Mapping PIOVRA in GDEVS/HLA Environment

Gregory Zacharewicz, Claudia Frydman, Norbert Giambiasi
LSIS UMR CNRS 6168
Université Paul Cézanne
Avenue Escadrille Normandie Niemen
13397 - Marseille cedex 20, FRANCE
{gregory.zacharewicz, claudia.frydman, norbert.giambiasi}@lsis.org

Keywords: DEVS, G-DEVS, Distributed Simulation, HLA.

Abstract

The aim of this paper is to specify the G-DEVS / HLA Environment developed within the PIOVRA project.

1. INTRODUCTION

In the following sections, we introduce in a first part the components involved in the distributed G-DEVS environment. In a second part, we give a detailed specification of data, behavior and function for each component of the environment presented in this document. In the last section, we present the G-DEVS HLA compliant environment.

2. RECALLS

2.1. G-DEVS

Traditional discrete event abstraction (e.g. DEVS) approximates observed input-output signals as piecewise constant trajectories. G-DEVS defines abstractions of signals with piecewise polynomial trajectories [4]. Thus, G-DEVS defines the coefficient-event as a list of values representing the polynomial coefficients that approximate the input-output trajectory. Therefore, a DEVS model is a zero order G-DEVS model (the input-output trajectories are piecewise constants).

G-DEVS possesses the concept of coupled model introduced in [12]. Every basic model of a coupled model interacts with the other models to produce a global behavior. The basic models are, either atomic models, or coupled models stored in a library. The model coupling is done using a hierarchical approach.

The concept of abstract simulator of [12] to define the simulation semantics of the formalism can be used for G-DEVS models. The architecture of the simulator is derived from the hierarchical model structure. Processors involved in a hierarchical simulation are Simulators that insure the simulation of the atomic models, Coordinators, which insure the routing of messages between coupled models, and the Root Coordinator, which insures the global management of the simulation. The simulation runs by exchanging specific messages (corresponding to different kind of events) between the different processors. The specificity of G-DEVS model simulation is that the definition of event is a list of coefficient values as opposed to a unique value in DEVS.

2.2. Distributed Simulation System: HLA (High Level Architecture)

The High Level Architecture (HLA) is a software architecture specification that defines how to create a global simulation composed of distributed simulations. In HLA, every participating simulation is called federate. A federate interacts with other federates within a HLA federation, which is in fact a group of federates. The HLA definitions set gave place to the creation of the standard 1.3 in 1996, which then evolved to HLA 1516 in 2000 [6].

The interface specification of HLA describes how to communicate within the federation trough the implementation of HLA specification: the Run Time Infrastructure (RTI).

Federates interact among them using the services proposed by the RTI. They can notably “Publish” to inform about an intention to send information to the federation and “Subscribe” to reflect some information created and updated by other federates. The information exchanged in HLA is represented in the form of classical object oriented programming. The two kinds of object exchanged in HLA are Object Class and Interaction Class. The first kind is persistent during the simulation, the other one is just transmitted between two federates. These objects are implemented with XML format. More details on RTI services and information distributed in HLA are presented in [6].

In order to respect the temporal causality relations in the simulation; HLA proposes to use classical conservative or optimistic synchronization mechanisms [3].

3. TRANSFORMATION OF WORKFLOW SPECIFICATIONS INTO G-DEVS MODELS

3.1. Local Coupled Models and Simulators G-DEVS « flattened »

We based the environment on the Generalized Discrete EVent System Specification (G-DEVS) formalism [4]. It defines behavioural (atomic) models and, by hierarchical composition, structural (coupled) models (e.g. Figure 1 left hand).

The concept of abstract simulator used by G-DEVS is based on [12] to define the simulation semantics of the formalism. The architecture of the simulator is derived from the hierarchical model structure as illustrated in Figure 1 right hand.

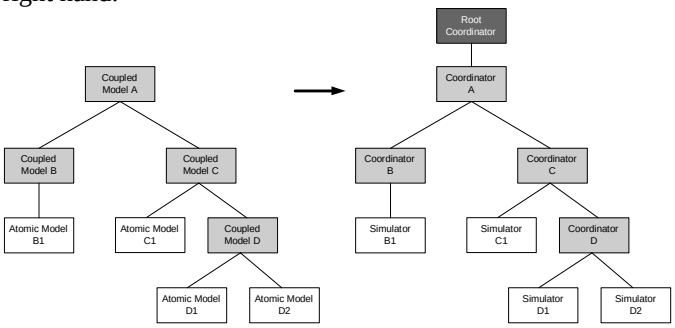

Figure 1. DEVS G-DEVS hierarchical modeling and simulation structure

The processors involved in a hierarchical simulation are Simulators, which insures the simulation of the atomic models, Coordinators, which insures the routing of messages between coupled models, and the Root Coordinator, which insures the global management of the simulation (e.g. Figure 1. a, without considering crosses out). The simulation runs by exchanging specific messages (corresponding to different kind of events) between the different processors. The simulator for G-DEVS models is similar to DEVS in its structure.

From the hierarchical structure of abstract simulation defined by [12], we propose a hierarchical "compact" structure for the PIOVRA G-DEVS / HLA Environment. We based our specification on the works proposed by [7] with the aim of decreasing the local exchange of messages between the coordinators and the simulators. We reduce the classical G-DEVS hierarchical structure [2] of intermediate Coordinators between the Root Coordinator and Simulators, at two hierarchical levels in our G-DEVS simulation structure. Thus, components remaining locally are a Local Coordinator (LC) and a set of atomic Simulators linked as direct successors (named LCS). The example Figure 2 right illustrates the LCS groups. For more details on Flattening techniques, please refer to [5], [7] and [9].

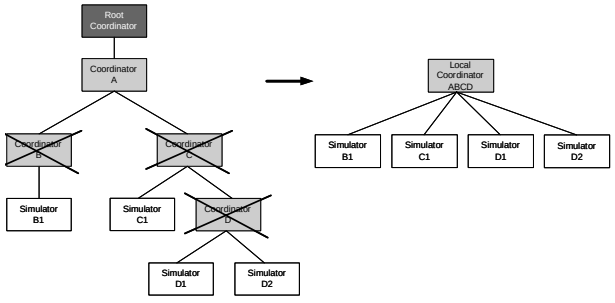

Figure 2. "Flattening" hierarchical simulation structure

Then, because the various models can be executed on distant computers, we require a technique to interconnect distributed models.

4. DISTRIBUTED SIMULATION COMPONENTS

Our goal is to obtain a distributed simulation environment. To achieve this objective, several LCS groups, defined previously, must intercommunicate to obtain a global distributed simulation (e.g. Figure 3 Computer 2 and 3). For that purpose, it is necessary to manage messages exchanged between the distributed components.

According to classical Distributed Computers hardware platform [3], the entities involved in a generic distributed simulation are described in Figure 3, we assumed that the different computers will be interconnected thanks to an interconnection network.

The LC manipulates local events regarding its local Logical Time and manages its local simulators. In the case of integrating this local coordinator in a distributed simulation, this component will also have to manage the events resulting from other distant LCS group.

To achieve the global synchronization of the simulation, a Distributed Root Coordinator (DRC) needs to be added. DRC corresponds in the example Figure 3 to group Computer 1. This DRC is designed for routing the events exchanged between the LCS groups; DRC has also to synchronize the sending-reception of these events by respecting the causality of the events treatment. DRC uses an event list containing events exchanged in the global simulation and a set of tables describing the coupling relations between the distant coupled models (EICList, EOCList, ICList).

The distant LCSs communicate by events passing trough the DRC. On receiving an event from a LC, DRC transforms it regarding to coupling relations from output to input events and inserts it in its EventList. Then, when a LC asks for event pending to be delivered, the DRC delivers it regarding to Synchronization imperatives in order to respect causality.

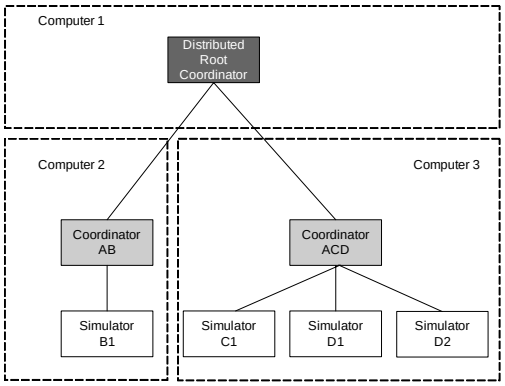

Figure 3. Distributed simulation structure

These modeling and simulation components presented above will be detailed in the next section of this document.

5. DETAILED ENVIRONMENT SPECIFICATION

5.1. Data Model

We define UML class diagrams to describe the data structure of the environment by showing the environment classes and the relationships between them.

In UML, a class is represented by a box with the name of the class written inside it. A compartment below the class name shows the classes attributes:

- Variables (prefixed with -)
- Functions of the class (prefixed with +).

Each attribute is described with at least its name, and optionally with its type, initial value, and other properties.

We use two types of logical connections on the class diagrams of this document:

- **Generalization-Specialization**
  A "solid line with a large hollow triangle" used to connect lines between two classes denotes the generalization (or inheritance) relationship.
- **Association**
  A direct line between two classes denotes the association relationship; it indicates that (at least) one of the two related classes refers to the other. This association relationship is also known as the "has a" relationship.

A notation at each end of the association relationship conveys cardinality/multiplicity of each class by indicating the multiplicity of instances of that entity (the number of objects that participate in the association). Common multiplicities are:

- 0..1 No instances, or one instance
- 1 Exactly one instance
- 0..* or * Zero or more instances
- 1..* One or more instances

G-DEVS Atomic and Coupled Data Model

The class diagram depicted in Figure 4 presents the data used to define G-DEVS atomic model and a coupled model.

In the atomic description, we present the state variables involved in particular the phase use to de-scribe graphical models. We define also an attribute to define the order of the event that can be treated by the model.

In the coupled models, we define two coexistent structures, the first is the hierarchical structure representation of the model (used in the modular modeling) and the non-hierarchical equivalent structure (used in the simulation for performance purpose as described in § 2.1).

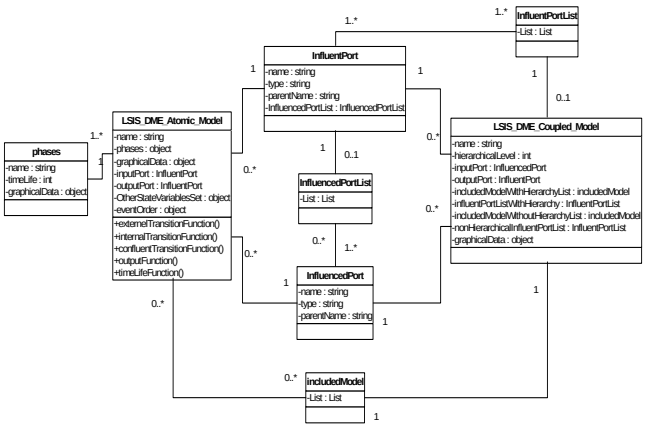

Figure 4. G-DEVS Atomic and Coupled Data Model

Simulator and Local Coordinator Data Model

We separate the modelling part from the simulation part as recommended in [12]. Figure 5 class diagram represents the local coordinator structure; it contains in particular the lists used to handle events and the coupling relations acquired from the model definition.

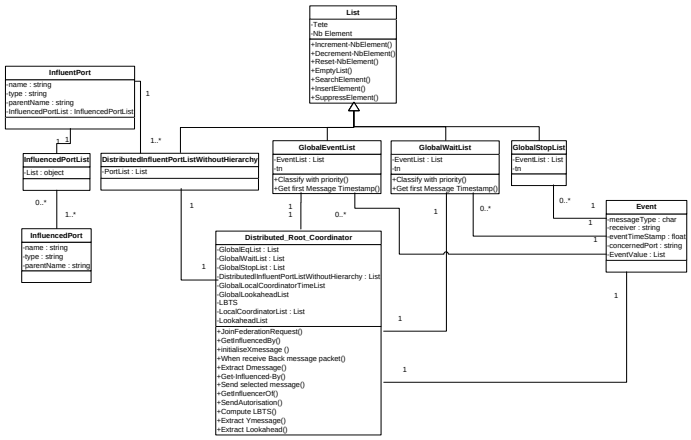

Figure 5. Simulator and Local Coordinator Data Model

5.2. Function model

The functions are presented in the form of pseudo code algorithms. These algorithms detail the functions of the component involved in the simulation: Root Coordinator, Local Coordinator and Simulator. In this paper, we detail the Function of LC and S in Figure 6 and Figure 7. DRC functions are not detailed because, in the implementation, HLA RTI will handle these features. Details of RTI functions can be found in [6].

The comments are noted with the prefix '//', the classical control structure are noted in bold in the pseudo code. The main function involved into the environment, are presented by underlined words. The data used referred to the class diagram presented above.

```
Eq                          // queue to store the Event List
Lq                          // queue to store the lookahead of coordinators Ports
Wq                          // queue to store the List of coordinators waiting
LBTS                        // value of lower bound time stamp
Event                       // Event message structure (Message Type, addressee/transmitter, event timestamp, concerned Port, Event Value)
WaitList                    // queue to store the Event sent
StopList                    // queue to store the Event treated
EICList                     // queue to store the list of couple (Root Coord, Port) (Coord, Port) that express the coupling between the global model
                            Input Port and the local Model Input Port
EOCList                     // queue to store the list of couple (Coord, Port) (Root Coord, Port) that express the coupling between the local Model
                            Output Port and the global model Output Port
ICList                      // queue to store the list of couple (CoordA, PortA) (CoordB, PortB) that express the coupling between a local Model
                            Output Port and another local model Intput Port
Value                       value in YMessage and Xmessage is a list of value for G-DEVS

When receive Ymessage ('y', Children_Coordinator, t, Children_Coordinator_Port, Value)
        Get-Internal-Influenced-By (Children_Coordinator, Children_Coordinator_Port)
            for each children influenced by output // According to ICList
                    Add Xmessage ('x', Children_Coordinator_Influenced, t, InfluencedPort, Value) in Eq // According to priority def
        Get-External-Influenced-By (Children_Coordinator, Children_Coordinator_Port)
        Print Output Event (Message)

When receive Dmessage ('d', Children_Coordinator, t, - , Value)
        Add to StopList
    // check for Message in process in WaitList
        if (Message also WaitList)
        then    remove this message from WaitList and StopList
                Print Event (Message also WaitList) done
        else    routing error

When receive Lookahead message ('l', Children_Coordinator, t, OutputPort, Lookahead Value)
        add Lookahead (Children_Coordinator, OutputPort, Lookahead Value) in Lq
        Get-Internal-Influenced-By (Children_Coordinator, Children_Coordinator_Port)
        if      ((Children_Coordinator influences Another_Children_Coordinator) & (Another_Children_Coordinator, InputPort, LocalTime, Ad-
                vance_Requested_Time) is in Wq
        // a coordinator influenced is waiting for Next-Event grant
        then    Remove Another_Children_Coordinator from queue Wq
                Compute-LBTS (Another_Children_Coordinator, InfluencedPort)
                    // using the lookahead min of the influences of the model that is simulated by
                        Another_Children_Coordinator

                if (there is a Message for Another_Children_Coordinator in Eq with timestamp t < LBTS & t < Advance_Requested_Time)
                then    Add in WaitList (Message)
                        send_selected (Message) & mark it "send"
                        // 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,-)
                                        send_nullmessage("null", coordinator, LBTS, InputPort,-) & mark it "send"
                                        // as an answer to the request
                                else    if (LocalTime == LBTS)
                                        then add (Another_Children_Coordinator, LocalTime, Advance_Requested_Time) in Wq

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_selected(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_Coordinator, LocalTime, Advance_Requested_Time) in Wq

When receive for Join-Federation-Request (Coordinator, EIC, EOC, IC)
                                                // A local Coordinator ask to join a federation.
                                                // EIC List of couple of External Input Coupling
                                                // EOC List of couple of External Output Coupling
                                                // IC List of couple Internal Coupling
Add EIC couple ((Root Coord, Port) (Coord, Port)) in EICList
Add EIC couple ((Coord, Port) (Root Coord, Port)) in EOCList
Add EIC couple ((CoordA, PortA) (CoordB, PortB)) in ICList

B = Get-Internal-Influenced-By (A)              // According to the couple ((CoordA, PortA) (CoordB, PortB)) in the IC List, the Output
                                                // PortA influences the Input PortB
A = Get-External-Influenced-By (C)              // According to the couple ((CoordC, PortC) (CoordA, PortA)) in the EIC List, A is an
                                                // internal model of C and the Intput PortC influences the Input PortA
A = Get-Internal-Influencer-Of (B)              // According to the couple ((CoordA, PortA) (CoordB, PortB)) in the IC List, the Intput
                                                // PortB is influenced by the Output PortA
B = Get-External-Influencer-Of (C)              // According to the couple ((CoordB, PortB) (CoordC, PortC)) in the EOC List, B is an
                                                // internal model of C and the Outtput PortB influences the Output PortC
Compute-LBTS (Children_Coordinator, InfluencedPort)
    LBTS = min of (lookahead in Lq that influences InfluencedPort of Children_Coordinator)
                                                    // sub-models influencer are defined according to coupling store in IC List.
```

Figure 6. G-DEVS Conservative-Local-Coordinator

```
Parent              // parent coordinator
tl                  // time of last event
tn                  // time of next event
G-DEVS              // associated model with total state (s, e)

When receive Imessage ('i', Simulator, t, - , - )
          tl = t - e
          tn = tl + ta(s)
          compute lookahead
          send Lookahead message ('l', Simulator, t, OutputPort, Lookahead Value) to parent
          send Dmessage ('d', Simulator, t, - , Value) to parent
When receive *message ('*', Simulator, t, - , - ) at time t
          if (t= tn)
          then      y = λ (s)
                    send Ymessage (y, t) to parent coordinator
                    s = δint (s)
                    tl = t
                    tn = tl + ta(s)
                    compute lookahead
                    send Lookahead message ('l', Simulator, t, OutputPort, Lookahead Value) to parent
                    send Dmessage ('d', Simulator, t, - , Value) to parent
          else error : bad synchronization
When receive Xmessage ('x', Simulator, t, InfluencedPort, x)
          if (tl<= t <= tn)
          then      e = t - tl
                    s  = δext (s, e, x)
                    tl = t
                    tn = tl + ta(s)
                    compute lookahead
                    send Lookahead message ('l', Simulator, t, OutputPort, Lookahead Value) to parent
                    send Dmessage ('d', Simulator, t, - , Value) to parent
          else error: bad synchronisation
When receive Nullmessage ('null', Simulator, t, InfluencedPort, - )
    e = t - tl
    compute lookahead
    send Lookahead message ('l', Simulator, t, OutputPort, Lookahead Value)
    send Dmessage ('d', Simulator, t, - , Value) to parent

get Lookahead(actual state)
          if (actual state has only internal transition)
          then      lookahead = tn
          else      if (actual state has (only external transition) OR (external & internal transition))
                    then lookahead = tn of next state with only internal transition

δext (s, e, x) // the functions of the G-DEVS models are recalled in the first section.
λ (s)
δint (s)
end G-DEVS-Simulator
```

Figure 7. G-DEVS Simulator

5.3. Behavioral Model

For the Behavioral specification, we used the Unified Modeling Language (UML) state diagram that is essentially a state diagram with standardized notation. The following tools have been used to make up the diagrams:

- Filled circle, denoting START.
- Hollow circle, denoting STOP.
- Rectangle, denoting state. Top of the rectangle contains a name of the state. Can contain a horizontal line in the middle, below which the activity is written that is done in that state
- Arrow, denoting transition. An expression can be written on top of the line, enclosed in brackets ([]) denoting that this expression must be true for the transition to take place
- Thick horizontal line with either x>1 lines entering and 1 line leaving or 1 line entering and x>1 lines leaving. These denote join/fork, respectively.

Overall Behavioral model

The overall state diagram (Figure 8) introduces the main objectives of the components, the objective of the LC is treating events and the DRC is designed to route the event exchanged between the distributed components of the global simulation.

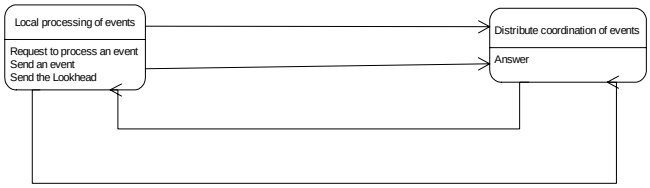

Figure 8. Overall Behavioral model

G-DEVS Conservative-Distributed-Root-Coordinator

Figure 9 diagram describes the treatments of the Conservative-Distributed-Root-Coordinator. We distinguish two main functions.

The first one is defined to receive message emitted by Local components.

The second is defined to answer to components calls to advance and treat their local messages.

In function of information, notably LBTS (Lower Bound on Time Stamp = min of Lookahead of influencers), from the different components, this component answers by granting calls or by delivering events time stamped earlier than the grant. In some cases, it delivers a null massage in order to avoid dead lock as described in [1].

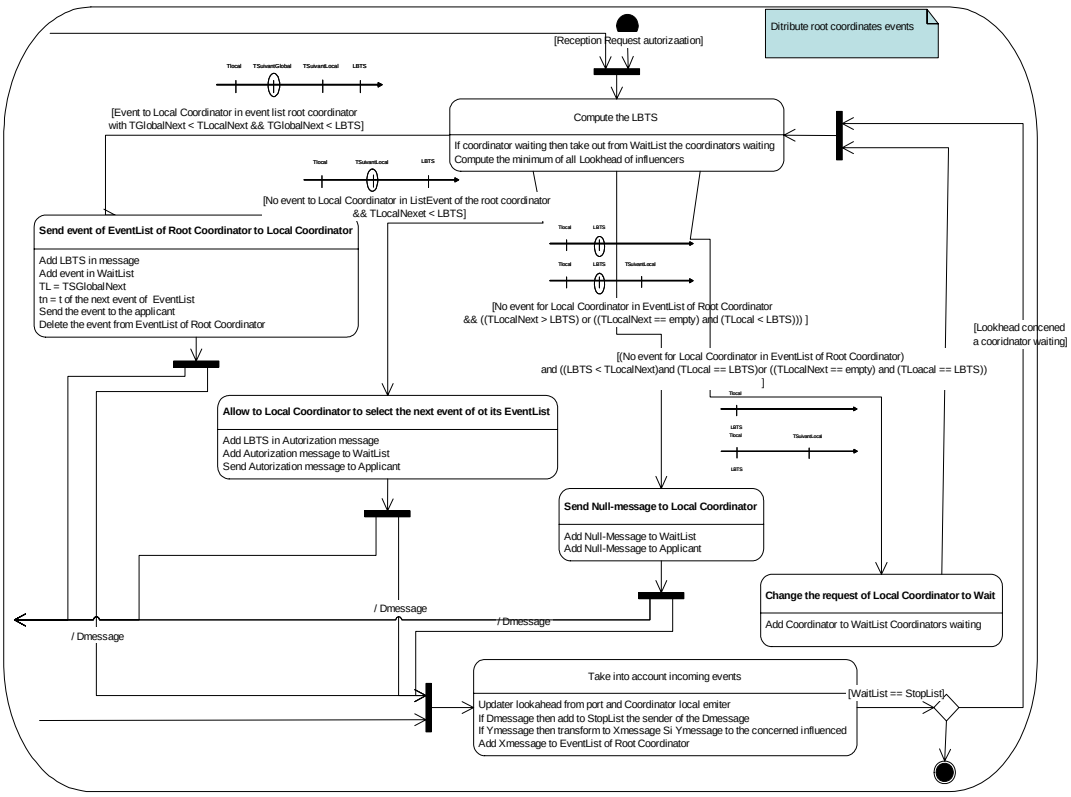

Figure 9. G-DEVS Conservative-Distributed-Root-Coordinator

G-DEVS ConservativeDistributedLocalCoordinator

After being registered at the DRC, the LC manages a local ordered event list containing local events to be treated. It calls the DRC to ask for a grant to treat its next message regarding its local time as described in Figure 10.

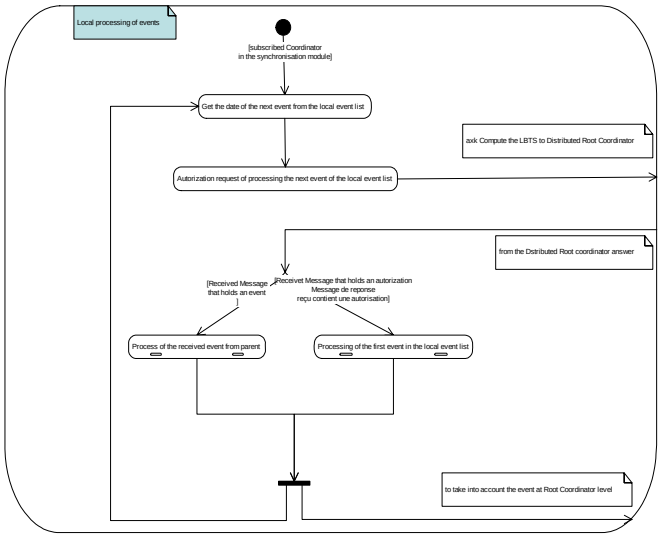

Figure 10. Conservative-Distributed-LocalCoordinator

When the LC receives the grant or the message from the DRC it transmits it to the local Simulator influenced and wait for acquittal message from this simulator.

When receiving an acquittal message and an eventual output messages from the simulator it sends to the DRC this eventual message. Finally, it starts again loop turn.

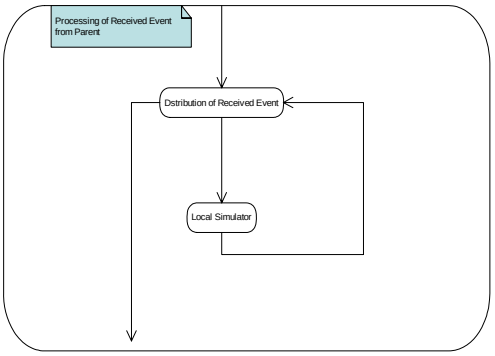

Figure 11. Conservative-Distributed-LocalCoordinator

We detail in Figure 11 the process of sending a message received from the DRC to the simulator concerned. Note that it can be also possible to receive, from the DRC, a null message to avoid deadlock situations.

Sending a local event is detailed in Figure 12; the event is suppressed from the event list and sent to child Simulator. This model describes the emission to the simulator concerned and the reception of the message of treatment acquirement corresponding.

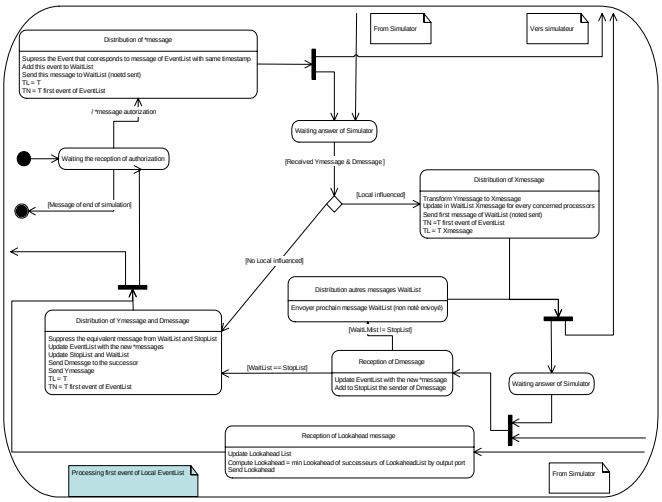

Figure 12. Local Event Processing

G-DEVS Conservative-Local-Simulator

Figure 13 model addresses the treatment of the different kind of events by a Simulator. Notably, the notion of null message has been added to classical events event took into account in the sequential version introduced by [12]. This message is used to avoid deadlock in conservative distributed time management.

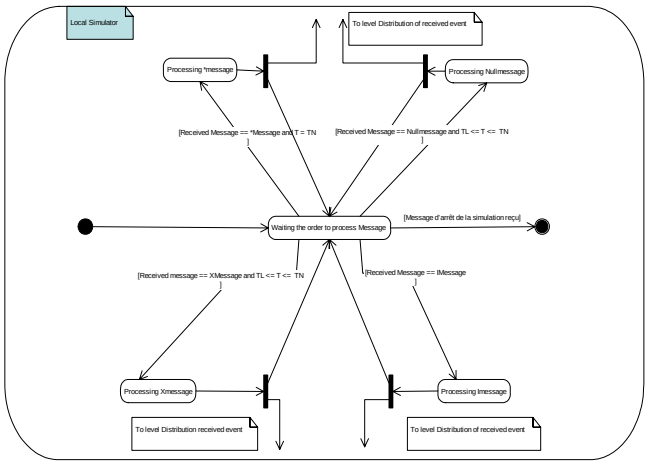

Figure 13. Local Simulator

6. HLA COMPLIANT DISTRIBUTED G-DEVS ENVIRONMENT

This specification supplies a conceptual framework of Distributed G-DEVS Simulation Structure. This abstract structure needs to be developed regarding to implementation requirement and context of the environment at a next step of the project management. The standard selected, in the conception step of the project development, for the project is HLA. Indeed, HLA specifies a mechanism for exchanging data between distributed components; in addition, it provides strong time management protocols. In addition, HLA RTI manages network communication protocols for distributed simulation, in order not to another charge in the project. Finally, others HLA compliant software can be added without recoding.

A first approach, presented in [11], of DEVS coordinators integration in an architecture respecting the HLA standard. They defined a local coupled model as a HLA federate whose coordinator of higher level will have responsibility to communicate with the federate Time Manager.

We choose to follow [11] mapping of LCS into HLA federates, but we do not use the Time Management federate and map the DRC, defined in this specification document, directly into the RTI because this specification of interface (RTI) proposes services which enclose those defined in our Distributed Root Coordinator. Indeed, the RTI manages Time Stamped messages and synchronizes the federates. The "global distributed" model (i.e. the federation) is constituted by creating a communication link between federates. Figure 14 illustrates this HLA mapping according to RTI Specification of [6].

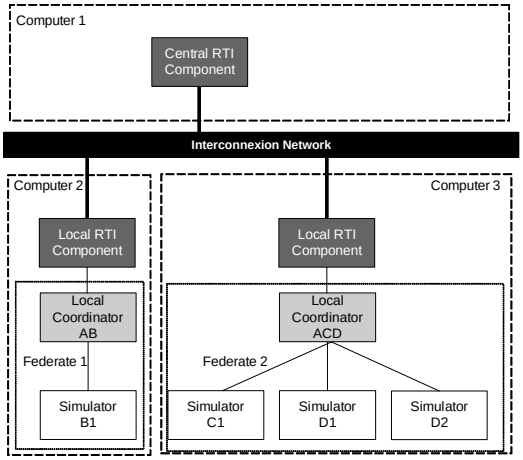

Figure 14. HLA-compliant simulation structure

We choose also to follow [11] proposal to integrate G-DEVS models coupling relations into HLA interactions.

Every G-DEVS coupled model and its associated LCS possess a SOM defining the information, which they are able to supply in a distributed simulation respecting HLA. The common data of the SOM of various federate of the local models allow producing the FOM of the federation G-DEVS / HLA (Cf tables Figure 15, Figure 16).

More precisely, a model G-DEVS possessing an output port "publishes" on a Class of Interaction (from its SOM) defining the relation of coupling of output (IC and/or EOC) by *publishInteractionClass*().

On the other side, G-DEVS model possessing an influenced port of input "subscribes" to the Class of Interaction (IC and/or EIC) published by the port which influences it by using *subscribeInteractionClass*(). The FOM will contain consequently all the coupling relations between the G-DEVS models of the distributed coupled model, this information will be shared in the form of classes of interaction.

Interaction Class Structure Table		
HLAinteractionRoot (N)	CouplingRelation (PS)	ExternalInputCoupling (S)
		ExternalOutputCoupling (P)
		InternalCoupling (PS)

Figure 15. Interaction Class Structure Table

The parameters of the interactions between federated are defined "TimeStamped Order" (Cf. Figure 16) to respect the principle of causality. These interactions are thus emitted with a time stamp associated to the local logical time of federate publishing and are stored in a scheduler of the LRC before being delivered to the influenced when this last one will temporarily be capable of treating this message.

Parameter Table					
Interaction	**Parameter**	**Datatype**	**Available Dimensions**	**Transportation**	**Order**
CouplingRelation.ExternalInputCoupling	Message Type	HLAASCIIchar	TypeMessage	HLAreliable	TimeStamp
	Transmitter	HLAASCIIstring	NA		
	Event time stamp	HLATimeType	NA		
	Concerned Port	HLAASCIIstring	NA		
	Event dimension	HLAboolean			
	Event Value	HLAopaqueData	NA		
CouplingRelation.ExternalOutputCoupling	Message Type	HLAASCIIchar	TypeMessage	HLAreliable	TimeStamp
	Addressee	HLAASCIIstring	NA		
	Event time stamp	HLATimeType	NA		
	Concerned Port	HLAASCIIstring	NA		
	Event dimension	HLAboolean			
	Event Value	HLAopaqueData	NA		
CouplingRelation.InternalCoupling	Message Type	HLAASCIIchar	TypeMessage	HLAreliable	TimeStamp
	Addressee or Transmitter	HLAASCIIstring	NA		
	Event time stamp	HLATimeType	NA		
	Concerned Port	HLAASCIIstring	NA		
	Event dimension	HLAboolean			
	Event Value	HLAopaqueData	NA		

Figure 16. Parameter table

We can also produce, in the FOM, a class of object which defines elements shared by the local simulations. These objects allow, for example, following the evolution of the values of state variables of a model. In that case, federate publishing to a follow-up of these variables will use the function *publishObjectClassAttributes*(). For example, PIOVRA Visualization/Interface Management Federate (VMF) will be interesting in subscribing to the follow-up of these values, in that case the function *subscribeObjectClassAttributes*() will be called. The G-DEVS model integrated in this HLA federate will also store its actual phase and its states variables (used with the elapsed time to define the total state of a G-DEVS model) in a HLA Object in order to inform eventually other federation members on that federate status.

We can notice that in the implementation, thanks to the HLA time management, we do not use the Null-message specified in the conceptual algorithms of this document, because the RTI have a global view of the simulation, so it manages deadlock by granting minimum time stamped call from a federate.

7. CONCLUSION

This paper presented the specification of a new distributed G-DEVS environment. Furthermore it detailed the conception of this environment by conforming to the HLA Standard. This Work is involved in the Piovra project and based on the technical report [10].

REFERENCES

[1] 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).

[2] Escudé B. PhD Thesis : ``Modélisation et simulation à événements discrets de systèmes hybrides", Université de Droit, d'Economie et des Sciences d'Aix-Marseille III, IUSPIM-DIAM, Marseille, 2000.

[3] Fujimoto R. M. 2000 Parallel and Distributed Simulation System. Wiley Interscience, NY.

[4] Giambiasi N., B. Escude and S. Ghosh, “G-DEVS A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems”, Transactions of the SC S International, 17(3):120-134, 2000.

[5] Glinsky E., G. A. Wainer: “DEVStone: a Benchmarking Technique for Studying Performance of DEVS Modeling and Simulation Environments”. DS-RT 2005: 265-272, Montreal CA, 2005.

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

[7] Kim K., W. Kang., B. Sagong., H. Seo. 2000. “Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One” 33rd Annual Simulation Symposium, ASS, (April 16-22, Washington, D.C).

[8] Zacharewicz G., N. Giambiasi, C. Frydman, “Lookahead Computation in G-DEVS/HLA Environment”, in: Simulation News Europe Journal (SNE) special issue 1 “Parallel and Distributed Simulation Methods and Environments”, vol. 16, n° 2, pp. 15 - 24, September 2006. ISSN, 0929-2268.

[9] Zacharewicz G., M. E.-A. Hamri, “Flattening G-DEVS / HLA structure for Distributed Simulation of Workflows”, in: AIS-CMS International modeling and simulation multiconference, pp. 11-16, Buenos Aires - Argentina, February 8-10 2007. ISBN 978-2-9520712-6-0

[10] Zacharewicz G., Massei M., Bruzzone A.G. 2006 "Definition of G-DEVS/HLA framework, tailored for PIOVRA", PIOVRA EDA Tech Report., Genoa, Italy

[11] Zeigler, B. P., G. Ball., H. J. Cho., J. S. Lee. and H. Sarjoughian. 1999. “Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions”. Spring Simulation Interoperability Workshop (SIW), (Orlando, FL, March 14-19), 99S-SIW-065.

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