11th Int. WS on Simulation & EGSE facilities for Space Programmes (SESP 2010)

SimArch: A Layered Architectural Approach to Reduce the Development Effort of Distributed Simulation Systems

Daniele Gianni[(1)], Andrea D'Ambrogio[(2)], and Giuseppe Iazeolla[(2)]

[(1)]European Space Agency
Kepleerlan 1, Postbus 299,
2200 AG, Noordwijk, The Netherlands
daniele.gianni@esa.int

[(2)]Department of Computer Science
University of Roma TorVergata
Via del Politecnico 1,
00133 Rome, Italy
{dambro, iazeolla}@info.uniroma2.it

INTRODUCTION

Distributed Simulation (DS) has gained popularity for the advantages of increased computational capabilities, simulator reusability and distributed execution [1]. Factors affecting the effectiveness of the DS approach are the skills and the extra-effort that the development of a DS system requires with respect to the equivalent Local Simulation (LS) system. In the case of IEEE High Level Architecture (HLA) [2], the most prominent technology for DS, the extra-effort has been estimated to be up to 60% of the effort required for the development of the equivalent LS system. In addition, the coding of extra 3.5KLOC (Lines Of Code) per federate is generally required [3][4].

In this paper, we present *SimArch*, a layered simulation architecture that considerably reduces the above extra-effort by enabling developers to transparently obtain the DS system from the equivalent LS one. *SimArch* defines five layers, each introducing a more abstract set of simulation services on top of the underlying layer, the bottom one being the distributed computing infrastructure. Several advantages can be obtained by adopting the *SimArch* layered approach [3]. These advantages concern the portability of simulation model over several simulation platforms and the availability of high-level simulation primitives, which use practically eliminates the DS development extra effort and avoids the coding of the additional 3.5KLOC per federate [3][4].

The paper is organized as follows. High Level Architecture section provides a brief introduction on the respective standard. *SimArch* section presents an overview on the layered architecture. Example application section presents the development of a DS system to simulate a ground segment's computer network for a Global Navigation Satellite System (GNSS).

HIGH LEVEL ARCHITECTURE

HLA is a standard that provides a general framework within which software developers can structure and describe simulation applications [2]. The standard promotes interoperability and reusability of simulation components in different contexts, and is based on the following concepts [5]: *Federate*, which is a simulation program and represents the unit of reuse in HLA; *Federation*, which is a distributed simulation execution composed of a set of federates; *Run Time Infrastructure (RTI)*, which is the simulation oriented middleware coordinating the federates and consists of a RTI Local (run at federate sites) and of a RTI Executive (run on a central server).

The standard is defined by five documents: *1516 HLA Rules*, which govern the behaviour of both federation and federates [2]; *1516.1 Object Model Template*, which defines the formats for documenting HLA simulations [6]; *1516.2 Interface Specification*, which defines both RTI - federate (*RTIAmbassador*) and federate - RTI (*FederateAmbassador*) interfaces [7]; *1516.3 Federate Execution and DEvelopment Process*, which provides a reference process for the development of HLA simulation systems [8]; *1516.4 Verification, Validation and Accreditation (VV&A)*, which defines the VV&A practices for HLA federates developed following the FEDEP [9].

The major improvement that HLA brings in with respect to its predecessors is an API-oriented development of distributed simulation systems. Compared to Protocol-oriented techniques (e.g. DIS [10] or ALSP [11]), HLA raises developers from all the concerns related to the DS communication and synchronization, and thus obtaining considerable effort savings in the development process. Besides this improvement, however, HLA still suffers from two main

drawbacks. The *first* is that the API is rather complex because it consists of a wide set of generic simulation services. Furthermore, the services concern only with the distributed environment and a considerable effort is always needed to develop the synchronization and communication logic between the local and the distributed environment [12]. The *second* drawback is that the standard leaves the communication protocol between the *RTI Local* and *RTI Executive* undefined. As consequence, federates cannot interoperate unless their *RTI Local* and *Executive* belong to the identical version released by the same vendor [13].

The overall consequence is that a considerable extra effort is necessary to develop a DS system when using HLA (compared to an equivalent LS one). According to experience gained from various experiments [3][12][14], this extra effort can be broadly outlined as follows:

- *Extra effort to acquire HLA knowledge and skills*: 30% for an average experienced developers, and 60% for a beginner, with respect to conventional LS systems.
- *Extra coding effort to create HLA federates*: about 3.5K extra LOC per federate.
- *Extra design effort to determine design choices*: for example which federates are to develop, which can be reused, which time advancement modality and simulation paradigm need to be adopted, which data need to be exchanged and among which federates, and which communication modalities best suite the simulator.

Using *SimArch*, developers can overcome such difficulties and are enabled to develop DS systems with no knowledge of HLA (or of any DS standard), with no extra LOCs per federate, with no HLA-related design choices.

SIMARCH

SimArch organises distributed simulation systems in several software layers, each introducing a more abstract set of simulation services on top of the underlying layer, with the bottom one being the distributed computing infrastructure. There are three main advantages for adopting this layering approach [3][4]. The *first* is that the simulation model is decoupled from the specific execution environment, and thus it can be reused across several simulation platforms. The *second* is that layers' implementations can be easily modified or replaced to accommodate custom deployment requirements (i.e. local or distributed deployment, performance optimization for given simulation workload, etc.). Finally, the *third* advantage is that simulation developers deal only with high level simulation services, and therefore developers can focus on the model description rather than being concerned with technical issues of distributed simulator implementations. *SimArch* consists of five layers [3], each dealing with a specific distributed simulation issue. Fig. 1 illustrates the architecture layers, which detailed description is given in [15].

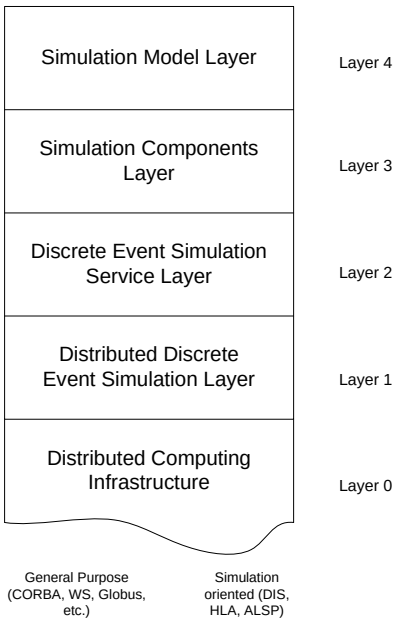

Fig.1 SimArch layers

In *SimArch*, the top Layer 4 is the layer where the simulation model is defined through the invocation of the simulation language primitives. The primitives' implementation, i.e., the components' simulation logic and the model configuration services, are provided by Layer 3; while Layer 2 deals with the simulation components synchronization and communication, transparently for local and distributed environments. The distributed version of this layer uses in turn Layer 1 to achieve global time synchronization and provides communication with the remote simulation components. Finally, Layer 1 provides a DES (discrete-event simulation) abstraction on top of the distributed computing infrastructure conventionally identified by Layer 0. Such bottom layer is structured within *SimArch* but its implementations do not belong to SimArch. As a consequence SimArch does not define the service interfaces between Layers 1 and 0. In the case of a HLA-based implementation of Layer 1, such interfaces are subsets of the *RTI-Ambassador* and *FederateAmbassador* interfaces for the communication between Layers 1 and 0 and between Layers 0 and 1, respectively.

For the sake of conciseness SimArch's services and data interfaces are not included in this paper. Interested readers may refer to [3][4][12][14][15][16] for details on the interfaces and implementation.

EXAMPLE APPLICATION

There are many complex systems that can be particularly convenient to simulate using a DS system. Generally, DS can be motivated by factors concerning needs for increase of simulator scalability, reusability and aggregability of available simulators, or exploitation of intrinsic simulator parallelism. More recently, the inherent distributed nature of the simulated systems has been raised as another possible motivation for the use of DS. The simulation of an inherently

distributed system may indeed take advantage of the distributed characteristics of a simulator. For example, a DS system can offer increased accuracy when incorporating communication using the actual network infrastructure between two sites instead of simulating it [17]. Similarly, a DS system can also exploit distributed input data streams from actual systems, to provide an increased accuracy of the simulated system. This can be the case motivating the use of DS for the simulation of a ground segment. In the simulation of a ground segment, a DS system could provide higher accuracy by using data streams generated by the space segment and received by antennas located within the ground facilities. These facilities are likely to be interconnected by communication network and geographically distributed over the Earth, and therefore a local simulator would not be suitable to use such data streams. An example case can be a computer network for the dissemination of TM/TC and Mission and Control Data between two facilities (a main facility and a back-up facility) that constitute the GNSS ground segment. Such a computer network can be heterogeneous and may consist of individual components, such as LANs of various types, gateways, several stages of the WAN for the intra-LAN communication, various types of hosts communicating by the LANs and the WAN, etc. It may also happen that the simulation systems of each one of those components is already available and maintained in different geographically remote sites. It could be thus convenient to reuse of all those disperse local simulation subsystems into a unique distributed simulation system. The traditional way of performing such a combination of remote simulation subsystems is by use of HLA. As stated above, *jEQN* may render this work extremely less difficult and yet economically efficient.

Fig. 2 illustrates an example of such a system. The system can be composed of:

(a) A set of LANs. In our scenario, we specifically consider only two LANs: a token ring (*LAN1*) that connects Host A, and an Ethernet (*LAN2*) that connects Host B;

(b) A set of communicating hosts connected to the LANs. In our scenario, we consider only two hosts, *Host A* and *Host B*, and two gateways, *GW1* and *GW2*, which connect LAN1 and LAN2 to the WAN, respectively;

(c) The general geographic communication backbone, generically denoted with the term WAN.

We assume that the interaction between the client (Host A) and the server (Host B) is based on message exchanges carried out by packet flows over the various components of the system, and that WAN is a X.25 Packet Switching network. The packet flow involves several technologies (Token Ring, X.25, Ethernet, etc.) and thus several mechanisms are necessary to deal with heterogeneity, namely:

(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 Host A to Host B, 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).

The performance evaluation of computer network systems generally relies on an Extended Queueing Networks (EQN) models of the system. An EQN model consists of a set of jobs flowing through a network of service centers and queues. A job represents a user of the system resources (e.g. network or host's CPU). Each service center represents a system resource for which user might need to compete for its use. Each queue represents the scheduling mechanisms that need to take place when a job is occupying a service center. Approximately, in a computer network system, an IP packet can be modelled as user, a network connection (e.g. WAN) can be modeled as service center, and a First-Come-First-Serve queue can be introduced to represent delay because other network traffic.

Once available a model for Fig. 2 system, the model can be used for many evaluations, such as Host A – Host B end-to-end delay. However, the discussion of this model and of its applications is out of this paper scope. Further details about the model can be found in [18].

In the following section, we illustrate how a DS system for Fig. 2 ground segment can be mechanically derived from the equivalent LS system. However, we do not consider the presence of the space segment and of antenna 1 and 2, and thus the incorporation of data streams from the space segment, as this aspect only constitutes the motivation for the use of DS for simulation of a ground segment and does not affect this paper methodology.

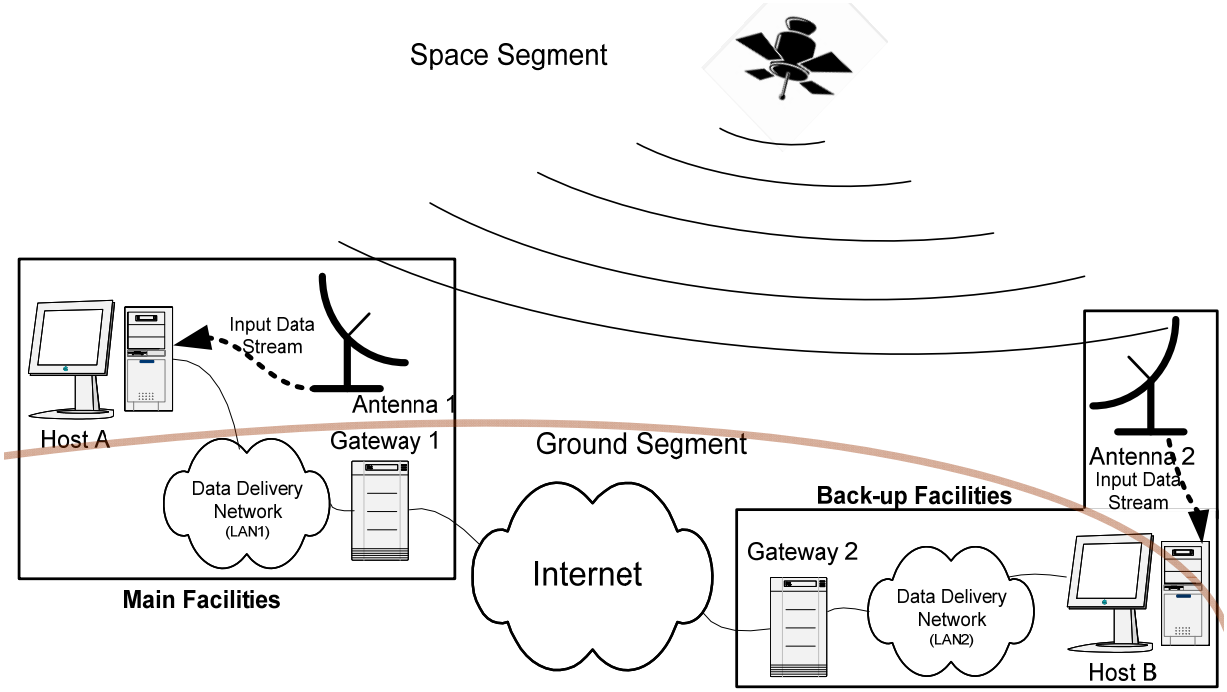

Fig. 2 Overall architecture of a GNSS ground segment's (main and back-up facilities) computer network

Simulator Development

Using *SimArch* and *jEQN*, we first developed the local version of the simulator [3] and then we derived the distributed one. The derivation process for the distributed simulator bases on a mechanical list of steps, which thus demonstrates how straightforward transforming a *jEQN* local simulation system into a distributed one is.

Fig. 3 illustrates the system model high-level description in terms of *jEQN* entities, links, input and output ports. An entity represents a system component (either logical or physical), a link represents a connection (either logical or physical) between two or more system components, and input and output ports represent a system component's sockets (either logical or physical) through which a link can be established. The entities defined in Fig. 3, and their interconnections, derive from Fig. 3 system (excluding the space segment and the antennas) and define the flow of the messages according to the model definition. For each entity, the simulator defines an output port per each point-to-point connection to another entity. Differently, each entity is provided with only an input port for all the incoming connections. This asymmetry is due to the fact that inside the software components no distinction is made upon the source of the incoming messages as these are processed only depending to the entity state and the associated semantic. However, when needed, the recipient entity can infer the message sender by accessing the respective data available within the message.

In the considered local simulation system version, a *jEQN* simulation entity is introduced for each main system entities in Fig. 3. The software architecture thus includes four types of entities: Host, LAN (both Token Ring and Ethernet), Gateway and WAN. The model defines two Hosts, a Token Ring, an Ethernet, two Gateways and the WAN, which are logically connected in *jEQN* as shown in Fig. 3. The Hosts can send requests through the respective output ports. The port is connected to the entity representing the local area network, which simulates the operations in such network before forwarding the message to the Gateway and then to the WAN. Once a message reaches the recipient Host, the process starts over, in reverse order. From this simple description, it remains of immediate understanding the number of output ports each entity has. For instance, LAN 1 needs to communicate to the Gateway, when forwarding the Host's request, and to the Host, when conveying the response. Analogous considerations hold for the other entities.

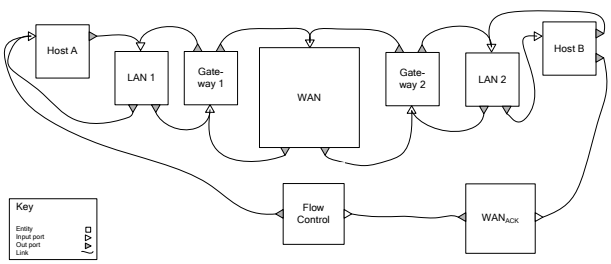

Fig.3 jEQN software architecture of the simulation system (local version)

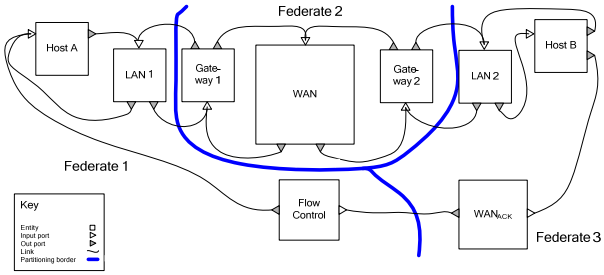

Fig.4 jEQN software architecture of the simulation system (distributed version)

In the software architecture of Fig. 3, the connections between Hosts, LANs, Gateways, Flow Control and WAN reflect the connections between the main components of the real system of Fig. 2.

The software architecture for the DS system can be described in terms of the above defined local version with minor adjustments. In particular, the sections of the model that concern only with local items, either an entity or a connection, are declared by the same statements of the local simulation system. Differently, the declarations of the parts involving references to remote items are to be defined by use of the *distributed versions* of the entity and connection components. Determining which items are to be declared with a distributed version is very easy and can systematically be inferred by the model partitioning, which defines the entities running on each host and the connection between two or more submodels. A very intuitive procedure to carry out such task is to draw the model entities and their connections and then operate the partitioning by drawing a continuous line that separates the submodels. The connections crossing the partitioning line will connect a local entity with a remote entity, and therefore have to be declared by the distributed version of the connection. Concerning the entities, a proxy for a remote entity has to be included locally in the definition of the submodel for each remote entity connected to the remote end of the above defined remote connections.

Following this procedure, assume we decide to partition the simulated model into three submodels as shown in Fig. 4. Specifically, Submodel 1, Submodel 2, and Submodel 3, where *Submodel 1* consists of the HostA and LAN1 entities; *Submodel 2* consists of the GW1, WAN and GW2 entities; and, finally, *Submodel 3* consists of the HostB and LAN2.

The distributed simulation system will be then composed of three federates, each simulating a Fig. 4 submodel, in addition to the standard HLA Federation Manager and the RTI [5], which are known components of any HLA-like distributed simulation system and therefore not discussed further.

As shown in the Fig. 4 distributed architecture, Federate 1 includes the definition of the local entities and local connections specified in the respective submodel. Fig. 5 shows how the general guideline applies to this Federate. The local entities, which are drawn with a continuous line, are Host A, LAN 1 and Flow Control, which regulates the packet transmission between host A and B. These entities are also locally connected and their connections are also shown with

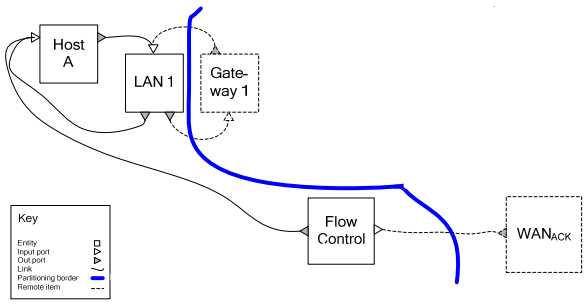

Fig.5 Federate 1

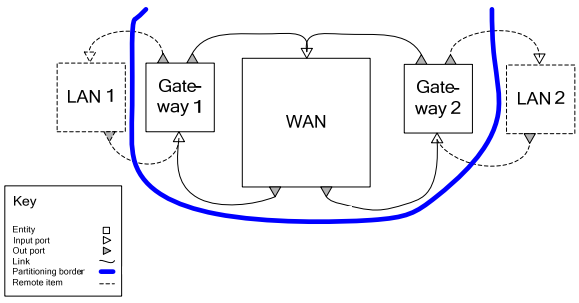

Fig.6 Federate 2

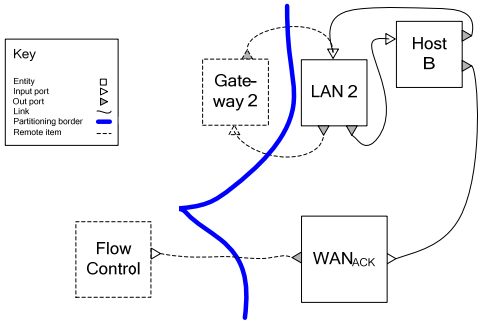

Fig.7 Federate 3

continuous lines. The declaration of such entities and of their connections follows the same statements for the local simulation system, which can therefore be copied and pasted in the federate code. Federate 1 also needs to declare the remote entities that can receive messages from the local entities, e.g. Gateway 1. Gateway 1 declaration can be done through the following statement that is specific of the distributed version.

Turning to Federate 2, as shown in Fig. 4, this federate consists of Gateway1, WAN and Gateway2 entities. In Fig. 6 such entities are local to the federate and thus are drawn with a continuous line. Differently, the confining non local entities are drawn by a dashed-line. The declaration of the local entities as well as the declaration of their connections can be copied by the respective segment of the local simulation system. Differently, the declaration of the confining entities and their connections with the local entities can be done by the special statements that are specific for the distributed version.

Finally, as shown in Fig. 4, Federate 3 consists of LAN2 and Host B entities. The coding of Federate 3 follows the same approach as for the other federates. The entities to be locally declared can be similarly inferred by the partitioning in Fig. 4. They are LAN2 and Host B, which are drawn with a continuous line in Fig. 7. Their declarations and the declaration of their connections can be copied from the respective segment of the local simulation system. Similarly, the remote entities and connections, for which a special distributed version is needed, are inferred from Fig. 4 and reported in Fig. 7 with a dashed line.

Once the distributed simulation system is developed, it can be started by first activating the RTI software, then activating the Federation Manager [5] and finally, in order, Federate 1, Federate 2 and Federate 3. The RTI software and the Federation Manager have to be properly configured.

Experimental Setup

We tested SimArch in national and international DS experiments. In particular, we carried out experiment in intercontinental configuration between Rome and Atlanta, using Pitch pRTI 1516 and CORBA-HLA [13], an interoperable HLA implementation that builds on Pitch pRTI. The details of SimArch experiments based on pRTI can be found in [19]**.** Concerning SimArch experiments based on CORBA-HLA, the experimental set-up is shown in Fig. 8. Five hosts were allocated in Rome: two hosts running Federate 1 and 2, one host running the Federation Manager, and two hosts running the CORBA-HLA and Pitch pRTI. One host was allocated in Atlanta, running the remaining Federate 3. The purpose of the experiment was not only to functionally test *SimArch* implementation, but also to validate the distributed *SimArch* execution of the model with respect to the local *SimArch* execution. The distributed execution produced numerical results that conform to the ones obtained from the local execution, considering reasonable error margins. Further details about the results of the experiments are available in [3] [20].

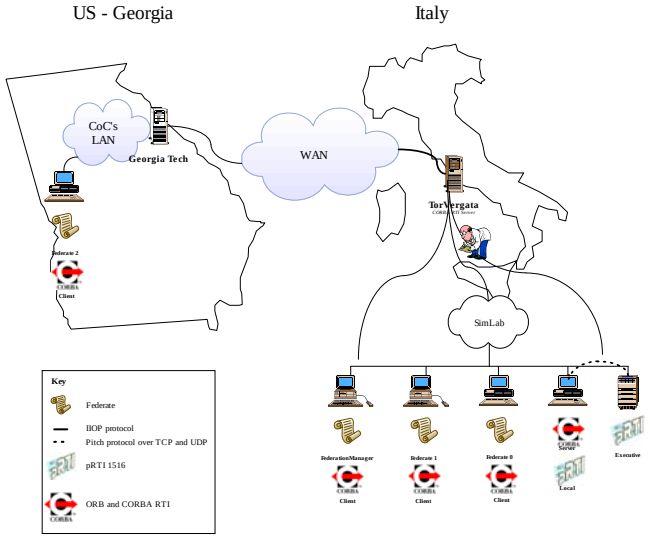

Fig.8 IWAN Execution configuration [21]

CONCLUSIONS

Building a DS system requires non negligible efforts and time, and specialized skills, in comparison to the development of a conventional local simulator. This extra effort has been estimated to be up to 60% of the effort required for the development of the equivalent LS system. In addition, the coding of extra 3.5KLOC per federate is generally required. In this paper, we have presented *SimArch* a layered architecture that transparently supports the local and distributed execution of simulation components. In this way, simulator developers can code DS systems as conventional LS systems, i.e. with no extra effort and no extra knowledge. Currently, *SimArch* implementation is based on IEEE HLA, but other standards and DS infrastructures can be transparently introduced. Similarly, *jEQN*, a simulation language for Extended Queueing Networks (EQN) models, is provided. However, other languages can also be developed and transparently integrated in the current implementation. We have also presented the details of an experimental setup that

we used to validate *SimArch* distributed executions with *SimArch* local executions. Further references to this paper technologies are also provided for interested readers. In the context of other projects, we have provided *SimArch* with a language for the simulation of wireless systems [22] and we have also extended the architecture to support agent-based simulation [23][24][25].

ACKNOWLEDGEMENTS

This work has been partially supported by the FIRB Project "Performance Evaluation of Complex Systems", funded by the Italian Ministry of Research and by the University of Roma TorVergata; by the FIRB Project "Software frameworks and technologies for the development of open-source distributed simulation code", funded by the Italian Ministry of Research; by CERTIA Research Center of the University of Roma TorVergata; by the FP7 euHeart Project, funded by the European Commission (FP7-ICT-2007-224495); and by the Internal Research Fellowship Schema of the European Space Agency. The authors would also like to thank prof. R. Fujimoto and the PADS laboratory at Georgia Institute of Technology, for their cooperation in the distributed experiments.

REFERENCES

[1] R. Fujimoto, Parallel and Distributed Simulation Systems, Wiley (2000).

[2] IEEE: Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - frameworks and rules. Technical Report 1516, IEEE (2000).

[3] D. Gianni, A. D'Ambrogio, and G. Iazeolla, "A Layered Architecture for the Model-driven Development of Distributed Simulator", *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTOOLS08)*, March, 2008, Marseille, France.

[4] D. Gianni, A. D'Ambrogio, and G. Iazeolla, "A Software Architecture to Ease the Development of Distributed Simulation Systems", *Simulation*, SCS, 2010 (to appear).

[5] F. Khul, R. Weatherly, J. Dahmann, Creating Computer Simulation Systems: An Introduction to High Level Architecture. Prentice Hall (1999).

[6] IEEE: Standard for modeling and simulation (M&S) High Level Architecture (HLA) - object model template (OMT) specification. Technical Report 1516.2, IEEE (2000).

[7] IEEE: Standard for modeling and simulation (M&S) High Level Architecture (HLA) - federate interface specification. Technical Report 1516.1, IEEE (2000).

[8] IEEE: Recommended practice for High Level Architecture (HLA) federation development and execution process (FEDEP). Technical Report 1516.3, IEEE (2003).

[9] IEEE: Recommended practice for High Level Architecture (HLA) verification, validation and accreditation (VV&A) of a federation-an overlay to the High Level Architecture Federation Development and Execution Process. Technical Report 1516.4, IEEE (2009).

[10] IEEE 1278-1993 - Standard for Distributed Interactive Simulation - Application protocols, IEEE (1993).

[11] Anita Adams, Gordon Miller, and David Seidel, November 1993, "Aggregate Level Simulation Protocol (ALSP) 1993 Confederation Annual Report", The MITRE Corporation.

[12] A. D'Ambrogio, D. Gianni, and G. Iazeolla, "SimJ: a Framework to Distributed Simulators", *Proceedings of the 2006 Summer Computer Simulation Conference (SCSC06)*, Calgary, Canada, 2006, pp. 149 – 156.

[13] A. D'Ambrogio and D. Gianni, "Using CORBA to enhance HLA interoperability in Distributed and Web-based Simulation", *Proceedings of International Symposium on Computer and Information Science 2004 (ISCIS04)*, Antalya, Turkey, Springer-Verlag, pp 696-705, Oct, 2004.

[14] A. D'Ambrogio, D. Gianni, G. Iazeolla, and A. Pieroni, "Distributed simulation of complex systems by use of an HLA-transparent simulation language", *Proceedings of the 7th International Conference on System Simulation and Scientific Computing, ICSC 2008*, Asia Simulation Conference, Beijing, China, pp. 460-467, Oct, 2008.

[15] D. Gianni and A. D'Ambrogio, "A Language to Enable Distributed Simulation of Extended Queueing Networks", *Journal of Computer*, Vol. 2, N. 4, July, 2007, Academy Publisher, pp. 76 – 86.

[16] D. Gianni and A. D'Ambrogio, "A Domain Specific Language for the Definition of Extended Queueing Networks Models", *Proceedings of the IASTED International Conference on Software Engineering (SE 2008)*, February, 2008, Innsbruck, Austria.

[17] G. Iazeolla, A. Pieroni, A. D'Ambrogio, and D. Gianni, "A Distributed Approach to the Simulation of Inherently Distributed Systems", *Symposium On Theory of Modeling and Simulation DEVS Integrative M&S Symposium, 2010 Spring Simulation Multiconference*, April, 2010, Orlando, US.

[18] A. D'Ambrogio and G. Iazeolla, "Steps towards the Automatic Production of Performance Models of Web-Applications", *Computer Networks*, n. 41, pp 29-39, Elsevier Science, 2003.

[19] Pitch, The Certified Runtime Infrastructure for HLA 1516, http://www.pitch.se (last access July 2010)

[20] D. Gianni, "Software Technologies for the Effortless Development of Geographically Distributed Simulators", PhD Thesis, University of Rome Tor Vergata, June, 2007.

[21] A. D'Ambrogio, D. Gianni and G. Iazeolla, "Software Technologies for the Interoperability, Reusability and Adaptability of Distributed Simulators", *Proceedings of the 2007 European Simulation Interoperability Workshop (Euro-SIW07)*.

[22] G. Iazeolla, A. Pieroni, A. D'Ambrogio, and D. Gianni, "A Distributed Approach to Wireless System Simulation", *The 6th Advanced International Conference on Telecommunications* (AICT 2010), Barcelona, Spain, May 9-15, 2010.

[23] D. Gianni, "Bringing Discrete Event Simulation Concepts into Multi Agent Systems", *Proceeding of the IEEE 18th International Conference on Computer and Modelling and Simulation (EuroSim08/UKSim08)*, pp. 186-192, Cambridge, UK, April, 2008.

[24] D. Gianni, A. D'Ambrogio, and G. Iazeolla, "DisSimJADE: A JADE-based framework for the Distributed Simulation of Multi-Agent Systems", *Proceedings of the Second International Conference on Simulation Tools and Techniques for Communications (SIMUTOOLS09)*, Rome, March, 2009.

[25] D. Gianni, A. D'Ambrogio, G. Iazeolla, and A. Pieroni, "HLA-transparent Distributed Simulation of Agent-based System", *Modeling Simulation and Optimization - Focus on Applications*, INTECH, pp. 205-223, March, 2010.