# ARTÌS: A Parallel and Distributed Simulation Middleware for Performance Evaluation*

Luciano Bononi, Michele Bracuto, Gabriele D'Angelo, and Lorenzo Donatiello

Dipartimento di Scienze dell'Informazione, Università degli Studi di Bologna
Via Mura Anteo Zamboni 7, 40126, Bologna, Italy
{bononi,bracuto,gdangelo,donat}@cs.unibo.it

**Abstract.** This paper illustrates the motivation, the preliminary design and implementation issues, of a new distributed simulation middleware named Advanced RTI System (ARTÌS). The aim of the ARTÌS middleware is to support parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components. The ARTÌS design is oriented to support the model components heterogeneity, distribution and reuse, and to increase the simulation performances, scalability and speedup, in parallel and distributed simulation scenarios. Another design issue of the ARTÌS framework is the dynamic adaptation of the interprocess communication layer to the heterogeneous communication support of different simulation scenarios. In this paper we illustrate the guidelines and architecture that we considered in the design and implementation of the ARTÌS middleware, and we sketch some case studies that demonstrated the ARTÌS utility and motivation, e.g., a distributed simulation of massively populated wireless ad hoc and sensor networks.

## 1 Introduction

The design of complex systems composed by many heterogeneous components requires appropriate analysis methodologies and tools to test and to validate the system architectures, the integration and interoperability of components, and the overall system performances [18]. The performance evaluation of complex systems may rely on simulation techniques because the model complexity obtained with alternative analytical and numerical techniques often results in unpractical or unaffordable methods and computation [18,8,1,2,9,25]. Well known sequential and monolithic event-based simulation tools have been created for analyzing general purpose system models (e.g., computation architectures, systems on chip, database systems) [15,16] and more targeted system models (e.g., computer networks) [14,17]. The problem with a sequential monolithic simulator is that it must rely on the assumption of being implemented on a single execution unit, whose resources may be limited, and it cannot exploit any degree of

* This work is supported by MIUR FIRB funds under the project "Performance Evaluation of Complex Systems: Techniques, Methodologies, and Tools".

computation parallelism. To obtain a significant insight of a complex system, detailed and fine-grained simulation models must be designed, implemented and executed as a simulation process, often resulting in high computation and high memory allocation needs.

This fact translates in computation and memory bottlenecks that may limit the complexity and the number of model components (i.e., the simulated system scalability) that can be supported by the simulation process. One solution to overcome these limitations can be found in parallel and distributed simulation techniques, in which many simulation processes can be distributed over multiple execution units. The simulation semantics, the event ordering and event causality can be maintained and guaranteed with different approaches (e.g., optimistic vs. conservative), by relying on distributed model-components' communication and synchronization services.

Parallel and distributed simulation platforms and tools have been demonstrated to be effective in reducing the simulation execution time, i.e., in increasing the simulation speedup. Moreover, parallel and distributed platforms could exploit wide and aggregate memory architectures realized by a set of autonomous and interconnected execution units, by implementing the required communication and synchronization services. Examples of the Parallel and Distributed Discrete Event Simulation (PDES) approach can be found in [8,9], e.g., Glomosim [25] based on PARSEC [1], Maisie [22], parallel and distributed implementations based on Network Simulator (ns-2) [14,20] based on RTI-Kit [20], on ANSE/WARPED [19], Wippet [12], SWiMNET [3], and many others [13,22]. More recently, the distributed simulation world agreed on the need for standards, and converged in the definition of a new standard, named IEEE 1516 Standard for parallel and distributed modeling and simulation [11]. The High level Architecture (HLA) has currently become a synonymous for the middleware implementation of distributed simulation services and a RunTime (RTI) simulation kernel, based on the IEEE 1516 Standard definition [11,4,5].

Unfortunately, the need for distributed model-components communication and synchronization services may require massive interprocess communication to make the distributed simulation to evolve in correct way. Complex systems with detailed and fine-grained simulation models can be considered communication-intensive under the distributed simulation approach. As a result, interprocess communication may become the bottleneck of the distributed simulation paradigm, and solutions to reduce the cost of communication must be addressed by the research in this field [8,2,9,24]. Additional research studies, aiming to exploit the maximum level of computation parallelism, dealt with dynamic balancing of logical processes' executions (both cpu-loads and virtual time-advancing speeds) by trading-off communication, synchronization and speedup, both in optimistic and conservative approaches [7,10,23,24].

The efficient implementation of interprocess communication is required as a primary background issue, to overcome the possible communication bottleneck of parallel and distributed simulations. The way interprocess communication can be sustained in distributed systems would depend mainly on the execution units' architectures and on the simulation system scenario. Recently proposed and implemented middleware solutions based on the IEEE 1516 Standard for distributed simulation and the High level Architecture (HLA) [11,5,4] have shown that the parallel and distributed simulation of massive and complex systems can suffer the distributed communication bottlenecks, due to suboptimal implementation of the interprocess communication services, over the simulation execution platform.

In this paper we propose an overview of the design, preliminary implementation results and guidelines, for a new, parallel and distributed simulation middleware named Advanced RTI System (ARTÌS). The design of the ARTÌS middleware architecture is based on the guidelines provided by the analysis and evaluation of existing HLA-based RTI implementations, and on the observations about the sub-optimal design and management of distributed interprocess communication. Specifically, we oriented the ARTÌS design towards the adaptive evaluation of the communication bottlenecks and interchangeable support for multiple communication infrastructures, from shared memory to Internet-based communication services.

This paper is organized as follows. In Section 2 we sketch the motivations for our study, and some comments on existing implementations. In Section 3 we present the design and architecture of the ARTÌS middleware, with some emphasis on the implementation guidelines. Section 4 sketches some simulation testbeds and case studies we defined to test the ARTÌS implementation. Section 5 presents conclusions and future work.

## 2 Motivation and Preliminary Discussion

In the following we give a short description of the motivations for this study, and the comments that have been originated by the analysis of existing middleware implementations of the HLA-based distributed simulation middleware.

Model components' reuse is considered a relevant issue to be supported in designing a new simulation system. On the other hand, model components may be confidential information on behalf of the companies that designed them. The owner companies could be interested, under a commercial viewpoint, in allowing their models to be embedded as "black box" components for evaluating the integration analysis and compliance with other solutions. The open model-component source code could introduce the risk to reveal the confidential know-how in the component design solutions. A way to overcome this problem would be given by supporting the model component simulation in distributed way, and more specifically, over execution units local to the owner company domain. Distributed model components would simply export their interfaces and interactions (i.e., messages) with the simulation middleware and runtime infrastructure (RTI) implementing a distributed simulation. This scenario would require that a general network communication infrastructure (e.g., the Internet) would support the message passing communication between distributed model components of a parallel or distributed simulation. This is the reason why we conceptualized a distributed simulation that could be performed over TCP/IP or Reliable-UDP/IP network protocol stacks, like in web-based simulations. Under the latter assumption, the distributed simulation platform is intended as a way to interconnect protected objects, instead of a way to improve the simulation speedup. Other possible killer applications for such a distributed simulation middleware design would be the distributed Internet Gaming applications, gaining an even growing interest nowadays. The opportune design of the simulation framework, based on the exploitation of the communication scenario heterogeneities and characteristics, could improve the overall simulation performance of distributed and remotely executed processes.

The most natural and efficient execution scenarios for parallel and distributed simulations often involve shared memory (SHM) and/or local area networks (LAN) as the infrastructures supporting inter-process communication and synchronization services. Nowadays, it is even more frequent the adoption of networked cluster of PCs, in the place of shared-memory or tightly-coupled multiprocessors, as the execution units of the distributed simulation, primarily for cost reasons. The aforementioned motivations for model reuse and wide distribution of the model component execution is demanding for a generalized support for inter-process communication, up to the Internet-based services. It is self-evident how the increase of the communication cost (i.e., the communication latency of local and wide area network-based communication) would result in a reduction of the simulation speed. In other words, any reduction of the communication-time cost would translate in more efficiency of the simulation processes. The communication-time reduction could play a fundamental role in determining the communication and synchronization overheads between the distributed model components.

As remarked in [2], a distributed simulation approach is not always guaranteed to gain in performance with respect to a sequential simulation. The problem with the distributed simulation arises when a high degree of interactions is required in dynamic environments, mapping on distributed synchronization and inter-process communication. The basic solution to distribute the events information among interacting distributed components was the information flooding (broadcast) solution. This solution is quite immediate to implement over a generalized communication platform, but it was also originating the communication bottleneck effect for the distributed simulation. It was immediately clear that a reduction of communication would have been needed, by following two possible approaches: model aggregation and communication filtering. Model aggregation incarnates the idea to cluster interacting objects, by exploiting a degree of locality of communications that translates in a lower communication load than the one obtained in flat broadcast (that is, communication flooding) systems. Model aggregation can be performed by simplifying the model, or by maintaining the model detail. Solutions based on model simplification have been proposed, based on relaxation and overhead elimination, by dynamically introducing higher levels of abstraction and merging in system sub-models [2,19,23]. These solutions allow a reduction of communication since the messages are filtered on the basis of the level of abstraction considered. Solutions preserving full model-detail have been proposed by dynamically filtering the event- and state-information dissemination. Examples can be found [2], based on interest management groups [23], responsibility domains, spheres of influence, multicast group allocation, data distribution management [21,5], grid distribution and routing spaces [21,5,7], model and management partitioning [3]. These approaches rely on the reduction of communication obtained when the update of an event- or state-information (e.g., event and/or anti-message) does not need to be flooded to the whole system, but is simply propagated to all the causally-dependent components. This is the basis of publishing/subscribing mechanisms for sharing state-information and event-notifications between causally dependent components [21,5,20]. The solution provided in order to dynamically filter the communication among distributed objects was the ancestor of the Data Distribution Management (DDM) concept realized and implemented in HLA-based solutions [11].

The High Level Architecture (HLA) is a middleware implementation based on recently approved standard (IEEE 1516) dealing with component-oriented distributed simulation [11]. The HLA defines rules and interfaces allowing for heterogeneous components' interoperability in distributed simulation. The definition of distributed model components (formally known as federates) with standard management APIs brings to a high degree of model re-usability. The HLA standard defines APIs for the communication and synchronization tasks among federates. The distributed simulation is supported by a runtime middleware (RTI). The RTI is mainly responsible for providing a general support for time management, distributed objects' interaction, attributes' ownership and many other optimistic and conservative event-management policies. The IEEE 1516 standard has gained a good popularity but still has not reached the planned diffusion. The main reasons are the complex definitions and design work required to modelers. On the other hand, the preliminary implementations of distributed simulation middleware solutions and architectures were too complex, too slow and required a great startup time to achieve the expected results. Specifically, since its definition, the IEEE 1516 Standard has been criticized about its structure and its effective ability to manage really complex and dynamic models [6]. By analyzing the existing RTI implementations, to the best of our knowledge, few currently available middleware solutions have been designed with some emphasis on the adaptive exploitation of the communication infrastructure heterogeneity which may be characterizing the distributed simulation-execution scenario. More specifically, the Georgia-tech RTI-kit [21] implementation has been realized by introducing some elasticity and optimization in the exploitation of the shared memory execution-system architecture, whereas many other implementations still rely on UDP or TCP socket-based interprocess communication even on a single execution unit. It is worth noting that rare implementations provided the source code to users, allowing them to configure the middleware on the basis of the user needs and execution-system architecture.

The support for heterogeneous communication services and architectures should be considered as a design principle in the implementation of a distributed simulation middleware. Moreover, the adaptive optimization and management of the middleware communication layer realized over heterogeneous network architectures, technologies and services should be considered both in the initialization phase, and at runtime, in a distributed simulation process. Our ARTÌS implementation aims to be Open Source, and to provide an elastic, easy to configure adaptation of the communication layer to the execution system.

## 3 The ARTÌS Middleware

The HLA implementation criticisms and the lack of Open Source implementations are the main motivations behind the design and implementation of ARTÌS (Advanced RTI System). The main purpose of ARTÌS is the efficient support of complex simulations in a parallel or distributed environment.

The ARTÌS implementation follows a component-based design, that should result in a quite extendible middleware. The solutions proposed for time management and synchronization in distributed simulations have been widely analyzed and discussed. Currently, ARTÌS supports the conservative time management based on both the time-stepped approach, and the Chandy-Misra-Bryant algorithm. In the near future we have planned to extend ARTÌS to support optimistic time management algorithms. The initial choice to support the conservative approach was a speculation on the highly unpredictable characteristics of our target models of interest [2], which would have led to frequent rollbacks. Anyway, we plan to investigate this assumption, and compare the optimistic and conservative approaches as a future work.

In ARTÌS, design optimizations have been applied to adapt adequate protocols for synchronization and communication in Local Area Network (LAN) or Shared Memory (SHM) multiprocessor architectures. In our vision the communication and synchronization middleware should be adaptive and user-transparent about all optimizations required to improve performances. The presence of a shared memory for the communication among parallel or distributed Logical Processes (LPs) offers the advantage of low latency, and reliable communication mechanism. Interactions are modeled as read and write operations performed in shared memory, within the address space of logical processes. A memory access is faster than a network communication, but the shared memory itself is not sufficient to alleviate the distributed communication problem. To take advantage of the shared memory architecture, concurrent accesses to memory require strict synchronization and mutual exclusion, together with deadlock avoidance and distributed control.

Figure 1 shows the structure of the ARTÌS middleware. ARTÌS is composed by a set of logical modules organized in a stack-based architecture. The communication layer is located at the bottom of the middleware architecture, and it is composed by a set of different communication modules. The ARTÌS middleware is able of adaptively select the best interaction module with respect to the dynamic allocation of Logical Processes (LPs) in the execution environment. The current scheme adopts an incremental straightforward policy: given a set of LPs

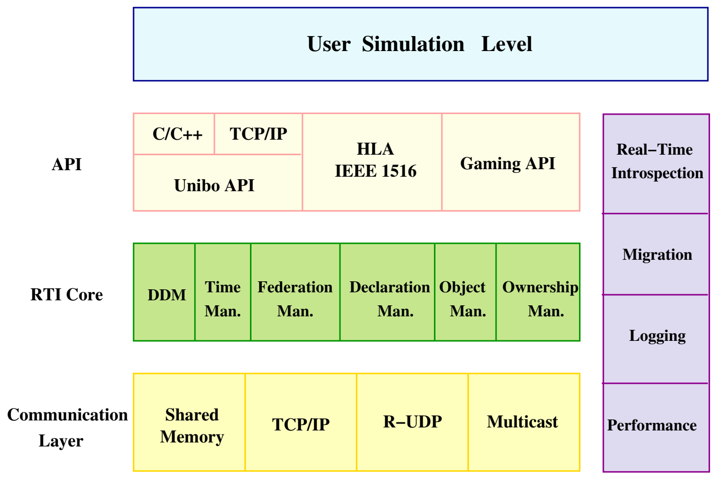

**Fig. 1.** The ARTÌS modules' structure and layered architecture

on the same physical host, such processes always communicate and synchronize via shared memory. To implement these services we have designed, implemented and tested many different solutions. The first implementation was based on Inter Process Communication (IPC) semaphores and locks. This solution was immediately rejected both for performance reasons (semaphore and locks introduce not negligible latency), and for scalability reasons (since the number of semaphores that could be instantiated in a system is limited and statically controlled in the operating system kernel). Among other possible solutions (e.g., we also considered busy-waiting) the current ARTÌS synchronization module works with "wait on signals" and a limited set of temporized spin-locks. This solution has demonstrated very low latency and limited CPU overhead, and it is really noteworthy for good performances obtained in multi-CPU systems, good scalability and also because it does not require any reconfiguration at the operating system kernel level.

In ARTÌS, two or more LPs located on different hosts (i.e., no shared memory available), on the same local area network segment, communicate by using a light Reliable-UDP (R-UDP) transport protocol over the IP protocol. Ongoing activity is evaluating the use of raw sockets for R-UDP data segments directly encapsulated in MAC Ethernet frames (i.e., bypassing the IP layer). The drawback of this solution is that it could be adopted only within a common LAN segment technology. Two or more LPs located on Internet hosts rely on standard TCP/IP connections. Ongoing activity at this level is performed by considering the exploitation of reliable multicast-IP solutions.

The ARTÌS runtime (RTI) core is on the top of the communication layer. It is composed by a set of management modules, whose structure and roles have been inherited by a typical HLA-based simulation middleware, compliant with the IEEE 1516 Standard. The modules currently being under the implementation phase are: the Data Distribution Management (DDM) in charge of managing the dynamic subscription/distribution of event and data update messages, the Time Management module currently implementing the conservative, time-stepped distributed time management, the Federation Management, Declaration Management, Object Management and Ownership Management modules in charge of administrating all the remaining management issues accordingly with the standard rules and APIs.

The ARTÌS runtime core is bound to the user simulation layer by modular sets of application programming interfaces (APIs). Each API group was included in order to allow a full integration and compliance of many distributed model components with the ARTÌS middleware. The Standard API is implemented as the HLA IEEE 1516 interface: this will allow the integration of IEEE 1516 compliant models to the ARTÌS framework. Only a subset of the full Standard API is currently implemented in ARTÌS. The new set of APIs of ARTÌS is called the *University of Bologna APIs* (Unibo APIs). These APIs are currently designed and implemented to offer a simpler access to distributed simulation services than the Standard APIs. This would make easier and simpler for the modelers to create and instantiate a distributed simulation with ARTÌS, than with the Standard APIs. The programming handles of the Unibo APIs are currently provided for C/C++ language code. We planned also to include in ARTÌS an API set specific for Internet Gaming applications, whose design is still in preliminary phase.

Additional orthogonal modules are planned to be dedicated to other specific features, oriented to the adaptive runtime management of synchronization and communication overheads. As an example, a real-time introspection mechanism would be devoted to offer an internal representation of the middleware state while the simulation is running. Logging and Performance modules would support the user simulation with online traces, statistics and runtime data analysis. Previous research works shown that more efficient simulation of dynamic models can be obtained by introducing additional software components implementing distributed model entities migration [2]. The Migration module should be orthogonal in the middleware, with the target to reduce runtime communication overheads accordingly with the coordination supported with other peer modules. To this end, future work is planned to include in ARTÌS a dynamic model-entity migration support that we conceived and illustrated, as a prototype, for the HLA-based middleware on [2]. By defining a dynamic allocation of simulated model entities over different physical execution units we obtained speed-up improvements (and communication overheads reduction) that could be further optimized in ARTÌS, when supported by opportune coordination of the migration modules.

## 4 ARTÌS Test and Validation

The verification and validation test of the preliminary implementation of the ARTÌS framework has been executed over a set of distributed and heterogeneous model components implemented in C code. The tests performed were based on models of wireless networks' components. The wireless ad hoc networks models were mobile hosts with IEEE 802.11 network interfaces, and static sensor networks based on small-communication-range components. We are also realizing models and experiments to test distributed protocols over large, scale-free networks. All the simulation experiments performed were realized with several thousands model components, and with many logical processes distributed over local (i.e., shared memory) and distributed physical execution units connected by a Fast-Ethernet (100 Mbps) LAN segment. The set of execution units we used in the tests is composed by: several dual Xeon 2.8 GhZ with 2GB RAM, and one Quadral Xeon 1.4 GhZ with 2 GB RAM, all with Debian GNU/Linux OS with kernel version 2.6. All the tests performed shown the ARTÌS framework was correctly implemented, by supporting a scalable number of model components in a conservative distributed simulation. The tests of TCP/IP (i.e., Internet-like) communication scenario were executed over our department's LAN network. The performance comparison of the shared memory, R-UDP and TCP/IP implementation is beyond the scope of this paper and will be done as a future work. Anyway, it was clearly indicated by preliminary results that orders of magnitude differences in simulation performances can be obtained under the different approaches, and this would confirm that adaptive and runtime management realized by the ARTÌS framework to exploit models' and system dynamics, would result in relevant simulation performance gain and overheads reduction.

## 5 Conclusions and Future Work

In this paper we illustrated the motivation and preliminary design and implementation issues of a new distributed simulation middleware named Advanced RTI System (ARTÌS). The aim of the ARTÌS middleware is to support parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components. The ARTÌS design is oriented to support the model components heterogeneity, distribution and reuse, and to reduce the communication overheads, by increasing the simulation performances, scalability and speedup, in parallel and distributed simulation scenarios. The ARTÌS design and architecture was presented to illustrate how the reduction of the communication cost and the exploitation of heterogeneous execution platforms could be considered in the design and implementation of parallel and distributed simulation middlewares based on the IEEE 1516 (HLA) Standard. Future work includes the complete implementation of the ARTÌS APIs and management modules, the full investigation of the ARTÌS performance and scalability, the implementation of model component migration and dynamic adaptation primitives, the design of new model components libraries and the distributed Gaming APIs.

## References

1. Bagrodia, R., Meyer, R., Takai, M, Chen, Y., Zeng, Z., Martin, J., Song, H.Y.: Parsec: A Parallel Simulation Environment for Complex Systems. Computer **31** (1998) 77-85
2. Bononi, L., D'Angelo, G., Donatiello, L.: HLA-Based Adaptive Distributed Simulation of Wireless Mobile Systems. In: Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS 2003), San Diego, California. IEEE Computer Society (2003) 40–49
3. Boukerche, A., Fabbri, A.: Partitioning Parallel Simulation of Wireless Networks. In: Joines, J.A., Barton, R.R., Kang, K., Fishwick, P.A. (eds.): Proceedings of the 32nd Conference on Winter Simulation (WSC 2000), Orlando, Florida. Society for Computer Simulation International (2000) 1449-1457
4. Dahmann J.S., Fujimoto R.M., Weatherly, R.M.: The Department of Defense High Level Architecture. In: Andradóttir, S., Healy, K.J., Withers, D.H., Nelson, B.L. (eds.): Proceedings of the 29th Conference on Winter Simulation (WSC'97), Atlanta, Georgia. ACM (1997) 142–149
5. Dahmann, J.S., Fujimoto, R.M., Weatherly R. M.: The DoD High Level Architecture: An Update. In: Medeiros, D.J., Watson, E.F., Carson, J.S., Manivannan, M.S. (eds.): Proceedings of the 30th Conference on Winter Simulation (WSC'98), Washington, D.C. IEEE Computer Society (1998) 797–804
6. Davis, W.J., Moeller, G.L.: The High Level Architecture: Is There a Better Way? In: Farrington, P.A., Nembhard, H.B., Sturrock, D.T., Evans, G.W. (eds.): Proceedings of the 31st Conference on Winter Simulation (WSC'99), Phoenix, Arizona, Vol. 2. ACM (1999) 1595–1601
7. Deelman, E., Szymanski, B.K.: Dynamic Load Balancing in Parallel Discrete Event Simulation for Spatially Explicit Problems. In: Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98), Banff, Alberta. IEEE Computer Society (1998) 46–53
8. Ferscha, A.: Parallel and Distributed Simulation of Discrete Event Systems. In: Zomaya, A.Y.H. (ed.): Handbook of Parallel and Distributed Computing. McGraw-Hill, New York (1996)
9. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. John Wiley & Sons, New York (2000)
10. Gan, B.P., Low, Y.H., Jain, S., Turner, S.J., Cai, W., Hsu, W.J., Huang, S.Y.: Load Balancing for Conservative Simulation on Shared Memory Multiprocessor Systems. In: Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000), Bologna, Italy. IEEE Computer Society (2000) 139–146
11. IEEE Std 1516-2000: IEEE Standard for Modeling and Simulation (M&S) High Level architecture (HLA) - Framework and Rules, - Federate Interface Specification, - Object Model Template (OMT) Specification, - IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP). (2000)
12. Kelly, O.E., Lai, J., Mandayam, N.B., Ogielski, A.T., Panchal, J., Yates, R.D.: Scalable Parallel Simulations of Wireless Networks with WiPPET: Modeling of Radio Propagation, Mobility and Protocols. Mobile Networks and Applications **5** (2000) 199–208
13. Liu, W.W., Chiang, C.-C., Wu, H.-K., Jha, V., Gerla, M., Bagrodia, R.L.: Parallel Simulation Environment for Mobile Wireless Networks. In: Proceedings of the 28th Conference on Winter Simulation (WSC'96), Coronado, California. ACM (1996) 605–612

14. NS-2: The Network Simulator. (2004) `http://www.isi.edu/nsman/ns/`
15. OMNeT++: Discrete Event Simulation Environment. (2004) `http://www.omnetpp.org`
16. Open SystemC Initiative. (2004) `http://www.systemc.org`
17. OpNet Simulation Platform. (2004) `http://www.opnet.com`
18. PERF Project. Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools, Italian MIUR-FIRB. (2002) `http://www.perf.it`
19. Rao, D.M., Wilsey, P.A.: Parallel Co-simulation of Conventional and Active Networks. In: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2000), San Francisco, California. IEEE Computer Society (2000) 291–298
20. Riley, G.F., Fujimoto, R.M., Ammar, M.H.: A Generic Framework for Parallelization of Network Simulations. In: Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99), College Park, Maryland. IEEE Computer Society (1999) 128–137
21. RTI-Kit. Parallel and Distributed Simulation, Georgia Institute of Technology Research. (2003) `http://www.cc.gatech.edu/computing/pads/software.html`
22. Short, J., Bagrodia, R., Kleinrock, L.: Mobile Wireless Network System Simulation. Wireless Networks **1** (1995) 451–467
23. Som, T.K., Sargent, R.G.: Model Structure and Load Balancing in Optimistic Parallel Discrete Event Simulation. In: Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000), Bologna, Italy. IEEE Computer Society (2000) 147–154
24. Vee, V.-Y., Hsu, W.-J.: Locality-Preserving Load-Balancing Mechanisms for Synchronous Simulations on Shared-Memory Multiprocessors. In: Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000), Bologna, Italy. IEEE Computer Society (2000) 131–138
25. Zeng, X., Bagrodia, R., Gerla, M.: GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks. In: Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98), Banff, Canada. IEEE Computer Society (1998) 154–161