Asian Journal of Information Technology 6 (7): 817-824, 2007
ISSN: 1682-3915

PADS Advances and Advanced RTI System

[1]R. Latha and [2]S.P. Rajagopalan
[1]Department of Computer Applications, St. Peter's Engineering College Avadi, Chennai-600 054, India
[2]Department of Computer Science and Engineering, Dr. MGR University, Chennai-600 095, India

Abstract: The parallel and distributed simulation has been proved to reduce the time required to complete the simulation of some scenarios, improve the code reusability, address the requests for fault-tolerance and support the spatially located architectures. This study on Parallel and Distributed Simulation systems reviews some of the traditional synchronization techniques and presents some recent advances. This study presents a new simulation middleware named Advanced RTI System (ART`IS), designed to support the parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components.

Key words: Middleware, Application Programming Interfaces (API), high level architecture, runtime infrastructure, Data Distribution Management (DDM), lookahead extraction

INTRODUCTION

A simulation is a system that represents or emulated the behavior of another system over time. In a computer simulation the system doing the emulating is a computer program (Fujimotto, 2000). In a very simplified form a computer simulation can be seen as a set of evolving state variables. The evolution of state variables over time can be managed with different timings. Time flow mechanisms are continuous or discrete.

The simulation advance of time can be time stepped or event-driven, in the first case the simulation state is updated every a fixed amount of time (Fig. 1), in the last only in presence of events (a change happens in the simulation) the state variables are updated (Fig. 2). The state variables evolution and the events computation can be managed in a sequential monolithic approach or in a decentralized way. If the simulation is executed by a set of Physical Execution Units (PEU) interconnected by a low latency network (i.e., shared memory multiprocessor), it is a parallel simulation. Conversely in presence of a high latency network (i.e., LAN, WAN, Internet) it is a distributed simulation. The physical system can be viewed as a collection of physical processes (executed on a set of PEUs) that interact in some fashion with each physical process being modeled by a Logical Process (LP) (Fujimotto, 2000) Each LP is usually executed by a single PEU (Fig. 3); the interactions are performed exchanging time stamped messages.

Parallel And Distributed Simulation (PADS) finds relevance in civilian applications as well as non-civilian applications. It deals with ways of using multiple processors in a single simulation. Research work has been done in the field of synchronization and Data

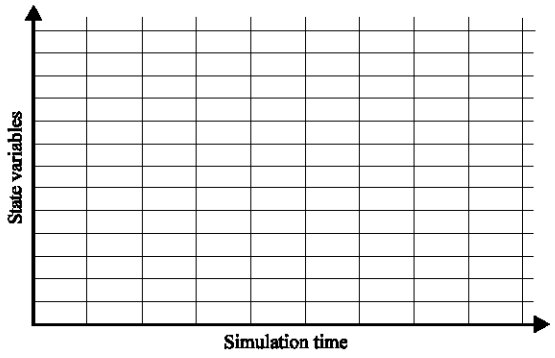

Fig. 1: Time-stepped approach-fixed amount of time

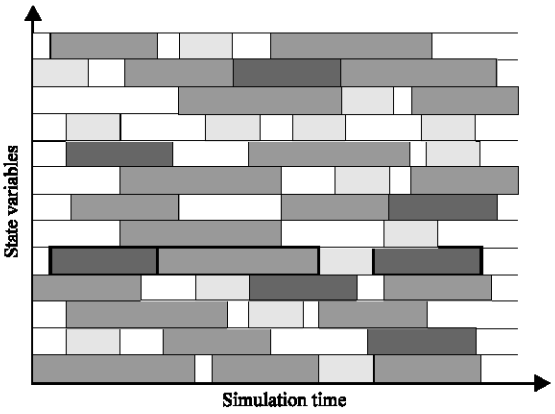

Fig. 2: Event-driven approach-change of states

Corresponding Author: R. Latha, Department of Computer Applications, St. Peter's Engineering College Avadi, Chennai-600 054, India

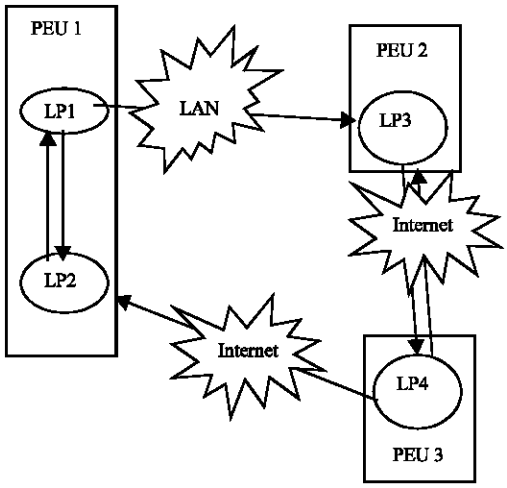

Fig. 3: A set of interacting logical processes and physical execution units

Distribution Management (DDM) to reduce the communication overheads, increasing the PADS speed-up with respect to the monolithic approach. The need for a common definition for distributed simulation has been satisfied with the High Level Architecture (IEEE1516) standard approval.

During the last decades, a lot of simulation tools has been developed and deployed: Some are proprietary and others are open. A large set of communities has shared knowledge and experiences directly or indirectly related to the PADS field. To overcome the problem of simulated entities migration and to improve the speedup of massive and complex systems, ART'IS middleware has been designed based on the analysis and evaluation of existing HLA based RTI implementation for an extensive work on performances involving dynamic adaptation of the interprocess communication layer and concurrent replication of PADS.

Recent emergence of new application demands, techniques and hardware platforms are resulting in enhancements to traditional techniques and formulation of newer techniques. Here a brief overview of traditional techniques followed by a presentation of some of the recent advances is presented. This is followed by the internal logical architecture and implementation of ART`IS, a new middleware for parallel and distributed simulation.

BASIC TECHNIQUES

Parallel and distributed simulation approaches can be broadly categorized into spatial parallel and time parallel schemes. In spatial parallel schemes, the application is partitioned along spatial dimensions underlying the application's models (e.g., three dimensional grid cells in physical system simulation, or computers and network routers in Internet simulations). Time parallel schemes partition the simulation along its time dimension (e.g., regular time intervals along the simulation time axis).

Time notations: In simulations, there are three distinct notions of time. The first is the physical time, which is the time in the physical system that is being modeled, e.g., 10-11pm on Dec 2000. The second is the simulation time, which is a representation of the physical time for the purposes of simulation (e.g., number of seconds since 10pm of Dec 2000, represented in floating point values in the range 0.3600 corresponding to the simulated time period of the physical time). Finally, the wallclock time is the elapsed real time during execution of the simulation, as measured by a hardware clock (e.g., number of milliseconds of computer time during execution). For each, the notions of time axis and time instant can be defined-time axis is the totally ordered set of time instants along the corresponding timeline.

Events and event orderings: An event is an indication of an update to simulation system state at a specific simulation time instant specifying a timestamp. When events are exchanged among processors, their delivery at the receiving processors needs to be carefully coordinated at runtime. Two commonly used orderings are: Receive-order and timestamp-order.

In Receive-Ordered delivery (RO), events from other processors are delivered to the receiving processors as and when the events arrive at the receiving processor. In contrast, in Timestamp-Ordered delivery (TSO), events are guaranteed to be delivered in non-decreasing order of their timestamps. The rationale behind timestamp-ordered processing is that it permits the models to be accurately simulated, such that events are processed in the same order as their corresponding actions in the physical system.

Basic synchronization approaches: In parallel simulation, the commonly used four approaches are: Conservative approach ensures safe timestamp-ordered processing of simulation events within each LP. Optimistic approach avoids blocked waiting by optimistically processing the events beyond the lookahead window. Relaxed synchronization approach relaxes the constraint that events be strictly processed in time stamp order. Combined synchronization approach combines elements of the previous three. Eg.HLA.

Data Distribution Management (DDM): A distributed simulation can be composed by thousands of simulated entities managed by a set of PEUs, interconnected by different kinds of networks. Optimized information spreading would require to exactly determining, what data set has to be sent to each PEUs, minimizing the amount of data sent and taking advantage of eventual information duplicates. The DDM as defined in the HLA is based on Class-Based Data Distribution and Routing Spaces (Fujimotto, 2000).

THE HIGH LEVEL ARCHITECTURE

In the HLA, an integrated execution of simulations is called a federation. Individual simulators participating in a federation are called federates. Federates (processor) can be of different types: Pure software simulators such as computer generated forces, human-in-the-loop simulators such as virtual simulators, or live components such as instrumented weapon systems. The time synchronization module of the HLA, called the Time Management (TM) has been built on insights from PADS research.

Interoperability challenge: The HLA's TM services address two important components: Overall event processing order by each federate and Synchronized event delivery to each federate. While enabling event processing order and synchronized event delivery, all in a single encompassing standard framework, the HLA needs to accommodate a large variety of individual types of simulators.

Synchronization services: The HLA TM module provides a clear interface that each federate must invoke in order to synchronize with other processors. The three most commonly used services are: Time Advance Request (TAR), Next Event Request (NER) and Flush Queue Request (FQR). A federate undertaking fixed time increments in simulation time can use TAR (T) to unconditionally advance its simulation time to T. Events from other federates that arrive with timestamps less than T are all delivered to this federate before the runtime permits the federate to advance to time T. Time-stepped parallel simulations typically use this service to coordinate their time steps across processors.

A federate operating under a discrete event paradigm invokes the NER (T) primitive to conditionally advance its simulation time to T. If the runtime discovers that other events with timestamps less than T are generated by other processors, the earliest of those events are delivered to the federate and time is advanced only to their timestamp. A federate equipped to execute its events in optimistic mode (i.e., ahead of receiving guarantees of correctness) can invoke FQR (T) to force the runtime to release any and all events that it currently has, irrespective of their timestamps. The runtime utilizes the supplied timestamp T to compute absolute global time advances.

Computing LBTS: A fundamental role of a TM implementation is in computing a quantity known as Lower Bound on incoming Time Stamps (LBTS). At each federate, the LBTS value specifies a guarantee on the least timestamp on any future incoming event. In other words, no event will ever arrive at that federate with a timestamp smaller than LBTS. In order to compute the LBTS value at each federate, a distributed algorithm is required that exchanges messages to coordinate the LBTS computation without deadlocks, live-locks or undue performance degradation. A close cousin to the LBTS computation is Global Virtual Time (GVT) computation in optimistic simulation and flush barrier algorithms.

Serving synchronization requests: The RTI internally maintains a priority queue of TSO events, ordered by their timestamps. When a federate invokes TAR (T), the RTI first examines if LBTS is >T. If so, the request is trivially satisfied-the RTI delivers all events from its TSO queue whose timestamps are $\leq$T and then issues a TAG (T). If T > LBTS, then the RTI initiates a new distributed LBTS computation (if one is not already in progress). The lesser of T and minimum timestamp in TSO queue is used as this federates contribution in the LBTS computation.

Recent advances: Recent advances are driven by new challenges raised by high-performance needs of applications such as large-scale telecommunication network simulation (e.g., Internet-scale TCP/IP simulations), hardware/VLSI system simulations.

Mixed synchronization: When large application scenarios are considered, it often is the case that the models are heterogeneous in nature, with varying amounts of computational loads across spatial dimensions and varying levels of modeling. In contrast to the heterogeneity needs, the traditional method of prevalence in building parallel simulation systems is to build a system specifically for one synchronization method or algorithm. A set of systems are being constructed to alleviate this problem when the application needs to accommodate multiple synchronization types and be resilient to future changes to parts of its models.

Lookahead extraction: In purely conservative parallel simulations, the single most important parameter is the lookahead value. A small lookahead can have a

large detrimental effect on overall runtime performance. In recent applications, the models are prohibitively complex to apply optimistic simulation techniques but are also very tightly-coupled, exhibiting very small lookahead between LPs. Wireless network simulation is one such application (e.g., predicting the performance of IEEE 802.11 Wi-Fi net-works). Static lookahead in such applications is typically very small, given by the time for electromagnetic signal propagation (at close to speed of light) over short distances. When coupled with the need to use such simulations in real-time context (for network emulation), the gain from improving the lookahead even by a small amount can be quite significant. Moving from purely static estimates of lookahead, the new techniques examine the LP inter-dependency structures dynamically at simulation run-time and manage to extract values of lookahead that are higher than those estimated statically.

Critical channel traversal: Another method of improving performance of conservative simulations focuses on scheduling alternatives. In the most common event scheduling approach, events from the future event list are executed in an earliest-timestamp-first manner. Newer scheduling algorithms such as Critical Channel Traversal are designed for exposing and taking advantage of inherent locality of event execution in conservative simulations. Benefits include better cache performance due to immediate reuse of event buffers across causal chain of events and improved spatial locality due to reflection of LP inter-dependence on event processing patterns. Tasks, closely related LPs are then used as the primary scheduling units and LPs within a task form secondary scheduling units. When LPs within a task interact among themselves more heavily than with other LPs outside that task, locality is enhanced and event scheduling overheads are decreased.

Optimizations for efficient rollback: While lookahead extraction is the bane of conservative parallel simulations, rollback support mechanism is its counterpart in optimistic parallel simulations. For a long time, checkpointing based approaches were the dominant form of enabling rolling back computations.

State saving for rollback: Checkpointing or state-saving methods make a copy of the state variables of LPs before the variables are overwritten by optimistic event computations. If and when those event computations have to be negated, state variable values are fetched from the checkpoint log and overwritten, thereby restoring the LP state to their correct values corresponding to the fault point. The most commonly used state-saving technique is Copy State-Saving (CSS), in which a copy of the entire state is made before an event is executed. In CSS, the state is saved every time an event is processed. A variation of copy state-saving is called Periodic State Saving (PSS). PSS is a generalized technique in which state is saved only periodically (every p^{th} event). The former set of events can be rolled back easily by restoring the state to the saved values. The latter set of events needs special treatment, since they do not have saved state. The state restoration for these events is achieved by starting with a past processed event that does have saved state and then re-executing the sequence of events from that past event to the event just before the rolled back event. Incremental State-Saving (ISS) is an alternative state saving technique in which only the modified portions are saved just before modification. Modifications to state are logged as pairs of address value pairs and stored in a log array for each LP. Optimistic simulators can permit CSS, PSS and ISS to be used simultaneously instead of mutually exclusively.

Reverse computation: As an alternative in Reverse Computation (RC) approach, a perfect inverse of event computation is invoked that serves to recover the modified state values to their original values instead of saving the values of state variables for rollback support. The RC approach has been successfully applied in different application domains, such as telecommunication network simulations (Carothers *et al*., 1999; Garrett *et al*., 2003) and physical system simulations (Tang *et al*., 2005). Figure 4 depicts the relation between PSS and RC using a generic snapshot of execution at which rollback is initiated. The rightmost vertical bar represents the latest optimistically executed event. The light vertical bar in the middle represents the point, to which computation needs to be rolled back.

The left-most dark vertical line indicates the most recent periodic checkpoint before the rollback point. If RC is used, reverse execution is invoked on all events from left to middle. If PSS is used, coast forwarding computation (event re-execution to regenerate state) is invoked from left to middle. Clearly, the efficiency of RC or PSS depends on expected distance of left or right points, respectively, from the rollback point. However, a crucial difference between RC and PSS is that the state size is independent of the computing platform for RC, whereas it depends on sizes of data types in PSS. Also, PSS has the drawback that not all events from time points earlier than GVT/LBTS can be reclaimed for reuse (i.e., cannot be fossil-collected). Those events that fall between the periodic checkpoint time and GVT/LBTS need to be retained for coast-forwarding.

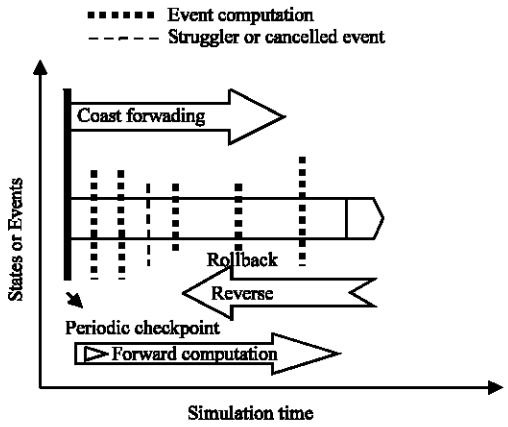

Fig. 4: Comparision of rollback operation using reverse computation vs.periodic state saving

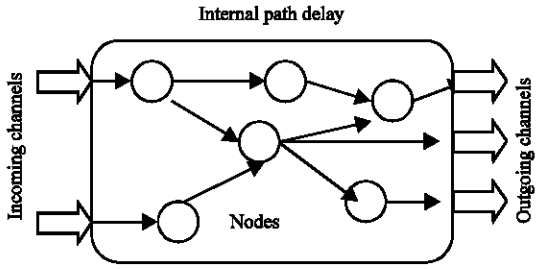

Fig. 5: Illustration of internal lookahead to improve null message timestamps

Intra-processor lookahead: Figure 5 illustrates any events between incoming channels on the left and outgoing channels on the right incur a minimum internal path delay which can be used to improve outgoing null message timestamps. In large-scale application scenarios, each processor typically hosts many LPs. Traditional conservative parallel simulation maps network links crossing processor boundaries to in/out channels in null-message algorithms and uses the minimum among all transmission delays as the lookahead values on all those out channels, irrespective of delays added by internal network nodes. An internal lookahead can be defined, equal to the minimum among all shortest paths between all pairs of in-and out-channels. This is easily computed to be the sum of transmission delays along the shortest paths from all in channels to all out channels, which can add up to much more than lookahead along one transmission link.

Fault tolerance and security: With the increase in the number of processors/federates participating in a large integrated simulation, the probability increases that one or more of the processors fail during simulation. Transparent techniques to sustain the progress of the simulation despite such crashes are a subject of recent interest. Schemes to incorporate such fault tolerance and/or continued optimistic executions are now being explored (Chen *et al*., 2006). Also, secure exchange of information across processors without compromising confidentiality of information contained in models is an important facet of interoperable integrated simulations. This concern for security of information exchange during simulation raises new challenges, such as runtime overhead for encrypted communication, which are beginning to be addressed only recently.

Art'is architecture: The Advanced RTI System (ART`IS) is designed to support parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components. The ART`IS middleware has been the basis for an extensive work on performances, involving dynamic adaptation of the interprocess communication layer and concurrent replication of parallel and distributed simulation.

**The art`is middleware:** The ART`IS 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 (Bononi *et al*., 2003). 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) and more targeted system models (e.g., computer networks). 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 computation parallelism.

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 speed-up. 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.

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. 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 speed-up, both in optimistic and conservative approaches. The efficient implementation of interprocess communication is required as a primary background issue, to overcome the possible communication bottleneck of parallel and distributed simulations. Middleware solutions based on the IEEE 1516 Standard and HLA have shown that the PADS 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.

Design overview: Model components' reuse is considered a relevant issue to be supported in designing a new simulation system. 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. Hence a distributed simulation is conceptualized that could be performed over TCP/IP or Reliable-UDP/IP network protocol stacks, like in web-based simulations. The most natural and efficient execution scenarios for PADS often involve Shared Memory (SHM) and/or LAN as the infrastructures supporting inter-process communication and synchronization services. The basic solution to distribute the events information among interacting distributed components was the information flooding (broadcast) solution. 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 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. 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.

The preliminary implementations of distributed simulation middleware solutions and architectures were often too complex, too slow and required a great startup time to achieve the expected results. The Georgia Tech RTI-kit 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.

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 ART`IS implementation aims to be Open Source and to provide an elastic, easy to configure adaptation of the communication layer to the execution system.

**Design of the ART`IS middleware:** The ART`IS 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, ART`IS supports the conservative time management based on both the time-stepped approach and the Chandy-Misra-Bryant algorithm (Chandy *et al*., 1981). The optimistic time management (Time Warp) is implemented and currently under validation.

In ART`IS, 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 my vision

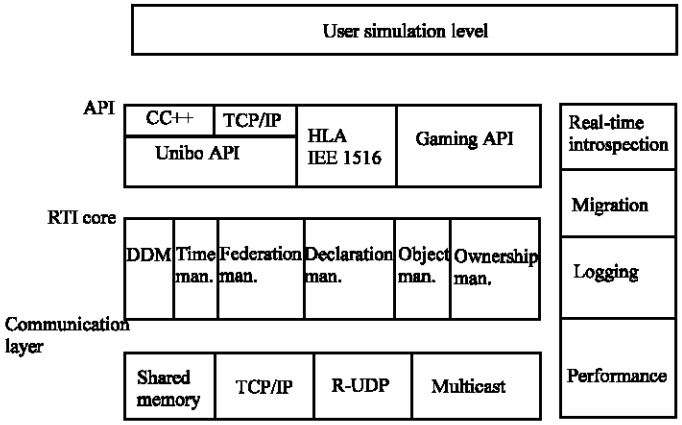

Fig. 6: Logical Architecture of the ARTIS middleware

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 6 shows the structure of the ART`IS middleware. ART`IS 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 ART`IS 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 on the same physical host, such processes always communicate and synchronize via shared memory.

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. Among other possible solutions (e.g. busy-waiting) the current ART`IS 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 doesn't require any reconfiguration at the operating system kernel level. In ART`IS, 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.

The ART`IS 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 DDM in charge of managing the dynamic subscription/distribution of event and data update messages, the TM module, 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 ART`IS 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 ART`IS middleware.

Additional orthogonal modules are planned to be dedicated to other specific features, oriented to the adaptive runtime management of synchronization and communication overheads. Logging and Performance

modules would support the user simulation with online traces, statistics and runtime data analysis. The migration module is orthogonal in the middleware, with the target to reduce runtime communication overheads accordingly with the coordination supported with other peer modules.

CONCLUSION

This study has illustrated the preliminary design and implementation issues of a new parallel and distributed simulation runtime. The goal is to build an efficient runtime, to support a wide set of APIs and to achieve IEEE standard compatibility. The runtime implementation is far from completeness but the middleware is sufficiently complete to support the simulation of massively populated wireless networks. In the future the current software can be optimized and implemented. A wide set of features has been externally developed and integrate them in the middleware. Our current efforts are toward the research of better algorithms for the management of the distributed event queues, further reducing the communication overhead and the support of High Performance Computing (HPC) architectures with hundreds of CPUs and non uniform shared memory.

REFERENCES

Bononi, L., G.M. Bracuto, D'Angelo and L. Donatiello, 2003. HLA-based Adaptive Distributed Simulation of Wireless Mobile Systems. In Proceedings of the 17th workshop on Parallel and distributed simulation. IEEE Computer Society.

Carothers, C. *et al.*, 1999. Efficient Optimistic Parallel Simulations using Reverse Computation. ACM. Trans. Modeling and Comput. Simulation, 9: 224-253.

Chandy, K.M. and J. Misra, 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. Commun. ACM., 24: 198-205.

Chen, D. *et al.*, 2006. A Framework for Robust HLA-based Distributed Simulations. International Workshop on Principles of Advanced and Distributed Simulation.

Fujimoto, R.M., 2000. Parallel and Distributed Simulation Systems. John Wiley and Sons, Inc., (1st Edn.), pp: 247-256.

Garrett, Y. *et al.*, 2003. Large-Scale TCP Models Using Optimistic Parallel Simulation. In: Proceedings of the 17th Workshop on Parallel and Distributed Simulation Computer Society.

Tang, Y. *et al.*, 2005. Optimistic Parallel Discrete Event Simulations of Physical Systems using Reverse Computation. Workshop on Principles of Advanced and Distributed Simulation.