On the Speedup/Delay Trade-Off in Distributed Simulations

Alessandra Pieroni *alessandra.pieroni@uniroma2.it*
Enterprise Engineering Department
University of Rome TorVergata
Rome, Italy

Giuseppe Iazeolla *giuseppe.iazeolla@uniroma2.it*
Engineering Department
University of Rome TorVergata
Rome, Italy

Abstract

Assume a local simulator (LS) of a given system is available and we wish to turn it into a distributed simulator (DS). In the DS case, the LS is partitioned into segments called federates, each federate being run by a separate host. Before implementing the DS (i.e., at design-time) we wonder: will the DS execution time be shorter than LS one? In some cases the DS may run slower than the equivalent LS. To answer this question we are to consider that the execution time of a distributed simulation system depends on 3 interacting factors: 1) the speedup (or run-time gain) resulting from the partitioning of the local simulator into federates. 2) The network delays in the federate synchronization messages exchange. 3) The network delays inthe federatedata messages exchange. The combination of such factors makes very hard predicting the benefits of the LS-to-DS transformation. In this paper, a LS/DS decision procedure to support the LS/DS decision process at design-time. The procedure is guided by a performance model of the DS. The use of the High Level Architecture (HLA) distributed simulation standard is assumed.

Keywords:distributed simulation,parallel speedup, computer networks delay.

1. INTRODUCTION

A simulation model can be seen as consisting of a set of sub-models. In local simulation (LS), a single model exists that simulates the entire system and is run by a single host. In distributed simulation (DS), various sub-models (called federates) simulate distinct parts of the system and are run by separated hosts connected via a LAN, MAN or WAN computer network or a composition thereof.
Predicting at *design-time* the convenience of implementing the DS version of the LS can be of interest. Indeed, the development of a DS system is a complex and expensive task, since of the cost of achieving the necessary know-how of the distributed simulation standard [1], the cost of the extra-lines of code to develop for each federate [2], the cost of the hosts, the computer networks, and the number of design alternatives to face (in terms of simulator partitioning, host capabilities, network potentialities and so on).

This paper introduces a method to support the evaluation of the DS convenience before implementation. The method investigates the effects of three interacting factors:

1) The speedup (or run-time gain) resulting from partitioning the local simulator into federates, spread across various hosts that operate in parallel; 2) the synch-communication overhead due to network delays in the exchange of synchronization messages among federates; 3) the data-communication overhead due to network delays in the exchange of data messages among federates.

The two communication overheads lower down the run-time gain obtained with the speedup.

A LS/DS decision procedure is proposed to choose (at design-time) whetherto remain on the LS version of the simulator orcarry out the implementation of its DS version. The procedure is guided by a performance model (PM) of the DS. The model can be used to perform what-if analysis and sensitivity analysis to observe how changing one or the other factor may affect the DS execution time. The PM assumes the DS is based on the HLA protocol standard and middleware [3].

The paper is organized as follows: Sect.2 presents the problem statement. Sect.3 illustrates the PM. Sect.4 illustrates the PM implementation in the OMNet++ simulation language and its use in the LS/DS decision procedure. Sect.5 presents the paper contribution with respect to existing literature and finally Sect.6 gives concluding remarks.

2. PROBLEM STATEMENT

Assume a local simulator (LS) of a given system Σ is available, and that we wish to turn it into a distributed simulator (DS).

In the DS case, the LS is partitioned into segments called federates, each federate being run by a separate host. Fig.1 shows the two federate case, with N_S denoting the network for the exchange of synch messages and N_D the one for data messages.

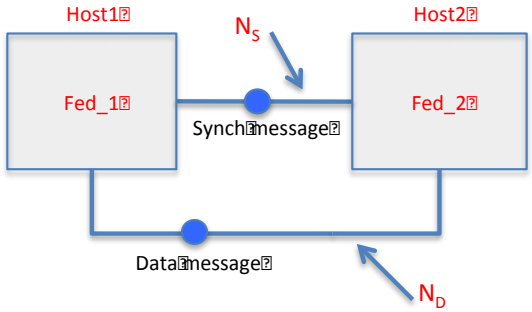

FIGURE 1:DS system with two federates.

Before implementing the DS (i.e., at design-time) we wonder: will the DS execution time be shorter than LS one? In some cases the DS may run slower than the equivalent LS. To answer this question a Performance Model (PM) of ak-federate system is introduced in Sect.3.

The following terminology will be used throughout the paper:

- Σ = System to be simulated
- LS(Σ) = Local Simulator of Σ
- T_{LS} = LS execution time
- DS(Σ) = Distributed Simulator of Σ
- T_{DS} = DS execution time
- PM(DS(Σ)) = Performance Model of DS(Σ) to predict the execution time T_{DS}.

The question is: when does DS(Σ) run faster than LS(Σ)? In other words, when does $T_{DS} < T_{LS}$ hold?

There are 3 conflicting factors that determine the T_{DS} value:

- *Speedup*: the run-time gain resulting from partitioning LS into federates spread across many hoststhat operate in parallel.

Thanks to the speedup one may obtain $T_{DS} < T_{LS}$, the speedup being defined by $S = T_{LS} / T_{DS}$, a positive speedup meaning $S > 1$. Let us call S the **no-delay speedup**, for reasons that will be soon clear.

- *Synch communication overhead*: all DS simulations must incorporate techniques to coordinate the execution of federates across the many hosts by synchronization messages. Such messages travel along a synch network N_S (that may be a LAN, a MAN or a WAN, or a composition thereof) whose delay ΔN_S may yield a $T'_{DS} > T_{DS}$ thus reducing the no-delay speedup S to a **synch-delay speedup** $S' < S$ with $S' = T_{LS} / T'_{DS}$.
- *Data communication overhead*: the federates also need to exchange data-packets by way of data messages. Such messages travel along a data network N_D (that may or may not coincide with N_S), whose delay ΔN_D may yield a $T''_{DS} > T'_{DS}$ thus reducing the synch-delay speedup S' to a **synch&data-delay speedup** $S'' < S'$ with $S'' = T_{LS} / T''_{DS}$.

The question above then becomes:
When does T''_{DS} turn out to be lower than T_{LS} ($T''_{DS} < T_{LS}$), thus still yielding a positive speedup $S'' > 1$?
In other words, when can the no-delay speedup win over the synchronization and data communication overheads?
Next section tries to answer such a question.

2.1 The speedup/communication overhead trade-off

As with most parallel computations, to obtain a positive speedup the portion of LS that can be parallelized must be large relative to the portion that is inherently serial. Let us denote by S(N) the maximum speedup that can be achieved using N processors, and by Q the fraction of computation that is inherently serial. According to Amdahl's law [4,5] even with an arbitrarily large number of processors ($N \to \infty$), S(N) can be no larger than the inverse of the inherently serial portion Q of LS.

$$S(N) = \frac{1}{Q + \frac{1 - Q}{N}} \quad (1)$$

Thus, one requirement for the DS code to achieve positive speedups is that the fraction Q should be small.

An appropriate partitioning of LS into a set of federates should then be found at design-time that improves S while maintaining the synch and data overheads low. In other words, a partitioning that yields a high computation-to-communication ratio (i.e., a large amount of computation between communications).

On this basis, an LS/DS decision procedure can be foreseen (Fig.2) to decide whether to remain on the LS version of the simulation system or carry out the implementation of its DS version.

In other words, assume an LS(Σ) has been developed and that its T_{LS} is not satisfactory. A search for an appropriate partitioning of LS(Σ) into federates and for an appropriate choice of the N_S and N_D networks has to be performed by the iterative use of the PM(DS(Σ)), to obtain a $T''_{DS} < T_{LS}$.

At each iteration, if the T''_{DS} predicted by the PM is sufficiently lower than T_{LS}, the decision to implement the DS(Σ) can be taken.

Otherwise, one may either try a new tentative partitioning or try alternative networks N_S and N_D of improved capabilities. In case no partitioning nor network improvements can be found, one may decide not to implement the DS(Σ).

En example use of the PM in the LS/DS decision procedure is illustrated in Sect.4. The PM cannot be evaluated by analytic methods and thus its evaluation is simulation-based. The coding of the PM is done in the OMNet++ simulation language [6] and an example coding is provided in Sect.4.

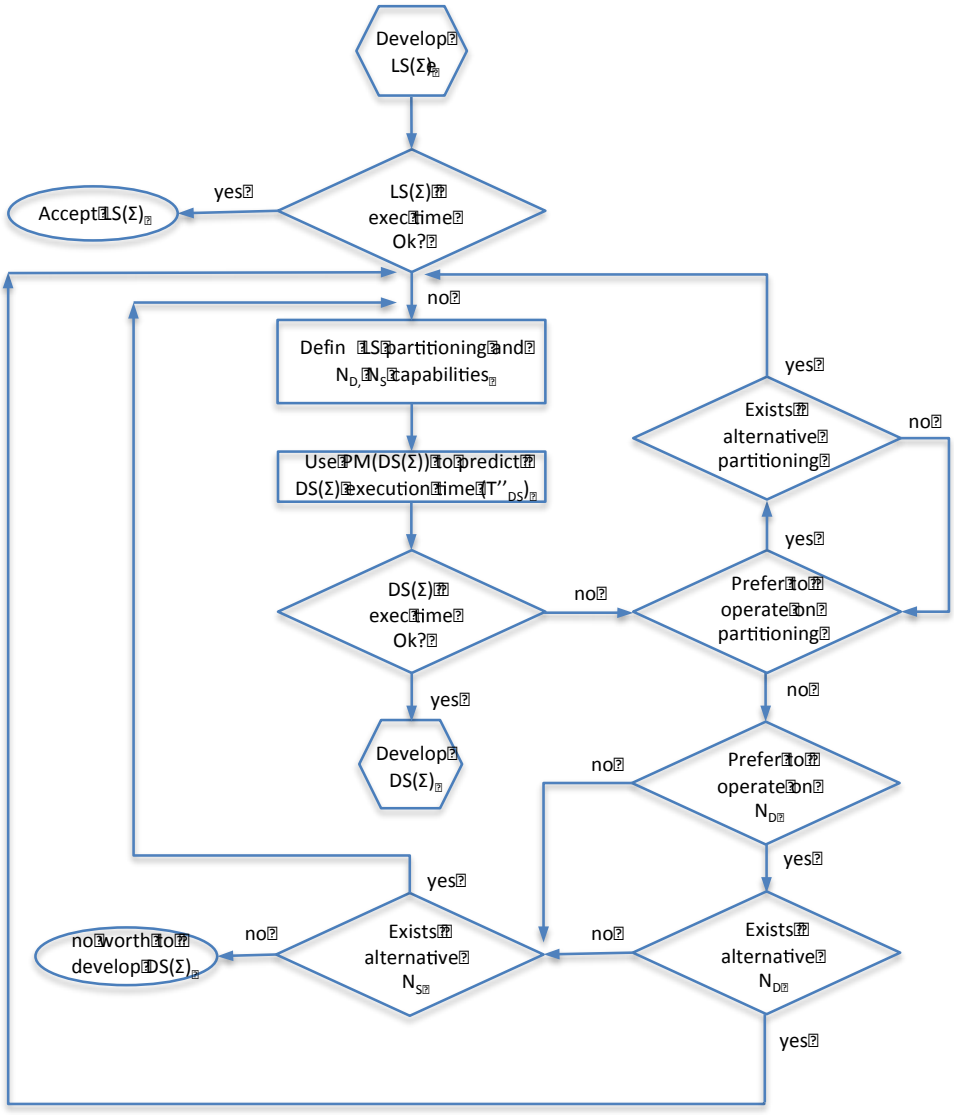

FIGURE 2: The LS/DS decision procedure

3. RELATED WORK

A number of existing contributions can be found in literature that address the prediction of execution times of simulation systems, see e.g., [5, 15, 16, 17, 18, 19, 20].

Contributions in [15, 16, 18, 19, 20] however deal with parallel simulation case rather than distributed case, as is this paper. Parallel simulations are run by a set of processors that are tightly connected by use of dedicated networks rather than by a computer network, as is this paper case (a set of processors connected by a computer network, such as a LAN, a MAN or a WAN, or a composition thereof). Moreover, parallel simulations are coordinated by an ad hoc network-operating system rather than by DS middlewares (e.g. HLA) as is this paper case.

In other words, our paper cannot take advantage from results of the parallel simulation literature. On the other hand, looking at the distributed simulation literature, the only work that,to our knowledge, one can refer to is [17], which however only deals with the performance of DS shared data-access algorithms, a topic that is not of interest to our paper, which instead is interested to the evaluation of the whole DS execution time (T_{DS}).

There are two essentially different ways for evaluating theT_{DS}.One way is to base the analysis on the execution of a DS run. The other way (this paper way) is to use the LS version of the simulator to derive parameters for predicting the T_{DS}. While the first method is potentially more accurate, its main disadvantage is that it requires the existence of the DS program, hence it cannot really predict T_{DS}, and it can only be used to evaluate the $T_{DS}$of a given DS implementation.

In a previous work [12], the trace information generated during the LS run has been obtained and it will be used in this paper now to derive parameters to give to the performance model (PM) of the DSfor the T_{DS} prediction.

The PM, this paper now introduces, is able to separately investigate the effects of the model partitioning, and also investigate separately the effects of the N_S delay and of the N_D delay. Besides being important for the LS/DS decision procedure, the knowledge of the effects of the two communication overheads is of importance to evaluate the representativeness of the DS(Σ) at design-time.

Indeed, depending on the nature of system Σ, there are situations in which the data and synch message delays are not critical and thus a communication network of any capability can be used. On other situations, instead, the system Σ can be of such a nature that the synch and data delays become very critical for the representativeness of the system simulator. In other words, the DS(Σ) looses the capability of realistically representing the original Σ in case the N_S and N_D networks are not sufficiently well performing.

4. THE PERFORMANCE MODEL OF DS(Σ)

It is assumed the reader is familiar with the structure of an HLA federation, based on the so-called *Run Time Infrastructure* (RTI) [7]. The RTI is the software that allows the federates to execute together. In Fig.3 the interface between the RTI and the federates is illustrated [8]. The federates do not talk to each other directly. They are instead connected to the RTI and communicate with each other using services provided by the RTI. The RTI offers to each federate an interface called *RTI Ambassador*.

Each federate on the other hand presents an interface called Federate Ambassador to the RTI.

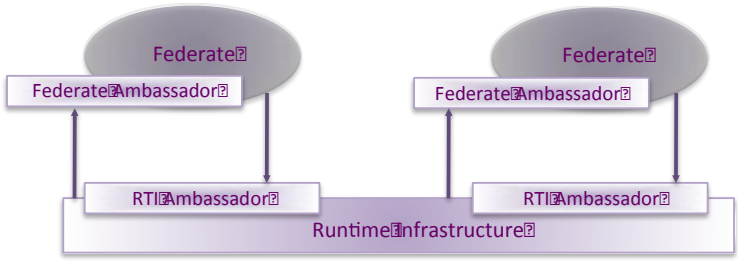

FIGURE 3: HLA federation structure

In the following we shall denote by:

- LEX the local execution internal to a federate, in other words, the standard simulation operations such as event processing, event routines, scheduling of local events, etc.

• HLAR the execution internal to a federate of an RTI service, e.g., an invocation of a time advance request.
• HLAF-Ex the execution internal to a federate of a service request coming from the Federate Ambassador.

Assume we deal a federation consisting of k federates. The PM(DS(Σ))model will consist of onesub-model for each of the k federates, and oftwo network models,one for the N_S and one for the N_D network.

The single federate sub-model is illustrated in Sect.3.1, and consists of a non-conventional EQN (Extended Queueing Network) model, that we denote *NC_EQN*, that includes both conventional EQN nodes and a number of *AND/OR* logic gates necessary to bring-in the logic of the HLA standard. The network model, instead, is a conventional EQN model andillustrated in Sect.3.2, to be used (with appropriate parameters) for both N_S and N_D.

3.1 The federation performance model

To answer the Fig.2 question "Exists alternative partitioning?" we shall assume that the LS is partitioned into a k-federates DS and shall evaluate the PM of such a partitioning.

The *NC_EQN* model of a k-federates DS is shown in Fig.4, where the details of the PM of only one Fededate (Fed_i)are illustrated. One may partition the LS code into the k-portions of the DS code in various ways. As we shall better see in Sect.4, the effect of the partitioningchoice is reflected in the valuegiven to parameter p_{SYNC} at model parameterization time.

In Fig.4 the interactionsare shown between Fed_i and all remaining federates (in the *publish/subscribe* RTI assumption). The set of all remaining federates is denoted by using the bold **Fed_x** notation. Therefore, the ***x****i* (or *i****x***) notation will be used in the subscript of various components in Fig.4.For example, gate AND_{xi} relates to the synch-messages exchanged between **Fed_x** and Fed_i. Consequently, we are to figure out number k-1 such *AND* gates in the illustration. The same can be said for all other*AND* and *OR* gates with a bold**x** in the subscript.

As visible in Fig.4, each Fed_i sends (receives) messages to (from)**Fed_x**through the N_S and N_D networks. The entire federation PM will thus consist of a set of Fed_i sub-models (as in Fig.4) that interact between themselves through the N_S and N_D nodes as in Fig.5, that shows how messages from various federates are enqueued in front of N_S or N_D to be served, i.e. forwarded to the destination federates.

The Fed_i model in Fig.4 includes:

- a time-consuming node (*Fed_i Host CPU*), that synthetically represents the host that runs the federate. Such a node service-time parameters vary with the serviced job class (C_i or C_{RC} see later).
- a set of non-time consuming nodes, namely:
  - *AND* nodes that perform AND-logic operations.
  - *OR* nodes that perform OR-logic operations.
  - *SPLIT* nodes, that split an incoming job class into two or more outgoing classes.
  - *Classifier* nodes that, basing on the class of the input job, forward the job in one or the other direction.
  - *Router* nodes that perform probabilistic routing of the incoming jobs.
  - a*Merge* node that merges two job classes.

The computation performed by the federation starts by launching the RTI interface and by initializing the HLA components local to each federate. Such initial computations are performed

only once and do not substantially affect the federation execution time and thus are omitted from the modeling. They are synthetically represented by the INIT job on the top of Fig.4.

The INIT job enters the *Split*$_0$ node and yields a main thread for each federate belonging to the Federation. The main thread C_i for Fed_i and its follow-on is detailed in Fig.4.

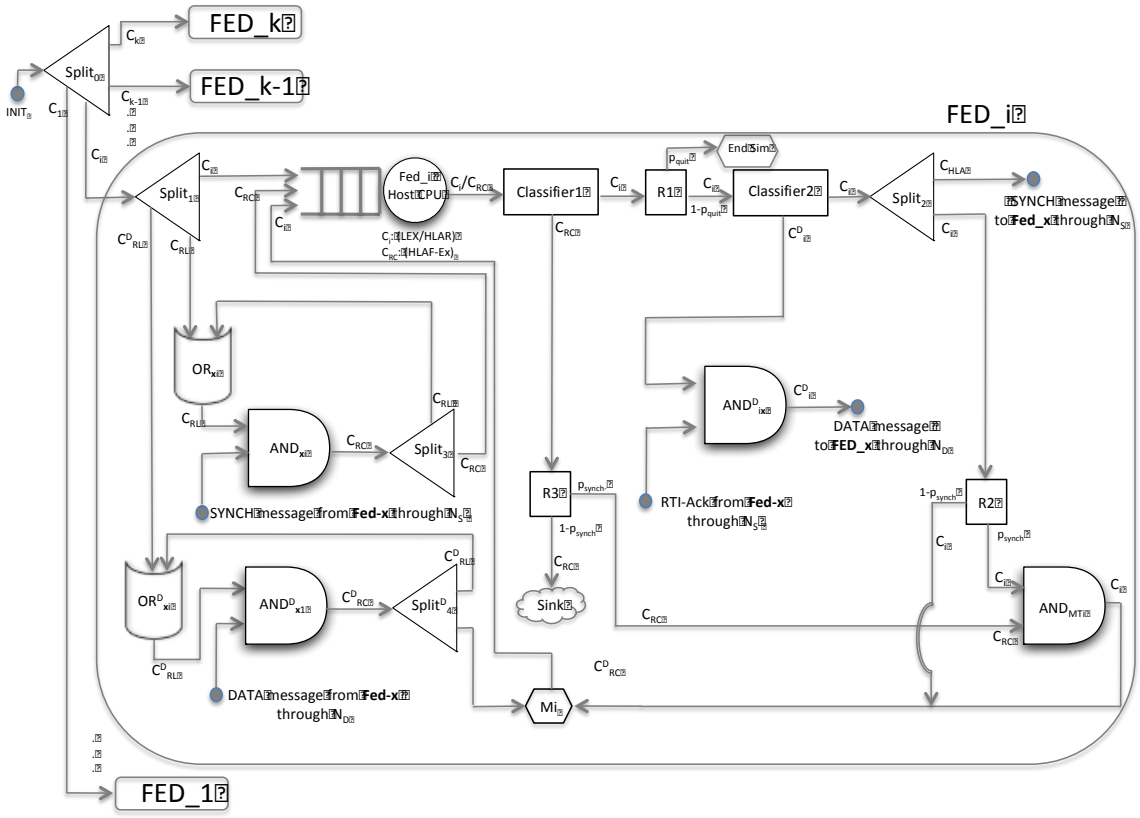

FIGURE 4: View of the federation PM with details of the i-th federate.

It is assumed that the *conservative-time management* HLA option is used, in other words, no federate advances logical time except when it can be guaranteed not to receive any events in its past. If we also assume *zero lookahead* (actually HLA does not accept a null value for lookahead and thus a very small value is given to this parameter) there is guarantee that federates do not receive events in the past and thus that they are fully synchronized.

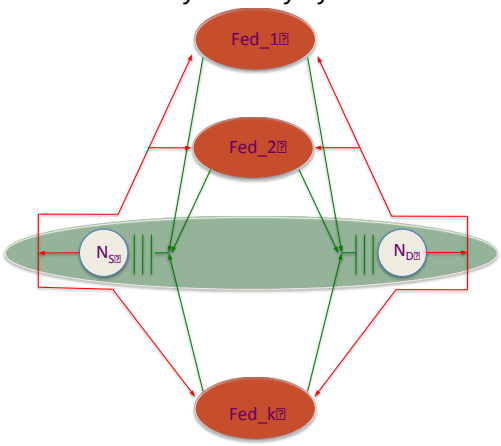

FIGURE 5: View of the federation PM including the communication networks.

For such choices, the federates will not process events in parallel and parallelism will only be found when federates include intrinsically parallel portions of LS. If this holds, a positive speedup will be obtained when transforming the LS into its DS version.

The computation performed by Fed_i is carried out by jobs of various classes that circulate in its PM, namely:

- Class C_i jobs
- Class C^D_i jobs
- Class C_{HLA} jobs
- Class C_{RL} jobs
- Class C^D_{RL} jobs
- Class C_{RC} jobs
- Class C^D_{RC} jobs

The only jobs that consume CPU execution time are C_i and C_{RC}.

The class C_i job[1]simulates the so-called federate*main thread*[8], performing LEX and HLAR computations.

The class C_{RC} job[1] simulates the so-called federate*RTI callback* [8],performing HLAR-Ex computations.

The class C^D_i job is a job derived from C_i and holding the data payload to be forwarded to **Fed_x**through network N_D, when the RTI-Ack arrives from **Fed_x**(see the AND^D_{ix} node). A class C_{HLA}job is a job derived from C_i and holding the synch-message to be forwarded to **Fed_x**through network N_S. A class C_{RL} job represents the so-called federate*request listener thread*, waiting for synch-messages from **Fed_x**(see the AND_{xi} node). A class C^D_{RL} job is the federate request listener thread, waiting for data messages from **Fed_x**HLAF-Ex computations. A class C^D_{RC} job is the federate request callback thread holding the data payload coming from **Fed_x**and to be used by the C_i job class.

The main thread C_i enters the $Split_1$ node and yields three outgoing jobs: C_i itself again, C_{RL} and C^D_{RL}. The job of class C_i enters the *CPU* processing queue from $Split_1$and circulates in the model (in a way that we shall soon illustrate), so iteratively re-entering the *CPU* processing queue coming from the AND_{MTi} node. The job of class C_{RL}, instead, enters the OR_{xi} node and from here the AND_{xi} and waits for a synch-message from **Fed_x**to generate a C_{RC} job, which through the $Split_3$ produces both a new C_{RL} job (that waits for future synch messages) and the C_{RC} itself again that enters the *CPU* processing queue.The same logic applies to the C^D_{RL} job coming from $Split_1$, which enters the $OR^D_{xi}$and the AND^D_{xi}nodes waiting for a data-message from **Fed_x**. The C^D_{RC} job outgoing $Split_4$does not enter the *CPU* processing queue directly but merges itself with the C_i circulating main thread through the merge node M_i. As said above, the C_i job entering the *CPU* performs LEX and HLAR computations, while the C_{RC} job performs HLAR-Ex computations.

The job leaving the *CPU* can be a C_{RC} or a C_i job.

- In case the job leaving the *CPU* is a C_{RC} job, the $Classifier_1$node forwards it to the router R_3, which sends the job to the AND_{MTi} node in case the*synchronous HLA service*[2] is invocated [8].

[1]The service time *parameters* for such a job class (distribution, mean $E(t_{CPU})$ and variance) can be obtained [11] basing on the model of the software run by the Fed_i CPU, and on the CPU capacity.

[2]In other words, when the federate needs to wait for a RTI callback (C_{RC}), in the case of invocation of a *Time Advance Request Service*, or of a *Next Message Request Available* service.

Otherwise, the C_{RC} job has no effects and is absorbed by the sink node. If directed to the AND_{MTi} node, the C_{RC} job gives consensus to the circulation of the main thread C_i, which thus re-enters the *CPU* processing queue.

- In case instead, the job leaving the *CPU* is a C_i job, the $Classifier_1$directs it to the R_1 router, which sends the job to $Classifier_2$if the simulation is not ended (1-p_{QUIT}). Here, if C_i contains a data-message, a C^D_i job is produced which enters the AND^D_{ix}node, and waits for the RTI-Ack from **Fed_x**in order to be forwarded to **Fed_x** through network N_D. If instead, the outcome from Classifier2 is a no-data message C_i this enters the $Split_2$ node and yields a C_{HLA} job (holding a synch-message to be forwarded to **Fed_x**through network N_S) and again a circulating main thread C_i, which (in case a synchronous HLA service is invocated(p_{SYNC})) reaches the aforementioned AND_{MTi} node to iterate the main thread circulation. In case, instead, of *no-synchronous HLA service*[3] (1- p_{SYNC}), the C_i job does not need the AND_{MTi} consensus to iterate the main thread circulation, and returns directly to the *CPU* processing queue.

In summary, synchronization and data messages that Fed_i exchanges with other federates**Fed_x** are enqueued in front of the Fed_i Host CPU to be processed.

Considered that in the *publish/subscribe* assumption Fed_i interacts with all remaining k-1 federates, the message flow arriving into the queue of the Fed_i Host CPU scales-up with the dimension k of the federation.

Another element that may increase the message flow into the CPU queue is the use of lookahead. Indeed, the frequency of the synchronization messages exchanged between federates per wall clock time-unit may be affected by the value of the lookahead parameter set by the user.

Such a parameter, however, assumes significant values only in some kind of distributed simulation models. So, in many cases, the federate PM needs not to model the rise of synch messages due to lookahead. This is the case of the Fig.4 model, which however can be easily extended to include lookahead synch messages generators, if needed.

Let us conclude this Section by pointing out that in building the Fed_i model we did not make any mention of the simulated system Σ. This is since the federate model we introduce in the paper is independent from Σ, i.e. it is valid for any Σ. In other words, the paper model can be used for any HLA-based simulation. Only its parameters may depend on Σ, as better seen in Sect.4.

3.2 The network performance model

A further model is necessary to answer the second Fig.2 question of the LS/DS decision procedure: “Exists alternative N_S (or N_D)?”. The needed model is the model of the computer network connecting the federation hosts. By use of such a model, the “Exists alternative N_S (or N_D)?”question can be answered by making what-if and sensitivity analysis of the various network components (LANs, GWs, WAN, etc.) of both N_S and N_D.

As said above,the entire federation PM consists of a set of Fed_i sub-models (as in Fig.4) and of the N_S and N_D network models for communication between federates (as in Fig.5). Such networks are used in common by all federates and thus synch and data-messages will enqueued in front of N_S and N_D as shown in Fig.5. Network N_S will thus introduce a ΔN_Sdelay for the synch-messages and similarly the N_Da ΔN_D delay for the data-messages.

[3]In other words, in case of invocation of *Send Interaction* service.

In other words, when Fed_i sends a synch-message to **Fed_x** through N_S, the message reaches **Fed_x** after a time that can be calculated by evaluating the ΔN_Sintroduced by N_S.

The evaluation of ΔN_S requires knowledge of the detailed model of N_S (and similarly for N_D). The generic network architecture we shall assume is illustrated in Fig.6 and consists of:

a) A set of LANs (LAN_1, …, LAN_k) where LAN_i is the LAN to which the Fed_i host is connected.
b) A set of GATEWAYs (GW_1, …, GW_k) where GW_i is the gateway that connects LAN_ito the WAN.
c) The WAN communication backbone.

Fig.7 gives the EQN performance model of such a network assuming the TCP/IP protocol is used.

The interaction between the Fed_i Host and the **Fed_x** Hosts is based on message exchanges carried out by packet flows over the various components of the network with the WAN being a X.25 packet switching network. The packet flow involves several technologies: The LAN_1 through LAN_k technologies (e.g.: Ethernet, Token Ring, etc.) the Gateways technology and the X.25 WAN technology.

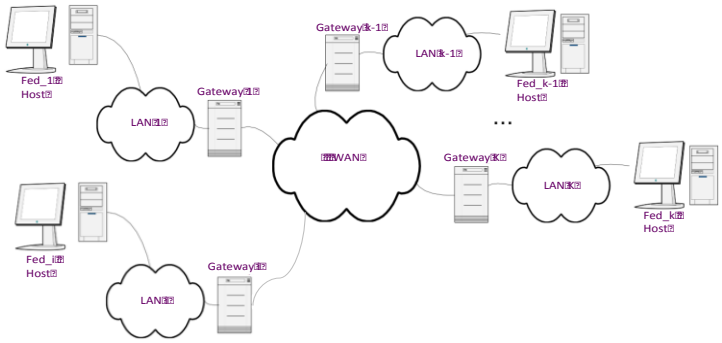

FIGURE 6: View of the network architecture.

The communications between Fed_i Host and **Fed_x** Hosts are based on three basic mechanisms (m1, m2, m3):

(m1) protocol conversion, from the transport level protocol TCP, to the network level protocol IP, to the data-link level and physical level protocols (and vice versa), in either direction from Fed_i Host to**Fed_x** Hosts, with the IP to X.25 protocol conversion (and vice versa) at the gateway level,
(m2) packet fragmentation and re-assembly at many protocol conversion interfaces,
(m3) window-type flow control procedure operated at transport level by protocol TCP for a fixed window size of value C (for the sake of simplicity no varying window sizes are considered, nor the use of congestion-avoidance algorithms).

In the Fed_i-to-**Fed_x** flow, illustrated in Fig.7, the packets are originated by the Fed_i Host application level in TCP format and then translated into IP format by the Fed_i Host network level to enter LAN_i. From LAN_i they exit in LLC/MAC802.5 format to enter the GW_ifragmentation section (FRAG) that fragments them into X.25 format to be transferred by the transfer section (TRANS) to the WAN. Vice versa for the **GW_x**, where X.25 packets are re-assembled by its reassembly section (REA) into LLC/MAC802.3 format to be forwarded to the LAN_x by the Transfer section (TRANS).

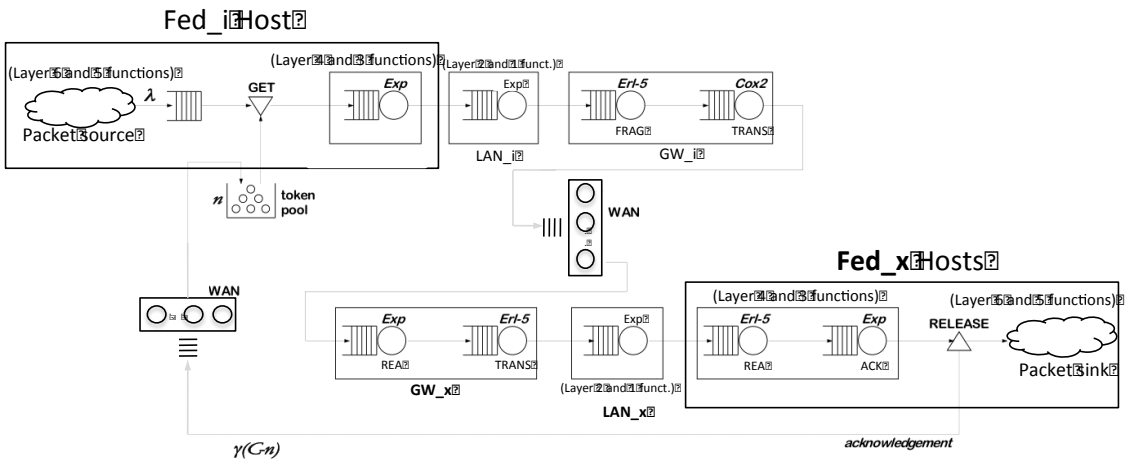

FIGURE 7: View of the network performance model.

LAN_x transfers such frames to **Fed_x** Hosts, which in turns re-assembles them into IP packets and then into TCP frames, in the re-assembly section (REA). The received TCP frames are finally passed to the application level and are acknowledged by the sending of an ACK packet back to Fed_i Host, by the acknowledger section (ACK). The token pool [9] is introduced to represent the window-type flow control procedure implemented by the TCP between the source and the sink (see later).

In summary, the considered network consists of various subsystems each of different complexity, each sub-system in turn consisting of various subsystems of various complexities. Each LAN, for example, is in itself a complex network (not illustrated here), and similarly the WAN.

Producing a model of the network with all details of each LAN, all details of the WAN, etc., could yield so many components and details to make the model very difficult to handle, and its evaluation very time-consuming.

In order to obtain a tractable model, a hierarchical hybrid approach [11] can be foreseen.
To this scope, three abstraction levels are introduced:

Level-1 abstraction: At this level the separable sub-systems are identified according to decomposability theory [10], and studied in isolation. Assume the LANsare separable sub-systems. In this case they can be preliminarily studied separately from the rest of the network, then evaluated to obtain their end-to-end delay, and finally substituted in the network model by equivalent service centers whose service timesare the end-to-end delays obtained above. If separation is possible, each LAN model (that normally consists of a very large number of service centers) is collapsed into a single equivalent center. The decomposability conditions for the LANs, can be verified[4], and are respected in the considered model.
In conclusion, at this level sub-systems LAN_1, through LAN_kare separately evaluated to obtain their equivalent end-to-end delay and areeach replaced by a single equivalent center, as illustrated in Fig.7. The evaluation gives the distributions (e.g.: exponential in the Fig.7 case) of the LAN equivalent service time and its *parameters* (mean, variance, etc.), calculated basing on the model of the software run by the LAN and the capacity of the hardware [11].

[4] Generally speaking, the decomposability condition holds when the events that are injected from the external systems into the separable sub-system (i.e. from Fed_i Host into LAN_i and from **GW_x** into **LAN_x**) are very rare with respect to the events that take place internally to the sub-system. This can be also empirically verified by comparing the average service rates of the external systems with the ones internal to the LANs. In the specific case, the formers are orders of magnitude smaller than the latters.

Note that the evaluation of the equivalent service time of each LAN may take into consideration the fact that there might exist many Hosts on the some LAN and that some of them might not be part of the federation.

To complete *Level-1* network model the GW_i, the **GW_x**, the WAN and the Hosts are also to be modeled.

The GW_i can be shown to consist of two stages (of Erl-5 and Cox-2 distribution, as in the illustration), with *parameters* (mean, variance, etc.) again calculated basing on the model of the software run by the gateway and its hardware capacity [11]. A similar work is done for **GW_x**, which is shown to consist of two stages (Exp and Erl-5) and relating *parameters*. As far as the WAN is concerned, this can be shown to be globally modeled by an Exp multi-server center, as illustrated in Fig.7 and relating *parameter*.

Finally, the two Hosts are modeled as being each divided into two sections to represent the division of work between the Layer 6 and 5 OSI protocol functions and the Layer 4 and 3 functions.

Level-2 abstraction: At this level, the window-type flow control procedure operated at transport level by the TCP protocol is modeled on the simplified network obtained at Level.1.

In order to represent such a flow control between the exit of the first section of Fed_i Host and the entrance of the first section of **Fed_x** Hosts, the so-called passive queue [9] is used, consisting of a token pool with the GET and RELEASE nodes. For a window size C, the pool consists of C tokens, and so up to C consecutive TCP frames can get a token at the GET node and be admitted. Non-admitted packets are enqueued in front of the GET node. On the other hand, each leaving packet releases its token at the RELEASE node, thus allowing another packet to enter. When data transfer takes place in the opposite direction, the GET node with its queue takes the place of the RELEASE node, and vice versa.

The Level-2 model is however still too complex to be evaluated in closed form, and thus its evaluation is made by simulation. The evaluation will yield the acknowledgement throughput [11] (or number of returned ACKs per time unit), denoted as *γ(C-n)*, where C is the chosen window size, n the number of acknowledged packets and (C-n) the number of still unacknowledged ones in the network.

Level-3 abstraction: At this level the entire network N_S (or N_D) is replaced by a single equivalent center (see Fig.8) whose service rate is the ACK throughput *γ (C-n)* calculated at abstraction level 2. In other words, the entire network N_S (or N_D) is now seen as a single server system with arrival rate λ (the packets arrival rate from the user application in Fed_i Host) and mean service time*parameters* of value $E(t_S)$ depending on the γ (C-n) throughput [11], namely:

$$E(t_s) = \begin{cases} 1/\gamma(i), & 0 \le i \le C \\ 1/\gamma(C), & i > C \end{cases} \qquad (2)$$

with *i* the number of packets in the queue, including the server. Such a model is of the M/M/1 type with state-dependent service time(i.e. dependent on the number n of packets in the center), which can be evaluated according to standardprocedures[5][9]. Its response time gives the network ΔN_S

[5]The Poisson assumption (M) for the arrival process of mean λ is a reasonable assumption for the packets flow from the client user application. The exponential assumption (M) for the network service time is a pessimistic assumption that introduces a security factor, which can however be replaced by a general service time assumption (G) by introducing the coxian approximation.

(or ΔN_D) delayto be used for N_S (or N_D) in the Fig.5 PM (in other words, the N_S or N_D equivalent service times to be used in the federation PM).

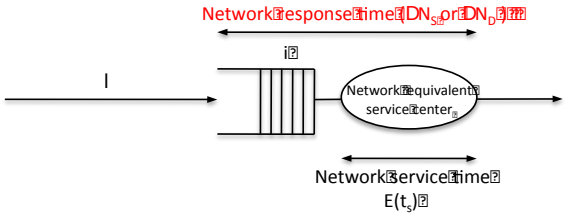

FIGURE 8: Synthetic model of N_S (or N_D)

As said at the beginning of this Section, by use of the N_S (or N_D) network model, what-if and sensitivity analysis can be performed of various network components (LANs, GWs, WAN, etc.) or of various functions (window size C) to answer the Fig.2 question "exists alternative N_S (or N_D)?" of the LS/DS decision procedure.

5. The OMNet++ version of the PM(DS(Σ)) and model parameterization

To perform an example prediction of the DS(Σ) execution time (T_{DS}) to be used in the Fig.2 decision procedure, we developed the OMNet++ simulation version of the Fig.5 model for a k=2 federates case (Fed_1 and Fed_2). Only the Fed_1 part (Fig.4) and the N_S and $N_D$nodes are shown in Fig.9. As said above, the model structure is valid for any system Σ and only its parameters, illustrated in Tab.1, (i.e., the CPU service time, the N_D and N_S service times, the p_{QUIT} and p_{SYNC} routing probabilities) may change with Σ.

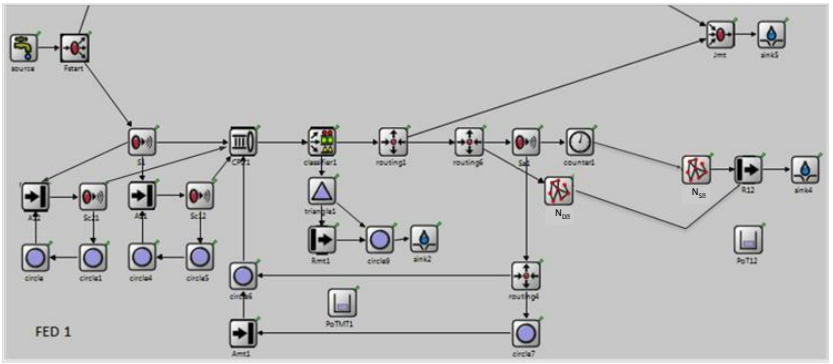

FIGURE9: OMNet++ simulation version of the PM(DS(Σ)) Fed_i.

The derived parameters for a Σ example case [12]are illustrated in Tab.1.

	Distribution	Parameters
Fed_i Host CPU service time t_{CPU} (i=1, 2)	*positive truncated-Normal*	$E(t_{CPU})$ = 10ms (Scen.A) $E(t_{CPU})$ = 500ms (Scen.B) $\sigma^2(t_{CPU})$ = 1
N_S, $N_D$service time t_S	*k-Pareto, k =4*	$E(t_S)$ = 21ms
Routing parameters	p_{QUIT}	0,001 (Fed_1); 0,001(Fed_2)
	p_{SYNC}	0,82 (Fed_1); 0,74 (Fed_2)

TABLE 1: Model parameters for a two-federate DS.

As can be seen from the table, there exist three types of parameters: the Fed_i Host CPU parameters, the $N_S$and $N_D$parameters and the routing parameters (p_{QUIT} and p_{SYNC}).

The Fed_i Host CPU service time parameters vary with the job class (C_i or C_{RC}) and are derived fromthe CPU capacity and the Fed_i software run by the CPU, as seen in Sect.3.1. For the sake of simplicity, in this example a common mean E(t_{CPU}) of10msor 500ms (for Scenarios A and B respectively, see later) is chosen for both classes.

The parameters for the N_D and N_S networks are instead derived from the software run by the network components and their capacity, as seen in Sect.3.2.

The routing parameters p_{QUIT} and p_{SYNC}, finally, can be derived from measurements on LS(Σ), in particular, by counting the number of events $n_{intEvents}$, $n_{disEvents}$, $n_{disToIntEvents}$ which respectively denote the number of local events (internal events), the number of events sent from a potential Fed_1 to a potential Fed_2, and the number of events received from a potentialFed_2. Such counting can be easily performed collecting the number of LS events in a simulation experiment for a given hypothetical LS partitioning into two federates. Indeed, it is possible to be convinced [12] that under the *conservative time-management* assumption, one may write:

$$p_{QUIT} = 1/ n_{cycles}, \quad (3)$$

where n_{cycles} is the number of local-HLA processing cycles. Value n_{cycles} can be estimated by the number of events locally processed within the model partition. More specifically,

$$n_{cycles} = n_{IntEvents} + n_{disToIntEvents} \quad (4)$$

Similarly, under the same assumption, one may write:

$$p_{SYNC} = n_{IntEvents}/ (n_{IntEvents}+n_{disEvents}) \quad (5)$$

Basing on the Fed_i Host CPU parameters, the N_S and N_D parameters and the routing parameters, the OMNet++ code simulation model has been run to obtain the T''_{DS}predictions shown in Tab.2. This was carried-out [13,14] in two scenarios A and B: Scenario A being one in which the fraction Q of inherently serial computation was high and Scenario B in which Q was low.

The first column in Tab.2 reports the local simulator execution time T_{LS}. The second column reports the distributed simulator execution time T''_{DS} predicted by OMNet++ simulator of the PM, and the third column the times of the real distributed simulator DS (that was implemented in Java+HLA). Such a column thus provides a validation of the PM results, and shows how the predicted results adequately match the real ones. Note that in Scenario B the execution times are in minutes while in Scenario A they are in seconds. This is since Scenario B is built in a way to yield a high computation-to-communication ratio. In other words, a large amount of computation between communications.

	T_{LS}	**PM results (OMNet++ predictions)**	**PM validation (real DS measurements)**
A (high Q)	0.7s	T''_{DS} = 8.3s	T''_{DS} = 8.2s
B (low Q)	33 min	T''_{DS} = 12.5 min	T''_{DS} = 12.0 min

TABLE 2: Execution-time results.

Tab.2 also shows how in the Scenario B the distributed simulator outperforms the local one. Indeed, in such a Scenario the DS execution time (T''_{DS}) is much lower than the LS time (T_{LS}).

Finally by using the expression $S'' = T_{LS}/T''_{DS}$, the results in Tab.2 were used to obtain the speedup results shown in Tab.3.

	PM results
A: High Q	S = 0.08
B: low Q	S = 2.64

TABLE 3: Speedup results.

This table shows that a quite good speedup (S'' =2.64) is obtained in the B Scenario. In other words, in this case the run-time gain obtained by the parallel execution on two hosts compensates for the data and synch communication overheads. In the scenario A, instead, the parallelism does not yield a sufficient run-time gain to compensate for the overheads, and the resulting speedup (S=0.08) is practically irrelevant.

The Tab.2 and 3 results are used by the decision procedure of Fig.2 to decide at design-time whether to remain on the LS version of the simulator or implement its DS version. In case the T''_{DS} execution time are not considered "ok" (see Fig.2), one may either try a new tentative partitioning (to modify the p_{SYNC} parameters, see Sect.3.1) or try alternative networks N_S and N_D of improved capabilities (to modify the $E(t_S)$ parameters, see Sect.3.2). In case no partitioning nor network improvements can be found, one may decide not to implement the DS(Σ).

6. CONCLUSION

The execution time of a Distributed Simulator (DS) depends on 3 interacting factors: the speedup, the synch-communication overhead and the data-communication overhead, due to network delays.
The combination of such 3 factors makes it very hard to predict the advantage of transforming a local version of the simulator (LS) into a distributed version (DS).

A LS/DS decision procedure has been proposed to decide at design-time whether to remain on the LS version of the simulator or carry out the implementation of its DS version. The procedure is guided by a performance model (PM) of the DS. The PM assumes the DS is based on the HLA protocol standard and middleware. The model can be used both to support the LS/DS decision process and to evaluate the representativeness of the DS(Σ) at design-time.

ACKNOWLEDGMENTS

Work partially supported by funds from the FIRB project on "Software frameworks and technologies for distributed simulation", from the FIRB project on "Performance evaluation of complex systems", from the University of Rome TorVergata research on "Performance modeling of service-oriented architectures" and from the CERTIA Research Center.

REFERENCES

[1] D. Gianni, A. D'Ambrogioand G. Iazeolla. "A Layered Architecture for the Model-driven Development of Distributed Simulators", Proceedings of the First International Conference on Simulation Tools (SIMUTOOLS'08), Marseille, France, pp. 1-9, 2008.

[2] A. D'Ambrogio, D. Gianni, G. Iazeolla: "A Software Architecture to ease the development of Distributed Simulation Systems", Simulation-Transaction of the Society for Modeling and Simulation International, Vol. 87, n.9, pp. 819-836, 2011.

[3] IEEE Std 1516. "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) frameworks and rules", 2000.

[4] R.M. Fujimoto. “Parallel and Distributed Simulation Systems”, John Wiley & Sons 1999.

[5] A. Park. “Parallel Discrete Event Simulation”, College of Computing. vol. PhD: Georgia Institute of Technology, 2008

[6] OMNeT++ Discrete event simulation v.4.0. User Manual, http://www.omnetpp.org.

[7] Pitch. The Certified Runtime Infrastructure for HLA 1516 – User’s guide. http://www.pitch.se, 2005.

[8] F. Kuhl, R. Weatherly, J. Dahmann. “Creating Computer Simulation Systems”, Prentice-Hall, 1999.

[9] S.S. Lavenberg. “Computer Performance Modeling Handbook”, Academic Press, New York, 1983.

[10] P.J. Courtois. “Decomposability: Queueing and Computer System and Applications”, Academic Press, 1997.

[11] A. D’Ambrogio, G. Iazeolla. “Steps towards the Automatic Production of Performance Models of Web-Applications”, Computer Networks, n.41, pp 29-39, Elsevier Science, 2003.

[12] D. Gianni, G. Iazeolla, A. D’Ambrogio. “A methodology to predict the performance of distributed simulation”, PADS10 the 24th ACM/IEEE/SCS Workshop on Principles of Advanced and distributed simulation Atlanta May 17-19, 2010.

[13] G. Iazeolla, A. Gentili, F. Ceracchi. “Performance prediction of distributed simulations”, Technical Report RI.02.10, Software Engineering Lab, Dept. Computer Science, University of Roma TorVergata , 2010.

[14] G. Iazeolla, M. Piccari. “The Speedup in distributed simulation”, Technical Report RI.03.10, Software Engineering Lab, Dept. Computer Science, University of Roma TorVergata, 2010.

[15] L. Chu-Cheow,L. Yoke-Hean, et al. “Performance prediction tools for parallel discrete-event simulation”, Proceedings of the thirteenth workshop on Parallel and distributed simulation Atlanta, Georgia, United States: IEEE Computer Society, 1999.

[16] J. Liu, D. Nicol, et al. “A Performance prediction of a parallel simulator”, Proceedings of the thirteenth workshop on Parallel and distributed simulation Atlanta, Georgia, United States: IEEE Computer Society, 1999.

[17] R. Ewald,D. Chen, et al. “Performance Analysis of Shared Data Access Algorithms for Distributed Simulation of Multi-Agent Systems”, Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation: IEEE Computer Society, 2006.

[18] R. Ewald, J. Himmelspach, et al. “A Simulation Approach to Facilitate Parallel and Distributed Discrete-Event Simulator Development”, Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications: IEEE Computer Society, 2006.

[19] K. S. Perumalla, R.M. Fujimoto, et al. “Performance prediction of large-scale parallel discrete event models of physical systems”, Proceedings of the 37th conference on Winter Simulation Orlando, Florida: Winter Simulation Conference, 2005.

[20] P. Teo, S. J. Turner, Z. Juhasz. “Optimistic Protocol Analysis in a Performance Analyzer and Prediction Tool”, PADS '05 - Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation Pages 49 – 58, 2005.