An Active Networks Simulation Environment

Dhananjai M. Rao
Philip A. Wilsey
Experimental Computing Laboratory
Department of ECECS
P.O. Box 210030, Cincinnati, OH 45221-0030
dmadhava@ececs.uc.edu

The steady increase in size and complexity of communication networks, coupled with growing needs and demands, has motivated the development of active networks. Active networking techniques embed computational capabilities into conventional networks, thereby massively increasing the complexity and customization of the computations that are performed with a network. One of the most effective techniques for studying and analyzing active networks is computer-based simulation, but customized and flexible tools are required to ease effective use of simulations. In an endeavor to address these diverse issues, an Active Networks Simulation Environment (ANSE) has been developed. ANSE provides an environment for modeling, parallel simulation, and Web-based simulation of active networks. This paper presents the issues involved in the design and development of ANSE and the results obtained from the experiments conducted to evaluate the effectiveness of ANSE. Studies indicate that ANSE provides an effective environment for modeling and simulation of large-scale active networks.

Keywords: Network topologies, time warp, verification, validation

1. Introduction

Computer and communication networks have steadily increased in size and complexity to meet the growing needs and demands of modern computing [1, 2]. Correspondingly, networking techniques also need to advance to effectively use the tremendous improvements in communication infrastructures and provide cost-effective and high-quality networking solutions. Toward this end, active networking architectures are being researched [3]. In an active network, the nodes constituting the network are capable of performing customizable general-purpose processing (or *services*) on the datagrams flowing through them [2-4]. Active networking techniques enable a massive increase in the complexity and customization of networking services. A more detailed description on the internal structure and organization of active networks is presented in Section 2.

Study and analysis of active networks are necessary to effectively design, manufacture, deploy, and maintain them [2, 4]. In-depth analysis and accurate profiling of these large, complex network architectures are not feasible using conventional analytical techniques [5], primarily because they are still at their nascent stages and are not well understood [6]. Hence, experimental techniques such as simulation must be employed [2, 5, 7]. Simulation, particularly discrete event simulation, has proven to be the most effective tool to study conventional networks and is widely used [2, 4, 5]. Parallel simulation techniques are often employed to provide better resource versus time trade-offs, thereby enabling simulation of large models in reasonable time frames [8].

Model development and its verification and validation (V&V) play a critical role in simulation studies [8, 9]. Furthermore, the network models should reflect the size and complexity of the actual networks to ensure that crucial scalability issues do not dominate during validation of simulation results. However, developing sufficiently verified and validated models of large, complex active networks is an involved and time-consuming task. The scenario is further aggravated due to several hurdles, such as (1) specification and use of large networks [1]; (2) infrastructure for rapid simulation of large models [4]; (3) portability, interoperability, and reuse of existing models, including those developed by third-party researchers and manufacturers [10, 11]; and (4) availability and accessibility of the models [8, 12]. Consequently, customized and flexible tools need to be employed to address the above-mentioned issues, thereby facilitating modeling, simulation, and analysis of active networks [2, 7, 13].

We have developed the Active Networks Simulation Environment (ANSE) in an attempt to build suitable modeling and simulation tools for active network modeling and simulation. ANSE consists of a set of tools and simulation frameworks that have been integrated using a unified modeling frontend. The unified modeling frontend includes a hierarchical Topology Specification Language (TSL) and an Application Program Interface (API). Network models are developed by using TSL by suitably interconnecting simulation objects developed using the API. These network models can be used to conduct stand-alone parallel simulations, develop Web-based repositories of the simulation objects (i.e., ANSE factories), or perform Web-based simulations by using the tools and frameworks constituting ANSE. ANSE's modeling frontend fully insulates the application modules from the specifics of the different frameworks constituting ANSE. In other words, the intricacies involved in enabling parallel and Web-based simulations are transparent to the model developer. The Web-based modeling and parallel simulation techniques have been used to address the (above-mentioned) hurdles that hinder effective study and analysis of active networks using simulations [2, 7, 8, 13]. ANSE also includes a library for modeling and simulation of active networks based on the Packet Language for Active Networks (PLAN). ANSE's API has been used to develop the PLAN library.

This paper presents the issues involved in the design and development of ANSE. Section 2 presents a brief description of time warp, the optimistic synchronization strategy employed by the frameworks constituting ANSE. A detailed description of the modeling frontend of ANSE is presented in Section 3, along with a brief overview of ANSE. The framework used to enable parallel simulation of active network models is described in Section 4. The design of the Web-based simulation infrastructure of ANSE is discussed in Section 5. The section also presents the design of an ANSE factory—a Web-based repository of active networking components. The issues involved in the design and development of the PLAN library are presented in Section 6. Section 7 presents the experiments conducted using the PLAN library for evaluating the overall effectiveness of ANSE. The results obtained from the experiments are also discussed in this section. Concluding remarks and pointers to future work are presented in Section 8.

2. Background

Time warp is an optimistic synchronization strategy [4]. A time warp–synchronized parallel simulation is organized as a set of communicating, asynchronous logical processes (LPs). The LPs communicate between each other by exchanging discrete *virtual time*–stamped events [4]. Virtual time is used to model the passage of time and defines a total order on the events in the system. Each LP processes its events, incrementing its local virtual time (LVT), changing its state, and generating new events. The LPs must be synchronized to maintain the causality of the simulation; although each LP processes local events in its correct time stamp order, events are not globally ordered. Causal violations may occur due to the optimistic nature of the time warp. Causality violations are detected by an LP when it receives an event with a time stamp lower than its LVT (a *straggler* event). On receiving a straggler, a *rollback* mechanism [4] is invoked to recover from the causality error. A rollback recovers the LP's state prior to the causal violation, canceling the erroneous output events generated by sending out anti-messages and reprocessing the events in their correct causal order [4]. Each LP maintains a list of state transitions along with lists of input and output events corresponding to each state to enable recovery from causal violations through rollbacks. A periodic garbage collection technique based on global virtual time (GVT) [4] is used to prune the queues by discarding history items that are no longer needed. The distributed simulation is deemed to have terminated when all the events in the system have been processed in their correct causal order.

3. Overview

An overview of the interaction between the modeling frontend and the different frameworks constituting ANSE is illustrated in Figure 1. As illustrated in the figure, ANSE consists of a set of diverse tools and frameworks that have been integrated using a unified modeling frontend. Each framework (i.e., a backend) is geared to perform a specific task or could be oriented for simulating with a given discrete event simulation kernel. The backends can have different API method calls and different semantics (within the context of discrete event simulation). On the other hand, the unified frontend remains a constant and is independent of the backend being used for performing the actual simulation. In other words, the frontend insulates the modeler from the intricacies and specifics of the different backends. The primary motivation of this design is that the same set of models can be used to simulate with several different frameworks. This approach eliminates the need to retarget models for different platforms, which in turn reduces several modeling overheads, such as verification and validation costs. Furthermore, the design eases interoperability of the models developed using the unified frontend.

As indicated in Figure 1, the modeling frontend of ANSE consists of two major components, namely, TSL and the ANSE API. Developing a network model using the frontend can be categorized into the following two basic steps (Figure 1): (1) developing simulation objects of the networking components (such as routers, nodes, and traffic generators) to be used in a given model using the API and (2) using TSL to describe the topology (or model) of the network along with cross-references to the various components (developed using ANSE's API) to be used in the model. As shown in Figure 1, the TSL descriptions are parsed into an object-oriented, in-memory intermediate form called TSL-IF. TSL-IF constitutes the primary input to the various backends constituting ANSE and for further processing. A more detailed description of the various components constituting the modeling frontend of ANSE is presented in the following subsections.

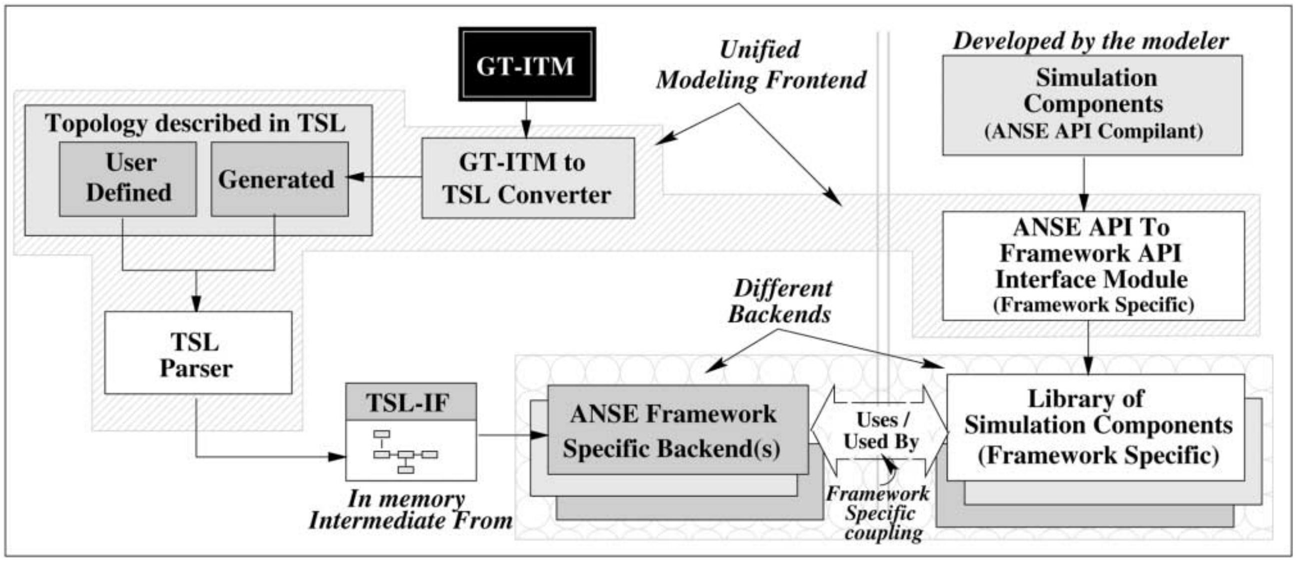

GT-ITM
Unified Modeling Frontend
Developed by the modeler
Simulation Components (ANSE API Compilant)
Topology described in TSL
User Defined
Generated
GT-ITM to TSL Converter
ANSE API To Framework API Interface Module (Framework Specific)
Different Backends
TSL Parser
TSL-IF
ANSE Framework Specific Backend(s)
Uses / Used By
Library of Simulation Components (Framework Specific)
In memory Intermediate From
Framework Specific coupling

Figure 1. An overview of an Active Networks Simulation Environment's (ANSE's) frontend

As shown in Figure 1, framework-specific modules are used to interface the models developed using ANSE's API with each backend, yielding a framework-specific library of components. The backends use the framework-specific library of components and TSL-IF to perform the actual simulation. Currently, ANSE includes two backends: one for performing stand-alone parallel and sequential simulations using the WARPED [15] simulation kernel and another backend for conducting Web-based modeling and parallel simulations. A detailed description of these backends is presented in Sections 4 and 5, respectively.

3.1 Topology Specification Language (TSL)

The primary input to ANSE (as shown in Figure 1) is the topology of the network to be simulated. The topology must be described using the TSL [16] syntax. The Backus Normal Form (BNF) of TSL grammar is shown in Figure 2. As specified by the grammar, a TSL specification consists of a set of interconnected topology specifications. Each topology specification consists of three main sections, namely, (1) the *object definition section* that contains the details of the modules that need to be used to simulate the topology, (2) the *object instantiation section* that specifies the various nodes constituting the topology, and (3) the *netlist section* that defines the interconnectivity between the various instantiated nodes. Figure 3 presents a network model along with the corresponding TSL description.

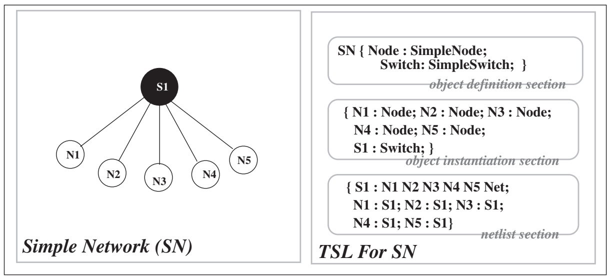
An optional *label* may be associated with each topology. The label may be used as an object definition in subsequent topology specifications to nest a topology within another. In other words, the labels, when used to instantiate an object, result in the complete topology associated with the label to be embedded within the instantiating topology. Figure 3 presents the TSL source code to model a larger network using hierarchical constructs. The model of the network is specified by interconnecting three instances of the network model shown in the figure. Using this technique, a simple subnetwork consisting of merely 10 nodes can be recursively used to construct a network with six levels of hierarchy to specify a network with a million (10^6) nodes [4, 17]. ANSE also includes a tool for translating Georgia Tech Internet topology models (GT-ITM) [18] into equivalent TSL descriptions. As shown in Figure 1, GT-ITM along with the GT-ITM to TSL conversion tool can be used to generate network topologies in TSL. A more detailed description of TSL is available in the literature [8, 19].

3.2 TSL Parser and TSL-IF

The input topology configuration is parsed into an object-oriented (OO) TSL intermediate format (TSL-IF). The TSL parser is generated using the Purdue Compiler Construction Tool Set (PCCTS) [4]. TSL-IF forms the primary input to the other modules of ANSE. TSL-IF is designed to provide efficient access to related data from the various TSL sections [17]. In conjunction with the parser, TSL-IF is also implemented in C++. The IF consists of a set of cross-referenced C++ classes, with each class representing a particular grammar entity. The IF is composed by filling in the references in the various classes generated by the parser with appropriate values. Since composition is achieved via base class references, each node can

```
design_file ::= include_list tsl_design_topology | tsl_design_topology
include_list ::= include_clause | include_clause include_list
include_clause ::= include " file_name ";
file_name ::= identifier | identifier . identifier
tsl_network ::= tsl_topology | label tsl_topology | tsl_topology tsl_network
        | label tsl_topology tsl_network
tsl_topology ::= { object_definition_section } { object_instantiation_section }
        { net_list_section }
label ::= identifier
object_definition_section ::= object_definition | object_definition object_definition_section
object_definition ::= object_name : url optional_parameter
object_name ::= identifier
url ::= host_name : port_number . factory
optional_parameter ::= parameter ; | ;
parameter ::= " string " | ""
factory ::= identifier | identifier . factory
port_number ::= number
object_instantiation_section ::= object_instantiation | object_instantiation object_instantiation_section
object_instantiation ::= object_instance : object_name optional_parameter
        | object_instance : object_name number optional_parameter
        | object_instance : label
object_instance ::= identifier
net_list_section ::= net_list | net_list net_list_section
net_list ::= object_instance : instance_list ;
instance_list := object_instance | object_instance instance_list
identifier ::= start_char any_char
start_char ::= [a - z, A - Z]
any_char ::= [a - z, A - Z, 0 - 9, _ ]
string ::= string_char | string_char string
string_char ::= [̃ ]
number ::= [0 - 9]
```

Figure 2. Backus Normal Form (BNF) of the Topology Specification Language (TSL) grammar

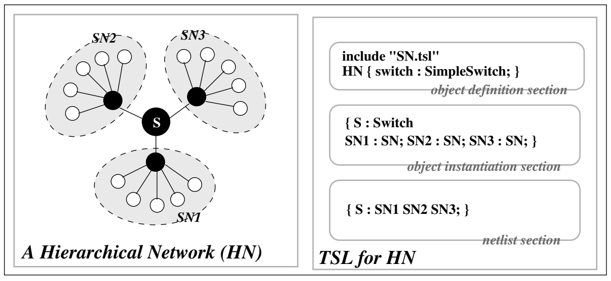


```
SN { Node : SimpleNode;
        Switch: SimpleSwitch;  }
```

```
{ N1 : Node; N2 : Node; N3 : Node;
  N4 : Node; N5 : Node;
  S1 : Switch; }
```

```
{ S1 : N1 N2 N3 N4 N5 Net;
  N1 : S1; N2 : S1; N3 : S1;
  N4 : S1; N5 : S1}
```

(a) Simple network


SN2
SN3
S
SN1
A Hierarchical Network (HN)
TSL for HN
object definition section
object instantiation section
netlist section

```
include "SN.tsl"
HN { switch : SimpleSwitch; }
```

```
{ S : Switch
SN1 : SN; SN2 : SN; SN3 : SN; }
```

```
{ S : SN1 SN2 SN3; }
```

(b) Hierarchical network

Figure 3. Examples of network models and corresponding Topology Specification Language (TSL) descriptions

refer to another node or even a subnetwork. This provides an efficient data structure for representing and analyzing hierarchical networks [4, 17].

3.3 ANSE API and Library

ANSE presents an interface to the application developer for modeling a network as set of communicating LPs. The LPs are modeled as entities that send and receive events to and from each other and act on these events by applying them to their internal state. Figure 4 shows the Universal Markup Language (UML) diagram for the core classes that constitute the API. As illustrated in the figure, the `Network Node` class forms the parent class for all the networking components in the system. The `ActiveNode` and `PLAN Node` are derived from this class. The `NetworkNode` class is used to model conventional networking components, while the `ActiveNode` is used to model active components. The `NetworkNode` also provides methods for accessing routing tables and supports primitive domain name services (DNS).

The state (`NetworkNodeState` and `Active NodeState`) and packet (`Packet` and `ActivePacket`) classes corresponding to the LP hierarchy are also shown in Figure 4. The state classes are used to encapsulate the state information associated with each node/component. The state information is used by WARPED (the underlying simulation kernel) to enable *rollbacks* [14] and recover from causal violations that could occur in time warp–based simulations [15]. The `Packet` class is used for all communications between the nodes constituting a network model. The packets, in turn, represent the discrete events in the simulation. The API has been developed in C++, and the object-oriented features of the language have been exploited to ensure it is simple and robust. The API plays a critical role in insulating the model from the underlying simulation kernel. The interface has been carefully designed to provide sufficient flexibility to the application developer and enable optimal system performance. Further details on the API are available in the literature [4, 6].

4. Parallel Simulation Framework (PSF)

The Parallel Simulation Framework (PSF) of ANSE has been developed to enable stand-alone parallel simulation of active network models developed using ANSE's modeling frontend. Figure 5 presents an overview of the PSF. As shown in the figure, the primary input to the framework is the TSL-IF (in-memory, intermediate form) of the network to be simulated. TSL-IF is obtained from ANSE's modeling frontend, which parses the input TSL description into TSL-IF. The elaborator is used to elaborate or "flatten" hierarchical TSL specifications. As shown in Figure 5, a backend code generator uses the elaborated network model to generate ANSE API compliant simulation code. The generated code is compiled with the necessary libraries—namely, the WARPED library, a general-purpose parallel discrete event simulation kernel; the ANSE library, which provides necessary implementations for the API calls; the PLAN library; and any other use-defined libraries (as the case may be) to obtain the final executable. This executable performs the actual simulation when run. Detailed descriptions of the various modules specific to the PSF are presented in the following subsections.

4.0.1 Elaborator

Hierarchical constructs provide convenient techniques to specific large networks by reusing the specification for smaller subnetworks [4, 17]. However, the hierarchical constructs have to be elaborated or "flattened" prior to simulation [4, 17]. Elaboration is the process in which each hierarchical level is broken down to its constituting components. The basic steps involved in elaborating a hierarchical specification are shown in Figure 6. As illustrated in the figure, the elaborator starts with a user-specified topology and recursively traverses the various subtopologies in the model and creates new instances of the subtopologies and the objects. Elaboration of subtopologies is done before they are imploded into the enclosing topology. Imploding hierarchies involves inclusion of all necessary object definitions, object instantiations, and corresponding data structures. Elaboration may be done *statically* or at *runtime*. Static elaboration occurs prior to code generation, while runtime elaboration occurs just before simulation commences, when the generated code is executed. As shown in Figure 5, static elaboration has been employed in PSF. The TSL-IF generated by the parser and the elaborated topology is also represented in TSL-IF (as shown in Figure 5).

4.1 Code Generator

As shown in Figure 1, the backend code generator uses the elaborated TSL-IF to generate a simulatable model from the given TSL description. The generated code in C++ is in concordance with all the other components of ANSE. The OO nature of TSL-IF has been exploited in the development of the code generator. The generated code is compliant with WARPED's. A model developer can directly develop the network model (compliant with WARPED's API) and bypass these stages. However, the complexity involved in model development will be considerably higher. The backend code generator can be replaced with a different code generator to retarget the generated code for different frameworks.

4.2 WARPED

The parallel simulation capabilities of the PSF have been enabled by developing the framework around a general-purpose discrete event simulation engine. OO techniques have been employed to isolate the various modules of ANSE from the underlying simulation kernel. This

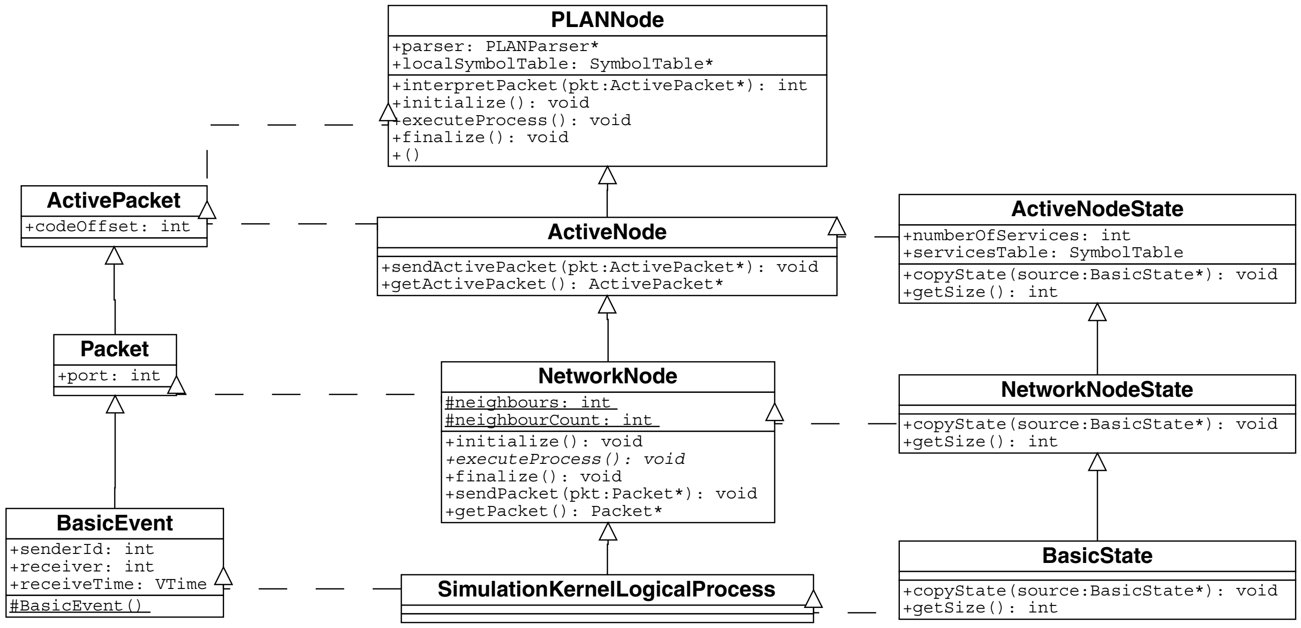


Figure 4. The core classes forming the Application Programming Interface (API) of the Active Networks Simulation Environment (ANSE)

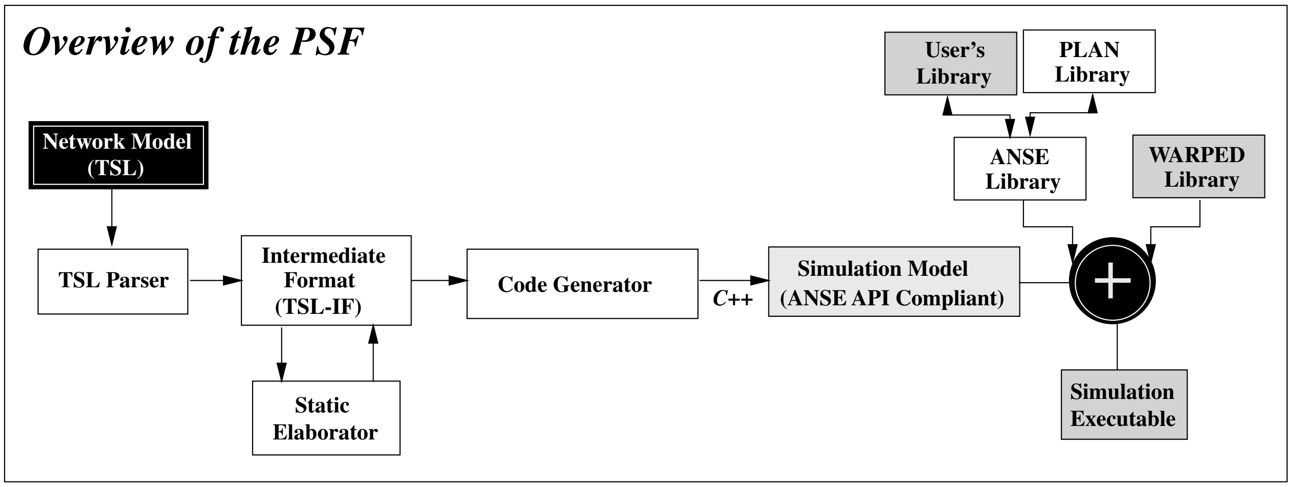


Figure 5. Overview of the Parallel Simulation Framework (PSF)

design not only provides a desired level of "separation of concerns" but also enables the use of different simulation kernels without having to modify application modules. The current implementation of ANSE uses the WARPED simulation kernel [15] to enable sequential and parallel simulation of active network models. WARPED is an API for a general-purpose discrete event simulation kernel with different implementations [15]. ANSE uses the sequential kernel and the time warp–based parallel simulation kernel of WARPED.

The WARPED kernel presents an interface to build logical processes based on Jefferson's original definition [14] of time warp [15]. The kernel provides an API to build different LPs with unique definitions of state [15]. The basic functionality for sending and receiving events between LPs using a message-passing system is supported by the kernel. In WARPED, LPs are placed into groups called "clusters." LPs on the same cluster communicate with each other without the intervention of the message-passing system, which is *faster* than communication through the message system

```
step 1: Initialize elaborator
    1. Initialize new symbol table and IF
    2. Search input IF and locate node corresponding to user specified top level topology
    3. Call elaboration (step 2) with new topology
step 2: Elaboration subroutine (parameter topology)
    1. Process the list of netlists specified in the topology. If the node is an object instantiation perform step 3. If
       the node is a topology label perform step 4.
step 3: Elaborate object instantiation
    1. Create new instance of the object instantiation with mangled labels.
    2. Create new object definition for the new instance with mangled labels and add to new symbol table, if necessary.
    3. Add new object instantiation to the new topology and update netlist entry.
step 4: Elaborate sub-topology
    1. Instantiate temporary symbol table and IF
    2. Recursively call elaboration with the sub-topology
    3. Implode new IF to the new topology
```

Figure 6. Phases in elaborating a Topology Specification Language (TSL) design

[15]. Although LPs are grouped together into clusters, they are not coerced into synchronizing with each other. Control is exchanged between the application and the simulation kernel through cooperative use of function calls. Further details on the API and working of WARPED are available in the literature [15].

5. Web-Based Simulation Framework (WSF)

Simulation has shown to be an effective technique for studying large and complex networks. Despite their effectiveness, the complexity of model development and model V&V continues to exacerbate their use. Some of the dominant issues are as follows: (1) portability, interoperability, and reuse of existing models, including those developed by third-party researchers and manufacturers [10, 11]; (2) availability and accessibility of the models [8, 12]; and (3) infrastructure for rapid simulation of large models [4]. These issues are magnified in the case of active network simulations because several active architectures and several active protocols have been proposed. These protocols need to be studied in conjunction with each other [13]. Furthermore, active networks have to coexist and interact with conventional networks because an immediate shift in the networking paradigm is not feasible. Therefore, active networks have been studied in conjunction with conventional networks. Consequently, effective reuse of existing models is not only an attractive solution but is often the only viable alternative. Web-based modeling and simulation are an effective solution to address these issues. The World Wide Web (WWW) provides an excellent infrastructure for information exchange and large-scale simulation [20]. However, the complex interactions required for modeling and simulation render the raw WWW services insufficient [16]. Hence, in an endeavor to address these issues and ease effective use of simulations, support for Web-based simulation has been developed as a part of ANSE.

The Web-based Simulation Framework (WSF) of ANSE has been developed by suitably modifying an existing framework for Web-based network simulations (FWNS) [19]. A brief description of the modified infrastructure is presented in this section, while a detailed description of FWNS is available in the literature [8, 19]. An overview of the WSF is shown in Figure 7. The WSF provides a framework for developing a Web-based repository of components and the infrastructure for Web-based distributed simulations. As shown in Figure 7, WSF provides a Hypertext Markup Language (HTML) interface based on Common Gateway Interface (CGI) scripts, as well as a text-based frontend that can be used to interact with the ANSE server. The server controls and coordinates the various parallel and distributed activities of the framework. As shown in Figure 7, the primary input is the topology of the network model or a TSL description. The input TSL description is parsed into TSL-IF in the conventional manner. Unlike the PSF, the TSL-IF is not elaborated but is used by the various components of the ANSE server in its hierarchical form to minimize memory consumption. However, processing of TSL-IF reflects the various phases of elaboration.

The ANSE server also performs the task of collaborating with the distributed ANSE factories and coordinating the simulations. The *simulation manager* (shown in Figure 7) performs the activities associated with coordinating with the various ANSE factories (via the *factory manager*) to set up a distributed, Web-based simulation. The *factory manager* performs the tasks of interacting with the distributed ANSE factories using a predefined protocol

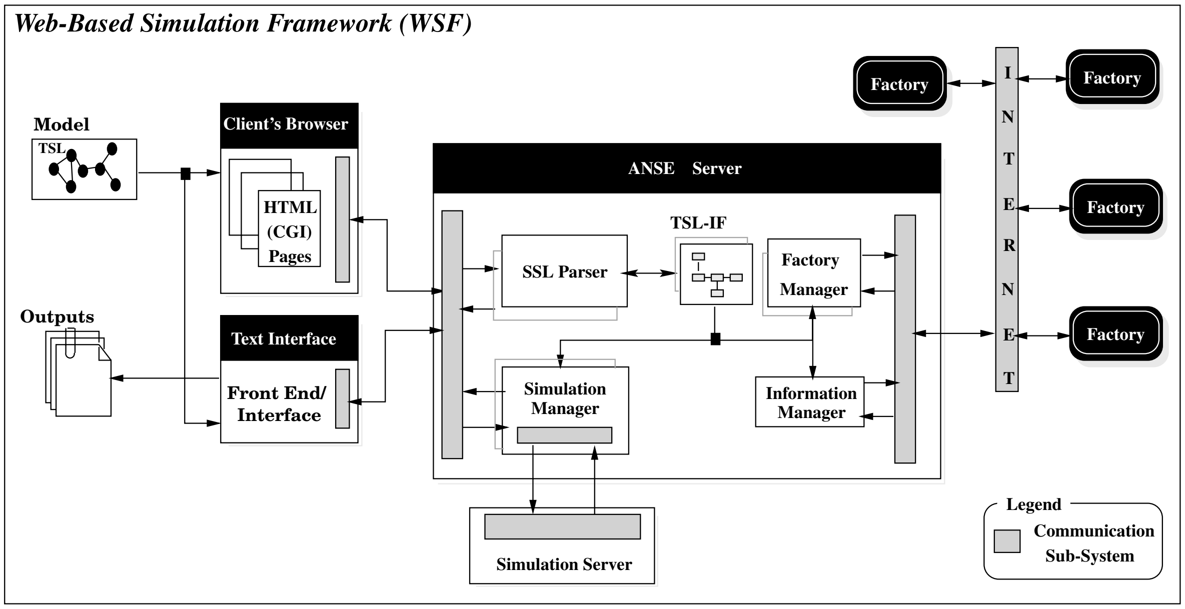


Figure 7. An overview of the Web-based Simulation Framework

provided by FWNS [19]. It not only provides a uniform interface to communicate with different object factories but also insulates the other modules of the ANSE server from the intricacies of the underlying protocols. A more detailed description of the components constituting the ANSE server is available in the literature [8, 19].

5.1 ANSE Factory

WSF also provides a framework for developing Web-based repositories of simulation objects, called a *factory*. An ANSE factory can be viewed as a Web-based repository of components with an added capability for simulating them. The factories provide a convenient mechanism for sharing the simulation object over the WWW, thereby increasing availability and accessibility of simulation modules. The factories play a pivotal role in providing a uniform interface to different simulation entities and simplify their maintenance and generation. This feature reduces overheads associated with interoperability and reuse of components without compromising proprietary information issues [16]. Figure 8 shows the layout of an ANSE factory. The initial handle to a factory is provided by the *gateway* module via a specified port on a given workstation. The Internet Protocol (IP) address of the workstation coupled with the port number must be used to refer and interact with a given factory. For instance, it is this IP address/port number pair that must be used in the TSL descriptions (along with an *object definition*) so that the simulation object to be used in the network model can be obtained from the ANSE factory. Figure 9 shows the TSL example shown in Figure 3a along with factory specifications.

The task of interacting with an ANSE server to create objects and set up the simulation is handled by the *session manager*. The *configuration manager* acts as an interface between the session manager and the simulation subsystem. It tailors the objects generated by the factory to meet the user's specifications. The simulation subsystem constitutes the actual simulation engine of the factory. In the current implementation, the WARPED simulation kernel has been used as the simulation engine. A factory is built from subfactories and *object stubs*. The object stubs are the atomic components of an ANSE factory. The stubs provide information about the simulation module, instantiate the module, and provide a conceptual link between the physical component and its simulation object. The simulation modules developed using ANSE's API (see Section 3.3) can be integrated into an ANSE factory without any changes. Simulation objects generated by the factories could be *local* or *remote*. Local objects provide a trade-off between intellectual property protection and component accessibility. Currently, all the simulation objects generated by an ANSE factory are *local* (i.e., they are also simulated on the same factory that houses them). Further details on component factories along with API specifications are available in the literature [8, 16].

6. PLAN Library

As illustrated in Figure 4, ANSE's API has been used to develop a library for modeling and simulating active networks based on PLAN [21]. PLAN is a simple, functional

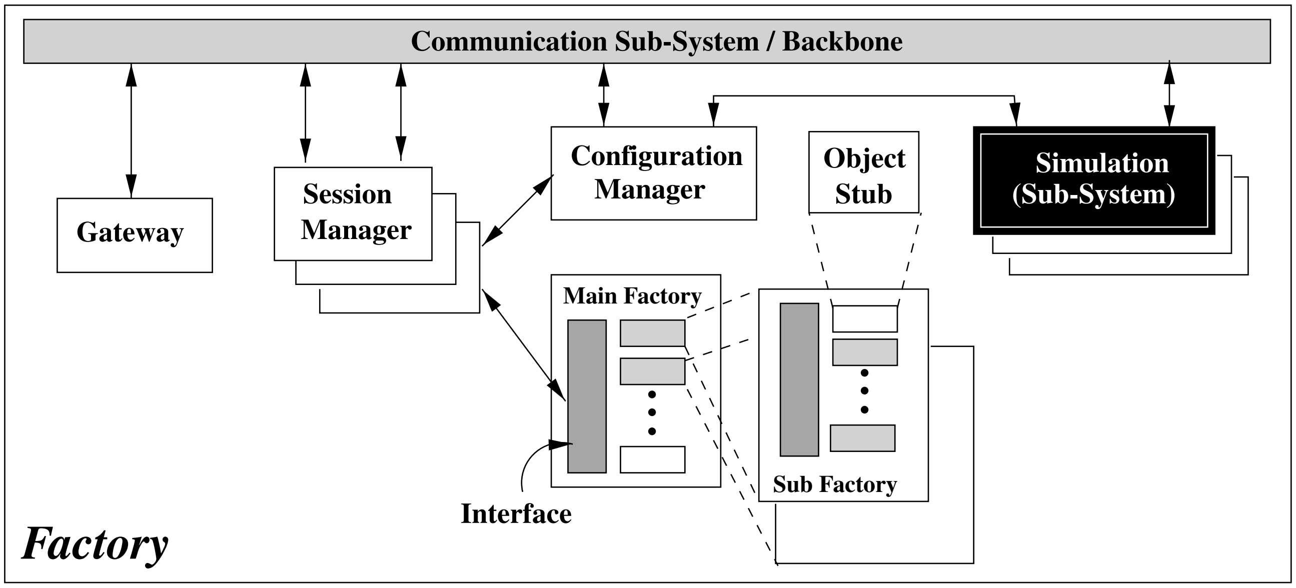
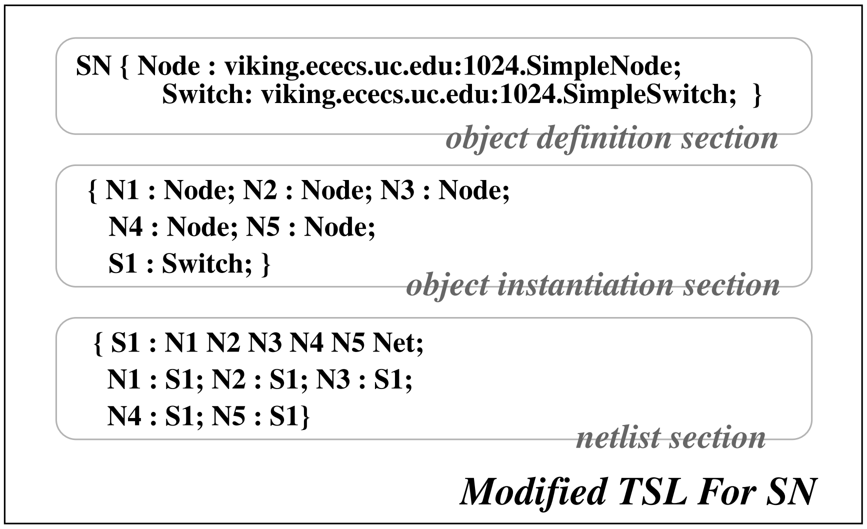


Figure 8. Layout of an Active Networks Simulation Environment (ANSE) factory



```
SN { Node : viking.ececs.uc.edu:1024.SimpleNode;
         Switch: viking.ececs.uc.edu:1024.SimpleSwitch;  }
```

object definition section

```
 { N1 : Node; N2 : Node; N3 : Node;
   N4 : Node; N5 : Node;
   S1 : Switch; }
```

object instantiation section

```
 { S1 : N1 N2 N3 N4 N5 Net;
   N1 : S1; N2 : S1; N3 : S1;
   N4 : S1; N5 : S1}
```

netlist section

Modified TSL For SN

Figure 9. Topology Specification Language (TSL) shown in Figure 3a, modified for the Web-based Simulation Framework (WSF)

programming language based on a subset of Markup Language (ML) with some added primitives to express remote evaluation [21]. In a PLAN-based active network, the active packets can contain a PLAN program that can be used to customize the active network to provide different networking services. The PLAN library provides a `PLANNode` that is capable of parsing and interpreting PLAN packets. A PLAN parser constructed using PCCTS is used to parse incoming PLAN packets into an OO intermediate format (IF). The IF is fed to a PLAN interpreter that executes the program contained in the packet. The interpreter supports all PLAN constructs, including recursive function calls, exceptions, and forwarding of any PLAN packets generated during interpretation. The PLAN library also contains a `PacketInjector` component that can be used to inject PLAN packets into the simulated network. The `PacketInjector` can be used to inject a PLAN program from a file or obtain the PLAN program interactively from the user. The `Packet Injector` can be driven using a variety of traffic generators based on random-number generators available as a part of the library. The random-number generators generate traffic based on mathematical distributions such as normal distribution, constant delay distributions, Poisson distribution, and Pareto distributions. The library also contained components for modeling different types of communication links with different parameters such as transmission delay and packet loss ratios.

The runtime structure of a typical PLAN-based active network is shown in Figure 10. As illustrated in the figure, a single instance of the PLAN parser and interpreter is shared by the different `PLANNodes`. The design helps to minimize the overall resource requirements (memory, in particular) of the simulations, thereby enabling simulation of larger networks using available hardware resources. However, in parallel simulations, a single instance of the PLAN parser and interpreter is used in each cluster. This approach is a trade-off between the overall memory requirements of the simulation versus simulation overheads such as communication and concurrency. It must be noted that concurrent demands for the parser and interpreter never arise because execution of events on a cluster proceeds in a serial order. In other words, although the WARPED clusters and the LPs operate asynchronously with each other, the events on a given cluster are executed serially. Hence, in a given cluster, only one LP can be active (or processing an event) at a time, and the PLAN parser and interpreter are assigned (or reserved) for use by that LP. Since parsing of PLAN packets and their interpretation are two distinct and independent stages, they can be cascaded or pipelined to improve performance—that is, when a previous packet is being interpreted, the next packet can be parsed. Such a design is of considerable benefit in shared-memory multiprocessor (SMP) platforms. Any dependencies or inconsistencies that could arise due to asynchronous pipelining can be resolved by directly using the optimistic simulation infrastructure. In other words, if inconsistencies arise, then the simulation will get *rolled* back and the events would get reprocessed in the correct causal order. However, such a design has not been adopted in the current implementation because other simulation techniques such as sequential simulation and conservative simulations may not be capable of supporting such a design.

As shown in Figure 10, the routing tables and DNS tables (built and maintained by `NetworkNodes`) are also shared between the various nodes constituting the simulation. This design also helps to reduce the overall memory requirements of the simulations. As explained earlier, concurrent access to these data structures does not arise. Hence, complex locking mechanisms or semaphores are not necessary to ensure their consistency and coherence. The routing and DNS tables are replicated at each cluster (in a parallel simulation) to minimize simulation overheads. The runtime modules of ANSE (present in the ANSE library) assist in constructing the tables by providing necessary information about the network model being simulated. Although the LPs are partitioned onto different clusters, the necessary information (such as name of the nodes along with the interconnectivity data) related to all the nodes is extracted and filled into the tables. In the current implementation of ANSE, the LPs are equally divided among the clusters used in a simulation (i.e., each cluster has almost an equal number of LPs). ANSE's API also includes interfaces for implementing other partitioning algorithms. It must be noted that partitioning (i.e., assignment of nodes to clusters) and parallel simulation are completely transparent to the application modules. Furthermore, the library has been developed such that the models and the generated code need not change based on the number of clusters or the underlying synchronization technique used in the simulations.

7. Experiments

The experiments conducted to evaluate the performance of ANSE and the results obtained from the experiments are presented in this section. Table 1 tabulates the characteristics of the models used to conduct the experiments. The network models were described in TSL and used the various components available in the PLAN and ANSE libraries. The network models consisted of a set of interconnected PLAN nodes. The larger models such as `Model3`, `Model4`, and `Model5` were constructed using the hierarchical modeling constructs supported by TSL. The number of PLAN nodes in each model is shown in Table 1. The other components of the model, such as traffic generators, packet injectors, and links, are grouped together and tabulated in Table 1 (under the "Others" column). A route-tracing PLAN program [21] was run on the simulated network model. The route-tracing PLAN packets hop from one node to another (as they get interpreted by each PLAN node in the simulated network), and at each node

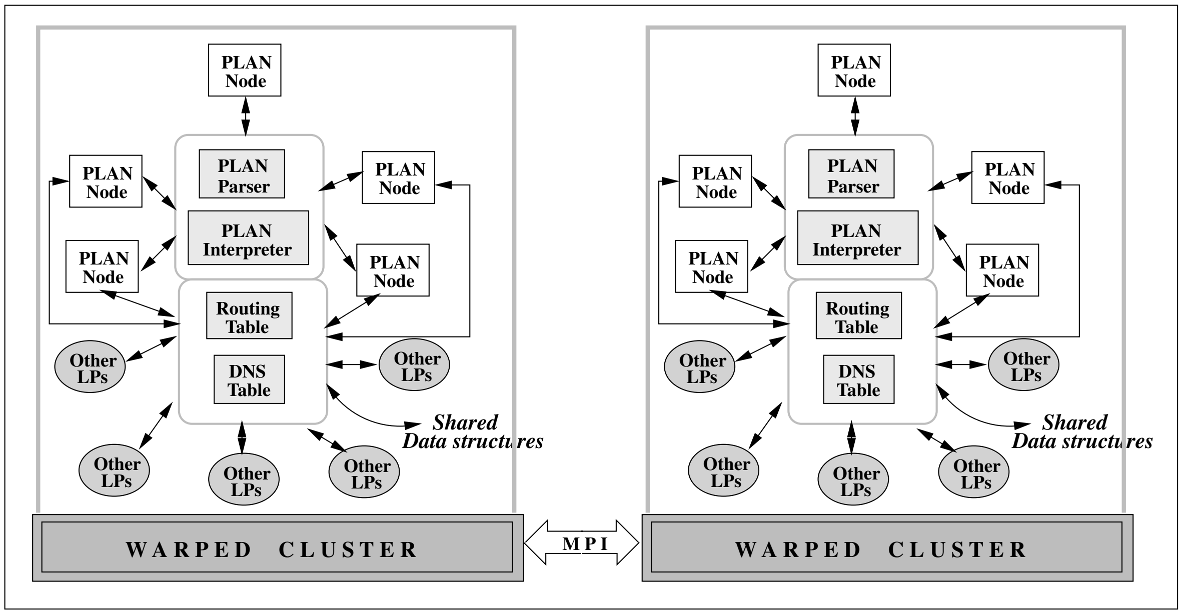


Figure 10. Runtime structure of a Packet Language for Active Networks (PLAN)–based simulation

they generate two new PLAN packets. One packet carries the information about the current hop back to the source node (i.e., the node at which the route tracing for that particular packet began). The other packet proceeds forward to trace the route until the destination node is reached. The destination node on each packet was randomly chosen from the set of nodes participating in the simulation. The `PacketInjector` (described earlier) was used to inject the PLAN packets into the simulated network. Each `PacketInjector` was programmed to generate 500 requests to trace the route to 500 randomly chosen PLAN nodes. The links interconnecting the nodes were configured through suitable parameters in the TSL description to have zero packet losses. In other words, a basic TCP/IP type of connectivity was modeled.

The graphs in Figure 11 present the time taken by PSF for performing the different phases of model generation such as parsing, elaboration, code generation, and compiling the generated code. The experiments were conducted on a Linux workstation consisting of dual Pentium II (300 MHz) processors with 128 MB of main memory (RAM). The workstations were interconnected by fast Ethernet. The timings were obtained using the standard Unix `time` command. The times plotted in the graphs are the average values computed from 10 experimental runs. As illustrated by the graphs shown in Figure 11, the overall time for generating a network model scales almost linearly with respect to the total number of objects (or LPs) constituting a network model. These data illustrate the scalability of the modeling and simulation infrastructure supported by ANSE. They also indicate that ANSE will be capable of generating large network models in reasonable time.

The parallel simulations were conducted using 1 to 16 WARPED clusters. The results obtained from the various simulations conducted using the PSF are presented by the graphs in Figure 12. The timing information for the various simulations was obtained using the standard Unix `time` command. The simulation times plotted in the graphs are the average values computed from 10 simulation runs. The timings obtained from the simulations conducted using the sequential kernel available with WARPED are also plotted in the graphs. The sequential simulator was configured for its most optimal configuration. As illustrated by the graphs in Figure 12, parallel simulation provides considerable improvements in performance even for medium-sized network models. For example, parallel simulation using 16 processors provides an order of magnitude improvement in performance when compared to a sequential simulation. The primary factor for the pronounced improvement in performance is the high-event granularity (i.e., time taken to process an event) of the active packets that need to be parsed and interpreted. As the number of processors used in the simulation is increased, the computational load gets distributed across the parallel processors, which in turn reduces the overall simulation time. However, as illustrated by Figure 12a, for small models, the performance deteriorates as the number of processors is increased. This is because the smaller models do not have sufficient concurrency and workload to use all the parallel processors. Hence, the overheads of parallel simulations (such as communication and synchronization costs) outweigh the gains accrued by increasing the number of processors.

Table 1. Characteristics of the models used in experiments

		Number of Processes			
Model	**Hierarchies**	**PLAN**	**Packet Injectors**	**Others**	**Total**
Model 1	1	9	9	41	59
Model 2	2	19	19	127	165
Model 3	2	26	26	193	245
Model 4	3	55	55	375	485
Model 5	3	110	100	781	991

PLAN = Packet Language for Active Networks.

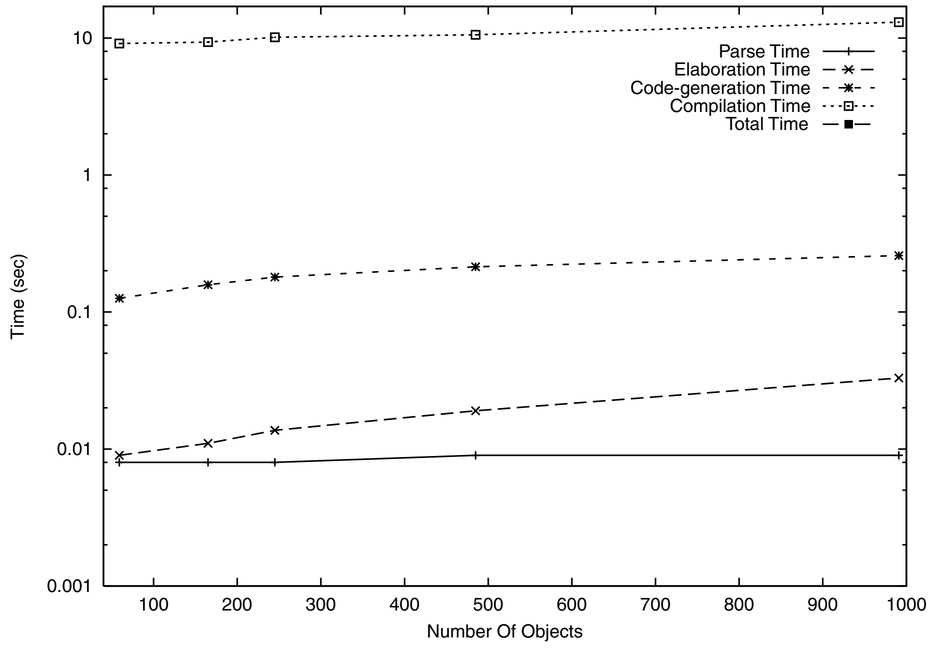
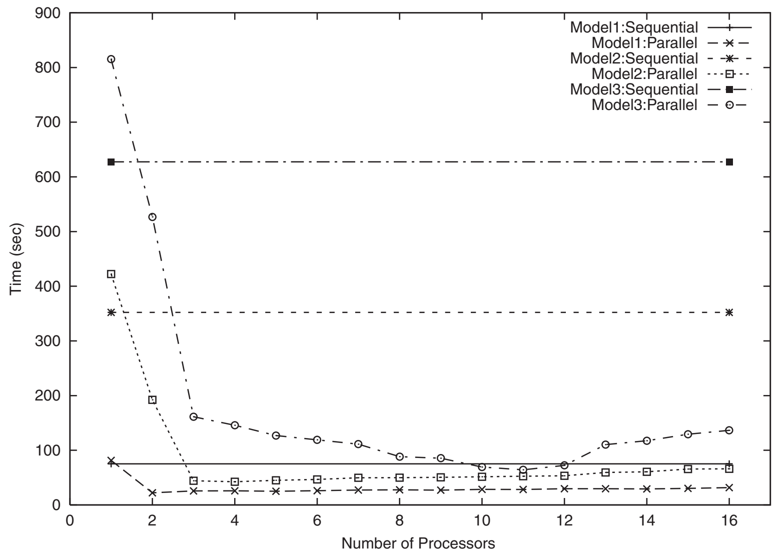


Figure 11. Time for different phases of model generation

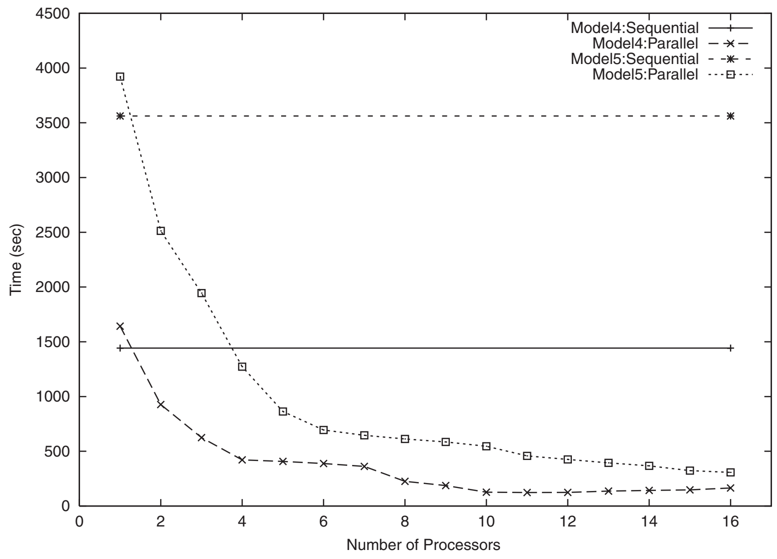
The graphs in Figure 13 present the time taken to simulate the different models shown in Table 1 using WSF. The timing also includes the time taken to set up the simulations. As shown by the graphs in Figures 12 and 13, the WSF simulations incur an additional 10% to 15% overhead. This overhead is due to the additional setup costs involved in communicating with the various ANSE factories and establishing a Web-based simulation. However, as indicated by the graphs in Figure 11, the front to back simulation time using the WSF is slightly less than that of the PSF. The data indicate that if a model is going to be repeatedly simulated, then it is better to use PSF, but it is faster to use the WSF for “one-time” simulations. The results also demonstrate the scalability of the parallel simulation frameworks. The experiments highlight that considerable improvements in the performance of the simulations can be achieved by employing parallel simulation techniques. The experiments also illustrate the overall effectiveness of ANSE for modeling and simulation of active networks.

8. Conclusions

The issues involved in the design and implementation of ANSE were presented in this paper. The experiences gained during the development of ANSE also highlight a number of issues on different aspects of active network modeling and simulation. Our experiences indicate that it is better to have a simple and flexible language such as TSL for modeling network topologies. It is useful to have a clear delineation between the languages for developing the software modules for networking components and network modeling languages. For example, TSL and ANSE can be used to enable simulation of conventional networks. They can also be used for performing cosimulations using a mixture of conventional and active networking components. The flexibility and general-purpose design of ANSE can be used to enable interoperability between different types of models and even different simulators. An experimental evaluation of ANSE was presented in the paper. The



(a) Small models



(b) Large models

Figure 12. Comparison between sequential and parallel simulation times

experiments demonstrate that considerable improvements in the performance of the active network simulations can be achieved by employing parallel simulation techniques. The experiments, in conjunction with the diverse set of issues addressed by ANSE, highlight the effectiveness of the active networks simulation environment provided by ANSE.

9. Acknowledgment

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

10. References

[1] Paxson, V., and S. Floyd. 1997. Why we don't know how to simulate the Internet. In *Proceedings of the 1997 Winter Simulation Conference (WSC'97)*, December, pp. 44-50.

[2] Rao, D. M., K. Swaminathan, R. Radhakrishnan, P. A. Wilsey, and P. Alexander. 1998. An Active Networks Simulation Environment. In *Proceedings of the Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS '98)*, September, pp. 127-31.

[3] Tennenhouse, D. L., W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. 1997. A survey of active network research. *IEEE Communications Magazine* 35 (1): 80-86.

[4] Rao, D. M., and P. A. Wilsey. 1999. Simulation of ultra-large communication networks. In *Proceedings of the Seventh International*

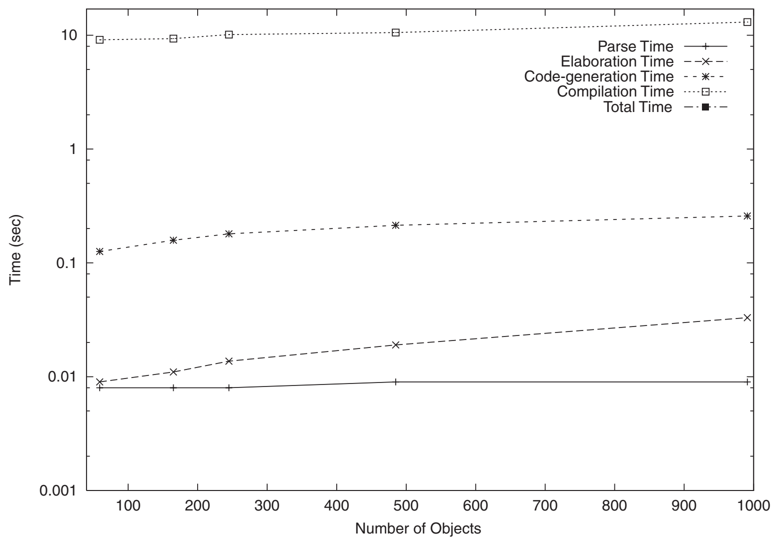


Figure 13. Total simulation time using the Web-based Simulation Framework (WSF)

Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, October, pp. 112-19.

[5] Riley, G. F., R. M. Fujimoto, and M. H. Ammar. 1999. A generic framework for parallelization of network simulations. In *Proceedings of the Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, October, pp. 128-35.

[6] Rao, D. M., and P. A. Wilsey. 2001. Modeling and simulation of active networks. In *Proceedings of the 34th Annual Simulation Symposium*, April.

[7] Swaminathan, K., R. Radhakrishnan, P. A. Wilsey, and P. Alexander. 1998. Large scale active networks simulation. In *International Workshop on Applied Parallel Computing (PARA98)*, edited by B. Kagstrom, J. Dongarra, E. Elmroth, and J. Wasniewski, vol. LNCS 1541, pp. 537-42. New York: Springer-Verlag.

[8] Rao, D. M. 2000. A network simulation toolkit. Master's thesis, University of Cincinnati.

[9] Robinson, S. 1997. Simulation model verification and validation: Increasing the users' confidence. In *Proceedings of the 1997 Winter Simulation Conference*, December.

[10] Vinoski, S. 1997. CORBA: Integrating diverse applications within distributed heterogenous environments. *IEEE Communications Magazine* 35 (2).

[11] Page, E. H., and R. E. Nance. 1994. Parallel discrete event simulation: A modeling methodological perspective. In *Proceedings of the ACM/IEEE/SCS 8th Workshop on Parallel and Distributed Simulation*, December, pp. 88-93.

[12] Rao, D. M., and P. A. Wilsey. 2000. Parallel co-simulation of conventional and active networks. In *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, August.

[13] Jefferson, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7 (3): 405-25.

[14] Radhakrishnan, R., D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. 1998. An object-oriented time warp simulation kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, edited by D. Caromel, R. R. Oldehoeft, and M. Tholburn, 13-23. New York: Springer-Verlag.

[15] Rao, D. M., R. Radhakrishnan, and P. A. Wilsey. 1999. FWNS: A framework for Web-based network simulation. In *1999 International Conference on Web-Based Modeling & Simulation (WebSim'99)*, edited by A. G. Bruzzone, A. Uhrmacher, and E. H. Page, vol. 31, pp. 9-14. Society for Computer Simulation.

[16] Rao, D. M., and P. A. Wilsey. 1999. An object-oriented framework for parallel simulation of ultra-large communication networks. In *Proceedings of the Third International Symposium on Computing in Object-Oriented Parallel Environments*, November.

[17] Zegura, E., K. Calvert, and S. Bhattacharjee. 1996. How to model an Internet work. In *Proceedings of IEEE INFOCOM*, April, pp. 594-602.

[18] Rao, D. M., R. Radhakrishnan, and P. A. Wilsey. 1999. Web-based network analysis and design. *ACM Transactions on Modeling and Computer Simulation*.

[19] Dahmann, J. S., R. M. Fujimoto, and R. M. Weatherly. 1997. The Department of Defense High Level Architecture. In *Proceedings of the 1997 Winter Simulation Conference*, December, pp. 142-49.

[20] Page, E. H., S. P. Griffin, and L. S. Rother. 1998. Providing conceptual framework support for distributed Web-based simulation within the High Level Architecture. In *Proceedings of SPIE: Enabling Technologies for Simulation Science II*, April.

[21] Hicks, M., P. Kakkar, J. T. Moore, C. A. Gunther, and S. Nettles. 1998. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM International Conference on Functional Programming Languages (SIGPLAN'98)*, May, pp. 86-93. Available: www.cis.upenn.edu/ switchware/papers/plan.ps

***Dhananjai M. Rao** is a PhD candidate at the Experimental Computing Laboratory in the Department of ECECS, Cincinnati, Ohio.*

***Philip A. Wilsey** is an associate professor at the Experimental Computing Laboratory in the Department of ECECS, Cincinnati, Ohio.*