An Object-Oriented Framework for Parallel Simulation of Ultra-large Communication Networks*

Dhananjai Madhava Rao and Philip A. Wilsey

University of Cincinnati, Cincinnati, OH, USA

Abstract. Communication networks have steadily increased in size and complexity to meet the growing demands of applications. Simulations have been used to model and analyze modern communication networks. Modeling and simulation of networks involving thousands of nodes is hard due to their sheer size and complexity. Complete models of the ultra-large networks need to be simulated in order to conduct in-depth studies of scalability and performance. Parallel simulation techniques need to be efficiently utilized to obtain optimal time versus resource tradeoffs. Due to the complexity of the system, it becomes critical that the design of such frameworks follow well established design principles such as object oriented (OO) design, so as to meet the diverse requirements of portability, maintainability, extensibility, and ease of use. This paper presents the issues involved in the design and implementation of an OO framework to enable parallel simulation of ultra-large communication networks. The OO techniques utilized in the design of the framework and the application program interfaces needed for model development are presented along with some experimental results.

1 Introduction

Communication networks coupled with the underlying hardware and software technologies have constantly increased in size and complexity to meet the ever growing demands of modern software applications. The Internet, a global data network, now interconnects more than 16 million nodes [1]. Modeling and studying today's networks with their complex interactions is a challenging task [1,2]. Simulation has been employed to aid the study and analysis of communication networks [2]. Parallel simulation techniques have been employed in large simulations to meet the resource requirements and time constraints. The network models are critical components of simulation analyses that should reflect the actual network sizes in order to ensure that crucial scalability issues do not dominate during validation of simulation results [1]. Many networking techniques that work fine for small networks of tens or hundreds of computers may become impractical when the network sizes grow [1]. Events that are rare or that do not

* Support for this work was provided in part by the Defense Advanced Research Projects Agency under contract DABT63–96–C–0055.

S. Matsuoka et al. (Eds.): ISCOPE'99, LNCS 1732, pp. 37–48, 1999.
© Springer-Verlag Berlin Heidelberg 1999

even occur in toy models may be common in actual networks under study [1]. Since today's networks involve a large number of computers ranging from a few thousands to a few million nodes, modeling and simulating such ultra-large networks is necessary.

Parallel simulation of ultra-large networks involves complex interactions between various components and places high demands on system resources — especially system memory. Hence, the data structures used in the system must be optimally designed with robust interfaces to enable efficient exchange of control and data in the distributed system [3]. To ease modeling, verification, simulation, and validation of ultra-large networks, well defined design principles such as object oriented (OO) design must be employed. In order to meet these diverse needs, an OO framework for simulation of ultra-large communication networks was developed. The framework is built around WARPED [3], an optimistic parallel simulation kernel. The framework is implemented in C++ and utilizes the OO techniques of inheritance, virtual functions, and overloading to develop efficient and robust interfaces.

The issues involved in the design and implementation of the framework along with a compendium of OO techniques used are presented in this paper. In Sect. 2 some of the related work in large scale network simulations are presented. Section 3 contains a brief description of WARPED. A detailed description of the framework and the application program interface (API) along with the OO techniques employed in their design and implementation are presented in Sect. 4. Some of the experiments conducted using the framework are presented in Sect. 5. Section 6 contains some concluding remarks.

2 Related Work

Simulation of large scale network models has received considerable attention in the past. Various combination of techniques have been used to improve the capacity and efficiency of large scale network simulations. Huag *et al* present a novel technique to selectively abstract details of the network models and to enhance performance of large simulations [4]. Their technique involves modification of the network models in order to achieve abstraction [4]. Premore and Nicol present issues involved in development of parallel models in order to improve performance [5]. In their work, they convert source codes developed for *ns*, a sequential simulator to equivalent descriptions in Telecommunications Description language (TeD) to enable parallel simulation [5]. Coupled with meta languages (such as TeD) [5], parallel network libraries and techniques to transparently parallelize sequential simulations have been employed [4]. Relaxation and even elimination of synchronization, a large overhead in parallel simulations, has been explored [6,7]. The relaxation techniques attempt to improve performance at the cost of loss in accuracy of the simulation results [6]. Fall exploits a combination of simulation and emulation in order to study models with large real world networks [8]. This method involves real time processing overheads and necessitates detailed model development.

In this paper, we present a technique for collating similar object descriptions to reduce and regulate the memory consumptions of the model and the simulation kernel. OO techniques have been employed to provide a robust API to ease model development and insulate the users from the underlying details. The API is similar to WARPED, the underlying simulation kernel. This enables other networking models, such as active networking models [9], to seamlessly interoperate with the framework. The paper also demonstrates effectiveness of the framework to enable ultra-large network simulations by providing experimental results.

3 Background

The framework for ultra-large networks simulation is built around the WARPED simulation kernel. WARPED [3] is an optimistic parallel discrete event simulator and uses the Time Warp mechanism for distributed synchronization. In WARPED, the logical processes (LPs) that represent the physical processes being modeled are placed into groups called "clusters". The clusters represent the operating system level parallel processes constituting the simulation. LPs on the same cluster directly communicate with each other without the intervention of the messaging system. This technique enables sharing of events between LPs, which considerably reduces memory overheads. Communication across cluster boundaries is achieved using MPI. LPs within a cluster operate as classical Time Warp processes; even though they are grouped together, they are not required to operate in time lockstep. A periodic garbage collection technique based on Global Virtual Time (GVT) is used. WARPED presents a simple and robust OO application program interface (API). Control is exchanged between the application and the simulation kernel through cooperative use of function calls. Further details on the working of WARPED and information on its API are available in the literature [3].

4 The Ultra-large Scale Simulation Framework (USSF)

The ultra-large scale simulation framework was developed to ease modeling and simulation of large communication networks. As shown in Fig. 1, the primary input to the framework is the topology to be simulated. The syntax and semantics of the input topology is defined by the Topology Specification Language (TSL), which provides simple and effective techniques to specify hierarchical topologies [9]. The topology is parsed into an OO Intermediate Format (TSL-IF). Static analysis is performed on the intermediate format to extract and collate common object definitions. The analyzed TSL-IF is then used to generate an optimal simulatable network topology. The current implementation of USSF, in conjunction with WARPED and the generated code, is in C++. The generated topology includes code to instantiate the necessary user defined modules that provide descriptions for the components in the topology. The generated code is

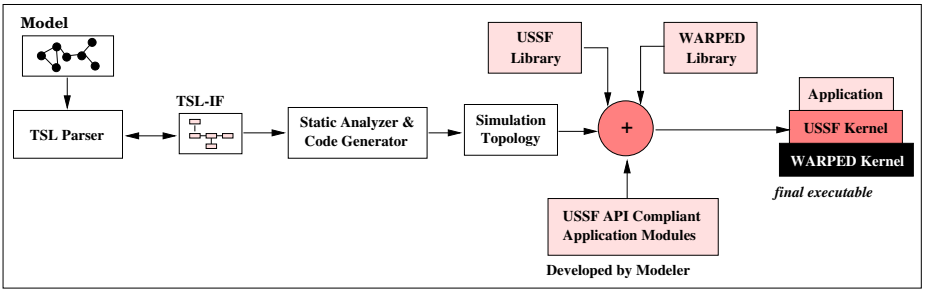


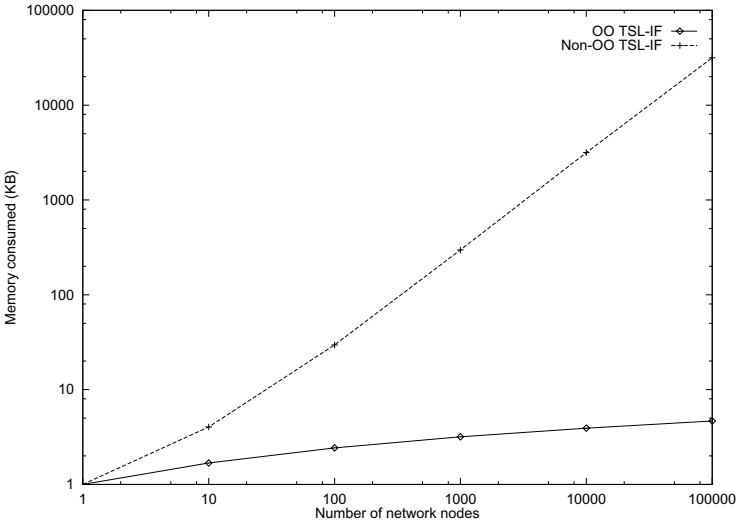
Fig. 1. Overview of USSF

compiled along with the USSF library, the WARPED library, and the application program modules to obtain the final simulation executable. The following sections describe the various components in detail.

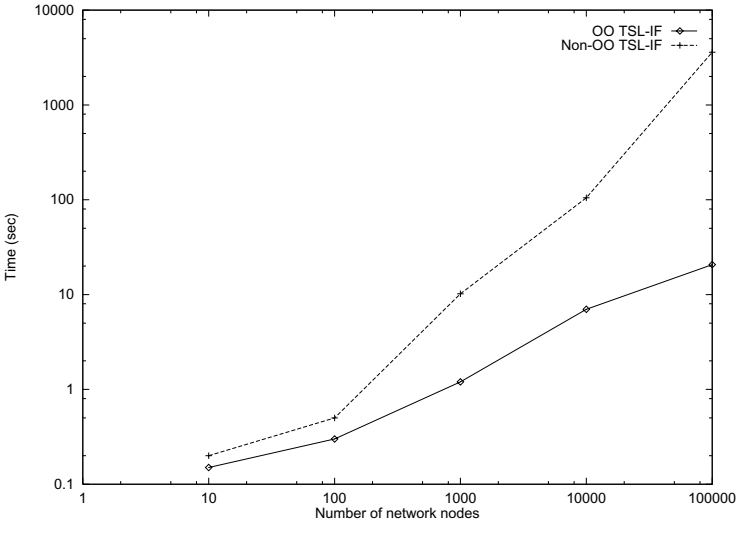
4.1 Topology Specification Language (TSL)

The topology of the network to be simulated is provided to the framework in Topology Specification Language (TSL) [9] syntax. A TSL specification consists of a set of topology specifications. Each topology specification consists of three main sections, namely; (i) the *object definition section* that contains the details of the modules to be used to simulate the topology; (ii) the *object instantiation section* that specifies the various nodes constituting the topology; and (iii) the *netlist section* that defines the interconnectivity between the various instantiated components. An optional *label* may be associated with each topology. The label may be used as an object definition in subsequent topology specifications to nest a topology within another. In other words, the labels, when used to instantiate an object, result in the complete topology associated with the label to be embedded within the instantiating topology. Using this technique, a simple sub-net consisting of merely ten nodes can be recursively used to construct a network with six levels of hierarchy to specify a million (10^6) node network.

The input topology configuration is parsed by a TSL-Parser into an OO TSL Intermediate Format (TSL-IF). The current implementation of the TSL-Parser is built using the Purdue Compiler Construction Tool Set (PCCTS) [10]. TSL-IF is designed to provide efficient access to related data from the various TSL sections. Each component of the grammar is represented by a corresponding TSL-IF node (or object). Every node in TSL-IF is built from the same basic building block; in other words, every node is derived from the same base class. The intermediate format is composed by filling in the appropriate references to the different nodes generated by the parser. Since composition is achieved via base class references, each node can refer to another node or even a sub network. This mechanism provides an efficient data structure for representing and analyzing hierarchical networks.



Memory consumption



Time for parsing

Fig. 2. Quantitative comparison between object-oriented (OO) and non-OO techniques used to construct TSL-IF

Prior to such an OO implementation of the IF, references to sub networks were replaced by their definitions; that is, "elaboration" was performed by

cloning (or duplicating) the sub networks every time nested topologies were encountered during parsing. Every node and sub-sub network were recursively cloned, new netlists were constructed, and merged with the outer topology. The elaboration is necessary in order to enable static optimizations and code-generation. This technique provided a simple mechanism to elaborate (or flatten) nested topologies and worked fine for small networks. As the network size increased, the time and memory requirements grew exponentially. In order to circumvent this bottleneck TSL-IF was developed. A quantitative comparison of the the two techniques is shown in Fig. 2. The data was collected on a workstation with dual Pentium II processors (300MHz) with 128MB of main memory running Linux (version 2.1.126). The memory consumption of the parsing routines was monitored by overloading the `new` and `delete` calls of C++. As illustrated in the figure, the OO representation dramatically out performed the traditional technique employed earlier. The OO nature of TSL-IF enabled development of a seamless interface with the static analysis and code-generation module, considerably reducing development overheads. The OO representation also enabled "just-in-time" elaboration (or a lazy elaboration) of the sub networks without any change to the other components, thus reducing time and resource consumptions. Analysis of topologies consisting of a million nodes was not feasible with the earlier technique as the system would run out of memory. Failure of the front end to meet the fundamental requirements proved a hurdle for further system development. Using the OO representation efficient analysis of ultra-large topologies, without incurring redevelopment costs, was possible. This is an excellent example to highlight the importance of the OO paradigm. The techniques provided an effective and efficient solution to enable analysis of ultra-large topology specifications. The generation of configuration information in TSL format can be automated to generate different inter-network topologies. Further details on TSL is available in the literature [9].

4.2 Static Analysis and Code-Generation Modules

The static analysis and code-generation modules of USSF play an important role in reducing the memory consumed during simulation. TSL-IF generated by the TSL parser forms the primary input. The static analyzer collates information on the modules repeated in the topology. Using this information, duplicate object descriptions are identified and subsumed to a single entry in the various internal tables. Having a single entry rather than a few hundred thousands dramatically reduces their memory consumption. It also reduces the processing overheads of the data structures during simulation. The code-generator module uses the analyzed intermediate format to generate the final simulation topology. The generated code is compiled and linked with the USSF library, the WARPED library, and the USSF API compliant user developed modules to obtain the final executable. Although the generated code is currently in C++, in compliance with the implementation of WARPED and USSF, the techniques employed are independent of the implementation language.

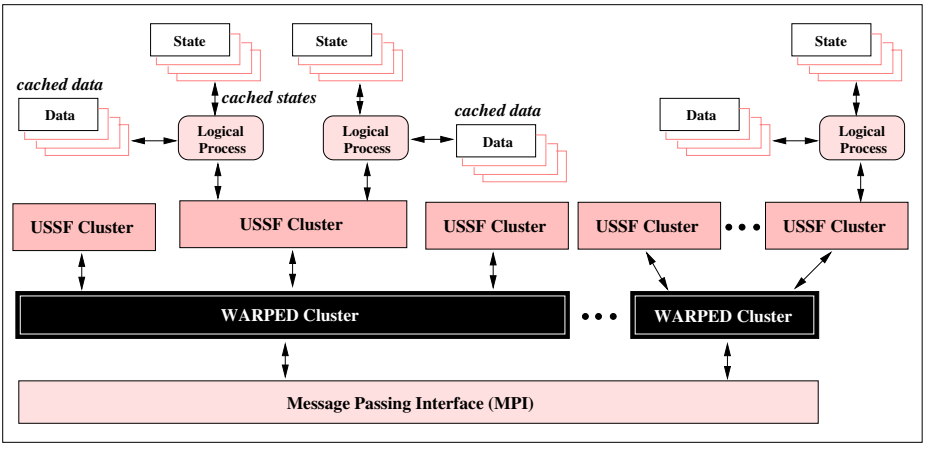


Fig. 3. Layout of an USSF simulation

4.3 USSF Kernel

The core functionality to enable simulation of ultra-large networks is provided by the USSF kernel modules. The kernel is built on top of the WARPED simulator and presents a similar application program interface (API) to the model developer. The core of the USSF kernel is the USSF cluster. The USSF cluster module represents the basic building block of USSF simulations. A USSF cluster performs two important functions; namely (i) it acts as an LP to WARPED; and (ii) it acts as a cluster to the application programmer. As shown in Fig. 3, the USSF cluster is used to group a number of LPs that use the same description together. A single copy of an user process is associated with different data and states to emulate its various instances. In order to enable this feature, the data associated with the different model descriptions need to be decoupled. The USSF API employs standard OO techniques to insulate the model developer from such implementation intricacies. The USSF cluster uses file caches to maintain the data and states of the various processes. The caching helps in regulating the demands on main memory. Separate data and state caches are maintained to satisfy concurrent accesses to data and state spaces and to reduce cache misses. A Least-Recently-Used (LRU) cache replacement policy is used. OO techniques have been used to decouple the various memory management routines from the core. This design not only provides a simple mechanism to substitute various memory management algorithms, but also insulates the USSF cluster from their implementation details.

The USSF cluster is responsible for scheduling the various application processes associated with it. The cluster appropriately translates the calls made by the WARPED kernel into corresponding application process calls. It is also responsible for routing the various events generated by the application to the WARPED kernel. The WARPED kernel permits exchange of events between the

USSF clusters. To enable exchange of events between the various user LPs, the framework cluster translates the source and destination of the various events to and from USSF cluster ids. In order for the USSF kernel to perform these activities, a table containing the necessary information is maintained by the kernel modules. The table is indexed using the unique process ids that need to be associated with each user LP. To reduce the number of entries in this table, a single entry is maintained for a group of LPs sharing a process description. The static analysis phase assigns contiguous ids to processes constructed using the same simulation objects. This fact is exploited to efficiently construct and maintain the table. The framework cluster also maintains a file based state queue in order to recover from rollbacks [3] that can occur in a Time Warp simulation. An simple incremental state saving mechanism with a fixed check-pointing interval is used for this purpose. Since the state of the USSF cluster consists of merely the offsets into this file, the memory overheads of state saving are considerably reduced. A simple garbage collection mechanism triggered by the garbage collection routines in WARPED is used to prune the state queues. Access to the various methods in the framework kernel is provided via a set of simple application program interface (API). The API is illustrated in the following subsection.

4.4 USSF Application Program Interface

The USSF API closely mirrors the WARPED API [3]. This enables existing WARPED applications to exploit the features of the framework with very few modifications. The USSF Kernel presents an API to the application developer for building local processes (LPs) based on Jefferson's original definition [3] of Time Warp. The API has been developed in C++ and the OO features of the language have been exploited to ensure it is simple and yet robust. The API plays a critical role in insulating the model developer from the intricacies involved with enabling ultra-large parallel simulations. The interface has been carefully designed to provide sufficient flexibility to the application developer and enable optimal system performance.

The basic functionality the kernel provides for modeling LPs are methods for sending and receiving events between the LPs and the ability to specify different types of LPs with unique definitions of state. The user needs to derive the LP's description, state and data classes from the corresponding interface classes. The USSF kernel utilizes the interface classes to achieve its functionality such as swapping of data and states, mapping data and state with corresponding LPs, state saving, rollback recovery, and handling interface calls from WARPED. Interfaces for creating and exchanging events are also defined. However, the user is required to override some of the kernel methods. More specifically, the `initialize` method gets called on each LP before the simulation begins. This gives each LP a chance to perform any actions required for initialization. The method `finalize` is called after the simulation has ended. The method `executeProcess` of an LP is called by the USSF kernel whenever the LP has at least one event to process. The kernel calls `allocateState` and `allocateData` when it needs the LP to allocate a state or data on its behalf. Although it is the responsibility of

the modeler to assign unique ids to each LP, the static analysis modules in the USSF perform this functionality. The USSF kernel provides all the necessary interfaces needed by WARPED and handles all the overheads involved in a enabling ultra-large simulations providing a simple yet effective modeling environment.

5 Experiments

This section illustrates the various experiments conducted using the ultra-large simulation framework. The network model used to conduct the experiments was a hierarchical network topology constructed by recursively nesting topologies. Since the topologies are elaborated to a flat network, any random network could have been used. In order to ease study and experimentation, a basic topology consisting of ten nodes was recursively used to scale the network models to the required sizes. Validation of the simulations were done by embedding sanity checks at various points in the model. The nodes representing the terminal points in the network generated traffic based on random Poisson distributions. The nodes generated packets of size 64 bytes whose destinations were chosen a normal distribution based on the size of the topology. The models of the network components used in the specification were developed using the framework's API. The TSL parser was used to analyze the hierarchical topologies and generate appropriate network models. The generated models were compiled and linked with USSF and WARPED libraries to obtain the final executable.

Figure 5 presents the time taken for simulating the generated topologies with WARPED and with USSF. Since the APIs of WARPED and USSF are similar, the models were fully portable across the two kernels. The data was collected using eight workstations networked by fast Ethernet. Each workstation consisted of dual Pentium II processors (300MHz) with 128MB of RAM running Linux (version 2.1.126). The simulation time for the smaller configurations is high due to the initial setup costs. An aggressive GVT period was used to ensure rapid garbage collection by WARPED. As illustrated in Fig. 5, for smaller sized models, WARPED performs better than USSF. The reductions in performance is due to the added overheads needed to enable ultra-large simulations. As shown in Fig. 5, the performance of WARPED deteriorates as the size of simulation increases. A study indicated that the drop in performance occurred when the memory consumptions of the parallel processes exceeded the physical memory sizes of the machines and virtual memory overheads began to dominate.

In order to study the scalability issues of the framework, the hierarchical topology consisting of hundred thousand nodes was used. The above mentioned experimental platform was used; and the number of processors used was varied between one and sixteen. Figure 5 presents the timing information obtained from this experiment. The performance of the framework increases as the number of processors are increased, since more computational resources are available for the concurrent processes to use.

As illustrated by the experiments, USSF enables simulation of very large networks, which was the initial goal of the research. The current implementation

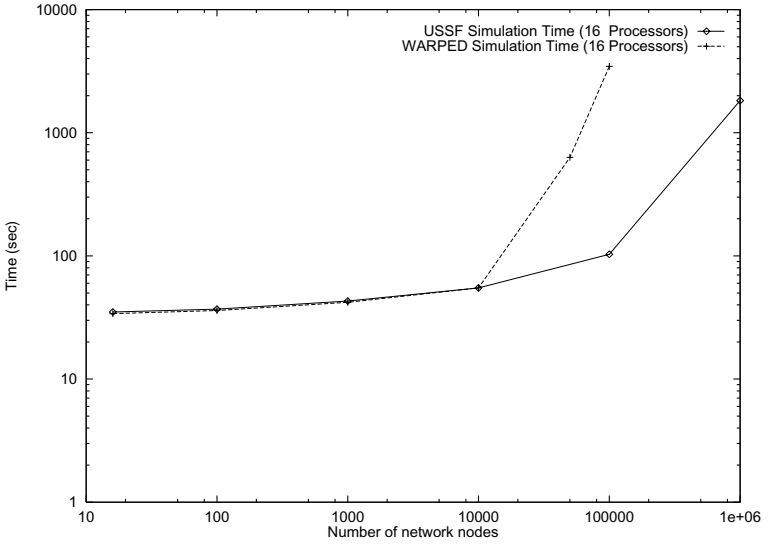


Fig. 4. Simulation time

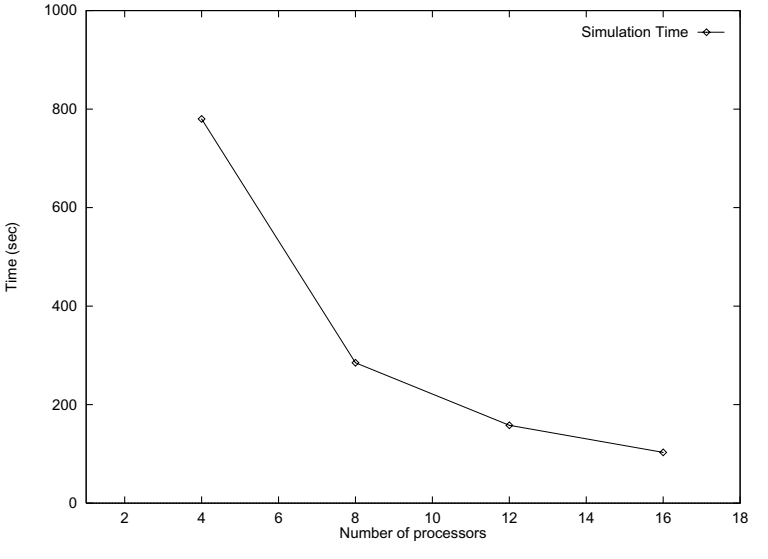


Fig. 5. Scalability data

does not include a number of other proposed partitioning and load balancing optimizations that improve performance of the Time Warp simulations [11]. Although OO design techniques incur some degradation in performance when compared to their non-OO counter parts, the primary motivation to design an OO framework was to develop an simple yet effective simulation environment. Studies are being conducted to improve the performance of USSF.

6 Conclusion

Modern communication networks and their underlying components have grown in size and complexity. Simulation analysis with models built to reflect the ultra-large sizes of today's networks is important in order to study scalability and performance issues. Parallel simulation techniques need to be employed in order to achieve time versus resource tradeoffs. The size and complexity of such parallel simulations requires the system to be carefully developed using standard design techniques such as OO design. Efficient and robust interfaces are necessary to ease application and simulation kernel development. In order to insulate the application developer from such intricacies and to ease modeling and simulation of large networks, an OO framework for simulation of ultra-large networks was developed. The issues involved in the design and development of the framework were presented. The experiments conducted using the framework were illustrated. The need for efficient OO design to enable the system was quantitatively highlighted. From the experimental results the capacity to perform such large simulations in resource restricted platforms was demonstrated. Currently, techniques to improve performance of the framework are being explored. USSF provides a convenient and effective means to model and study ultra-large communication networks of today and tomorrow.

References

1. V. Paxson and S. Floyd. Why we don't know how to simulate the internet. In *Proc. 1997 Winter Simulation Conference*, pp. 44–50, December 1997. 37, 37, 37, 37, 38
2. A. M. Law and M. G. McComas. Simulation software for communications networks: The state of the art. In *IEEE Communications Magazine*, pp. 44–50, March 1994. 37, 37
3. R. Radhakrishnan, D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. An Object-Oriented Time Warp Simulation Kernel. In *Proc. Int. Symp. Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, volume LNCS 1505, pp. 13–23. Springer-Verlag, December 1998. 38, 38, 39, 39, 44, 44, 44
4. P. Huang, D. Estrin, and J. Heidemann. Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols. In *Proc. Int. Symp. Modeling, Analysis and Simulation of Computer and Telecommunication Networks*, October 1998. 38, 38, 38
5. B. J. Premore and D. M. Nicol. Parallel simulation of TCP/IP using TeD. In *Proc. 1997 Winter Simulation Conference*, pp. 437–443, December 1997. 38, 38, 38
6. D. M. Rao, N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. Unsynchronized parallel discrete event simulation. In *Proc. 1998 Winter Simulation Conference*, pp. 1563–1570, December 1998. 38, 38
7. P. A. Wilsey and A. Palaniswamy. Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation. In *Proc. Int. Conf. on Simulation and Hardware Description Languages*, pp. 143–148. Society for Computer Simulation, January 1994. 38
8. K. Fall. Network emulation in the Vint/NS simulator. In *Proc. 4th IEEE Symp. Computers and Communications*, July 1999. 38

9. D. M. Rao, R. Radhakrishnan, and P. A. Wilsey. FWNS: A Framework for Web-based Network Simulation. In *1999 Proc. Int. Conf. Web-Based Modelling & Simulation (WebSim 99)*, pp. 9–14, January 1999. 39, 39, 40, 42
10. T. J. Parr. *Language Translation Using PCCTS and C++*. Automata Publishing Company, January 1997. 40
11. R. Fujimoto. Parallel discrete event simulation. *CACM*, 33(10):30–53, October 1990. 46