order zero GDEVS coupled model, illustrated Fig. 5 using the graphical notation defined by [SON, 94]. The states of the model are represented by the nodes of the graph. Every node contains the name of the phase and the associated life time. Solid arcs represent the external transitions and dotted arcs represent the internal transitions.

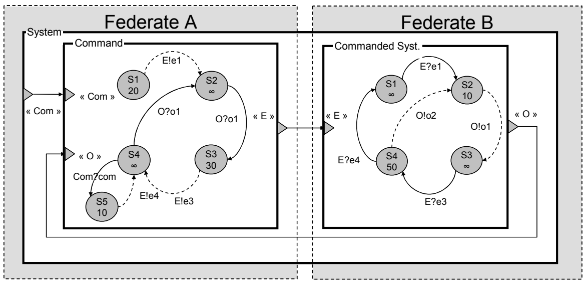

Fig. 5. Distributed coupled models example

The coupled model System (S) is defined by:

$\text{Iports}_S = \{\text{Com}\}$ and $\text{Oports}_S = \{\emptyset\}$

D = {Command, Commanded Syst.}

EIC = {((System, Com), (Commanded Syst., Com)),

IC = {((Command, E), (Commanded Syst., E)),
((Commanded Syst., O), (Command, O))}

The atomic model Command is defined as follow:

S = {S1, S2, S3, S4, S5} ; XM = {Com, O} ;
YM = {E} ; s° = S1

The atomic model Commanded Syst. is defined as follow:

S = {S1, S2, S3, S4} ; XM = {E} ;
YM = {O} ; s° = S1

We wish to execute models Command and Commanded Syst. on two distant computers. For that purpose, the environment proposes to integrate them into a HLA-compliant distributed simulation, to create a federation and to fill an associated FOM.

Thus the federation contains two federates associated to both coupled models, the federate A (FA) contains the Command GDEVS model and federate B (FB) contains the Commanded Syst. GDEVS model.

The FOM contains objects and interactions classes shared into the federation. In the considered case, the attributes of both models are mapped as objects and the coupling relations are mapped as interactions. The federation thus contains two interactions "E" and "O". FA subscribes to the interaction O and publishes on the interaction E; while FB subscribes to the interaction E and publishes on the interaction O. Note that the SOM of FA also contains an interaction "Com" not includes in the FOM considered because no federate will publish on it.

Then, the federates define their lookahead values. In the studied case, FA and FB both have a lookahead of 10 time units; it will thus pass by at least 10 time units between a message reception and a message emission as a result of the received message. Federates are both time regulating and time constrained at the same moment, so they can send and receive "Time Stamped" messages. The environment defines in the program of every federate an algorithmic loop, presented in the above section, to select the next local event of the federate and to interrogate the RTI about the authorization to treat it.

Fig. 6 presents the communication, through the RTI, between FA and FB of Fig. 5. This example illustrates the interest of using the service *queryLITS*() as proposed in Fig. 4. Numbers represent units of logical time, corresponding in the schema to federates logical times or sent messages timestamp. Arcs represent causal relations. Note that the temporal comparisons of the example are done with regard to a "wall clock" time as defined in [FUJ, 00] which measures the time during the execution of the simulation (e.g. in Fig. 6, the alphabetical order of letters surrounded represents a wall clock time order).

We focus on the succession of causal relations, depicted in Fig. 6 by dotted arcs and surrounded letters, where FA asks *queryLITS*() (noted e) after the RTI has received *queryLITS*() (noted a) and *NMRA*(∞) (noted c) of FB.

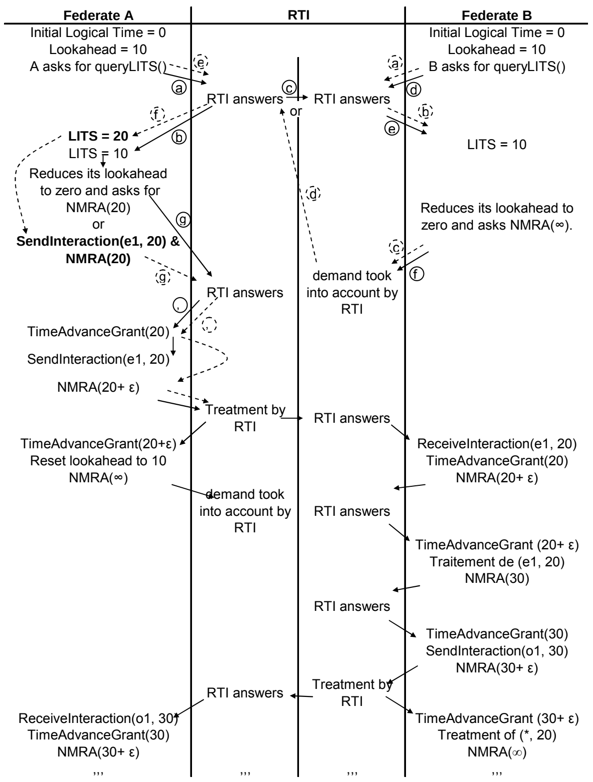

Fig. 6. Communication between federates trough RTI

On the one hand, FB has informed with *NMRA*(∞) (noted c) the RTI, that it would not emit output message on "O" at time stamp 0 (i.e. no internal transition associated to the current state). Thereby, the next output message of FB must be a consequence of a received message on port "E". Furthermore, received messages of FB result from FA and FA has informed the RTI that its logical time is zero, so its next emitted message will be dated minimum 10 because of FA lookahead. In result, the next outgoing message of FB will be at least dated equal to:

$\text{FB LITS}_{\text{(Min (FA lookahead(=10), RTI Message time for FB))}} + \text{FB lookahead}_{(=10)} = 20$

On the other hand, FA LITS is computed by the RTI regarding to FA HLA-subscriptions to the interaction "O" published by FB. As a response to *queryLITS*() (noted ⓔ), the RTI computes the LITS of FA equal to least output message of FB: 20. Therefore, the federate FA can emit at once its output message planned at timestamp 20 (noted ⓖ) and preserve a lookahead of 10 time units because we defined a priority to internal event in case of simultaneity.

A federate that conserves a not null lookahead releases of constraint its influenced federates for a period equal at least to its lookahead and so improves the simulation parallelism. Note that this situation is very desirable but not occurs all the time. Sometimes, depending on various federates wall clock time progression, federates must reduce their lookahead to zero as described by full circled letters and plain arcs sequences represented in Fig. 6.

III. FUTURE WORK

Reference [FUJ, 98] defined the lookahead as a performance factor for distributed simulations. For this purpose, we think that refining the computation of this value could speed up the simulation.

Indeed, we use a minimal lookahead value for distributed GDEVS models. Our current work consists in improving the lookahead computation to find maximal lookahead for GDEVS models with explicit states and constant D(S).

Distributed GDEVS models, wherein D(S) depends on state variables, are also under our scope. We will try to define methods to compare D(S) functions of the different model states to obtain a maximal value of the lookahead.

IV. CONCLUSION

In this paper, a GDEVS HLA-compliant simulation environment has been proposed. The key contribution is a new HLA integrating algorithm that uses conservative synchronization mechanism and the HLA lookahead. This algorithm does not move decision functions of the RTI towards local GDEVS simulators. In addition, we present a "flatten" GDEVS simulation structure that reduces message exchanges and so improves execution speed. These two propositions improve the performance of the distributed simulation.

Finally, this GDEVS HLA-compliant simulation environment generates distributed GDEVS models that can be integrated into heterogeneous HLA-compliant programs with respect of time management constraints.

REFERENCES

[BRY, 77] Bryant R.E., "Simulation of packet communication architecture computer systems", Technical Report MIT/LCS/TR-188, MIT, 1977.

[CHA, 79] Chandy K.M., Misra J., "Distributed simulation: A case study in design and verification of distributed programs", IEEE Transactions Software Engineering, Vol. SE-5 No.5, pp 440-452, 1979.

[DMSO, 98] DMSO : High Level Architecture, DMSO, 1998

[FUJ, 97] Fujimoto R.M., "Zero lookahead and repeatability in the high level architecture", Spring Simulation Interoperability Workshop, Orlando, FL, 3-7 March 1997.

[FUJ, 98] Fujimoto R.M., "Time management in the high level architecture". Simulation, vol. 71, no. 6, pp. 388-400, 1998.

[FUJ, 00] Fujimoto R.M., "Parallel discrete event simulation", Fujimoto, R.M., Wiley Interscience, January, 2000.

[GIA 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.

[IEEE1, 00] IEEE std 1516-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules" The Institute of Electrical and Electronic Engineers, 2001.

[IEEE2, 00] IEEE std 1516.1-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification" The Institute of Electrical and Electronic Engineers, 2001.

[IEEE3, 00] IEEE std 1516.2-2000, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification" Institute of Electrical and Electronic Engineers, 2001.

[IEEE, 03] IEEE std 1516.3-2003, "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federation Development and Execution Process (FEDEP)" The Institute of Electrical and Electronic Engineers, 2003.

[JEF, 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

[LAK, 00] Lake, T., B.P. Zeigler, H.S. Sarjoughian, J. Nutaro, "DEVS Simulation and HLA Lookahead", Simulation Interoperability Workshop (SIW), 00S-SIW-160, 2000.

[LAM, 78] Lamport L., "Time, clocks and the ordering of events in a distributed system". Communication of the ACM, Vol 21, 7, 1978.

[SAM, 85] Samadi B., "Distributed simulation, algorithms and performance analysis". Phd, UCLA, USA, 1985.

[SON, 94] Song H.S. Gon K T. "The DEVS framework for discrete event systems control" 5th Annual Conference on AI, Simulation and Planning in High Autonomous Systems (Gainesville, FL, USA), pp. 228-234. 1994.

[ZEI, 76] Zeigler B.P. Theory of Modelling and Simulation. Wiley & Sons, New York, NY, 1976.

[ZEI1, 98] Zeigler B. P., J. S. Lee, "Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment", Proc. SPIE Vol. 3369, p. 49-58, Enabling Technology for Simulation Science II; Alex F. Sisti; Ed. Aug 1998

[ZEI2, 98] Zeigler, B.P., G. Ball, et al. "The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering." ECE Dept., UA, Tucson, AZ, DARPA Contract N6133997K-0007. 1998.

[ZEI, 99] Zeigler, B.P., G. Ball, H.J. Cho, J.S. Lee. "Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions." Simulation Interoperation Workshop (SIW), Orlando, FL, 1999.

[ZEI, 00] Zeigler B.P., Praehofer H., Kim T. G., "Theory of Modeling and Simulation." 2nd Edition, Academic Press, New York, NY 2000.