Web-Based Network Analysis and Design

DHANANJAI MADHAVA RAO, RADHARAMANAN RADHAKRISHNAN, and PHILIP A. WILSEY
University of Cincinnati

The gradual acceptance of high-performance networks as a fundamental component of today's computing environment has allowed applications to evolve from static entities located on specific hosts to dynamic, distributed entities that are resident on one or more hosts. In addition, vital components of software and data used by an application may be distributed across the local/wide area network. Given such a fluid and dynamic environment, the design and analysis of high-performance communication networks (using off-the-shelf components offered by third party manufacturers) has been further complicated by the diversity of the available components. To alleviate these problems and to address the verification and validation issues involved in engineering such complex networks, a web-based framework for the design and analysis of computer networks was developed. Using the framework, a designer can explore design alternatives by constructing and analyzing configurations of the design using components offered by different researchers and manufacturers. The framework provides a flexible and robust environment for selecting and verifying the optimal solution from a large and complex solution space. This paper presents issues involved in the design and development of the framework.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications—*Distributed objects*; D.2.m [**Software Engineering**]: Miscellaneous—*Rapid prototyping*; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Distributed*

General Terms: Design, Languages

Additional Key Words and Phrases: Computer networks, formal specification and verification, parallel discrete event simulation, Web-based design and analysis

Support for this work was provided in part by the Advanced Research Projects Agency under contracts J-FBI-93-116 and DABT63-96-C-0055.
Authors' addresses: D. M. Rao, Department of ECECS, University of Cincinnati, POB 210030, Cincinnati, OH 45221–0030; email: dmadhava@ececs.uc.edu; R. Radhakrishnan, Department of ECECS, University of Cincinnati, PO Box 210030, Cincinnati, OH 45221–0030; email: ramanan@ececs.uc.edu; P. A. Wilsey, Department of ECECS, University of Cincinnati, POB 210030, Cincinnati, OH 45221–0030; email: paw@ececs.uc.edu.

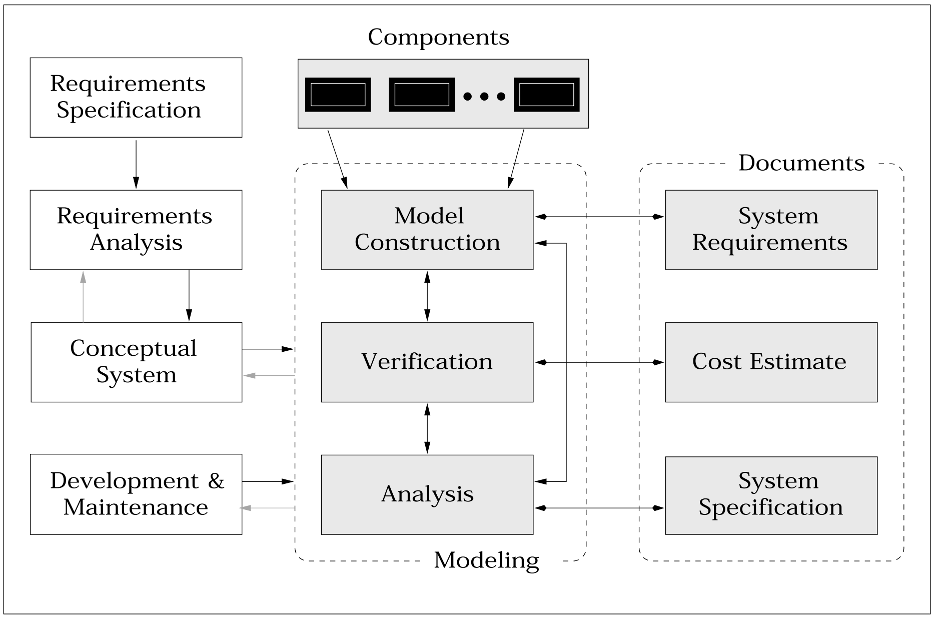

Fig. 1. System life cycle.

1. INTRODUCTION

Traditionally, network engineering has involved the design, production, and maintenance of complex networks that satisfy a set of cost and time constraints [Sage 1992]. Although the entire process of network engineering is a modeling activity [Pressman 1997], a large percentage of the modeling activity occurs predominantly during the design phase. A number of vital and useful characteristics about the proposed network result as by-products of the design phase of a system. These include network requirements, network specifications, network model(s), and cost-performance estimates. To determine this useful information, comprehensive models of the proposed network are developed during the design phase. The models are then used to ensure that the proposed network has the desired properties and to ensure that it meets the various requirements. In addition, to ensure that the models used to study and design networks are accurate and correct, verification and validation of these models must be carried out. However, the growing complexity of today's network designs mandates the use of simulations (parallel simulations in particular) for analysis and verification.

Figure 1 illustrates the phases involved in engineering computer communication networks. The phases involved in the engineering effort closely mirror those in the design of any conventional system. The design phase plays a critical role in the life cycle of a system [Pressman 1997]. Many of today's networking systems are composed using off-the-shelf components [Batory and O'Malley 1992] offered by third party manufacturers. Modeling the system usually involves composing the system from different subsystems and components. The model of the system is then verified and analyzed. During the cyclic modeling phase (Figure 1), various features and attributes of the proposed system are documented. Some of the important documents include system requirements, cost estimates, and system specification. Due to recent advances in technology, network designers now have a larger number of alternatives to consider and choose from. This multitude of options makes the process of choosing the right networking solution a complicated task. Exploring the different probable solutions is seldom done due to a number of hurdles faced, such as collection of appropriate information and data, model validation, modeling of message traffic, design and analysis of simulation runs, and time constraints [Law and McComas 1994]. Several of these tasks may require the expertise of experienced designers/developers who may or may not be available. In addition, there is the added pressure of short "design-to-implementation" time frames in the industry. Revisions in the design necessitate corresponding revisions in design-related documents (such as system specification, system requirements, and cost-time estimates [Pressman 1997]). This increases the time and cost of the design to implementation lifecycle.

The growing complexity of software systems has also required the reuse of software components as reinvention of extant technology is not affordable [Batory and O'Malley 1992]. Although considerable research has been carried out in the areas of composeable systems and reusable components [Penix et al. 1998], the software models developed for simulation based verification and analysis are seldom reused. There are several reasons for this [Page et al. 1998]. Some of the dominant hurdles faced are: (i) models developed for simulation by manufacturers and network designers are confidential (giving rise to intellectual property issues); (ii) the models may not be portable or interoperable [Vinoski 1997] (lack of a standard modeling language); and (iii) the models may not be readily available or accessible [Page and Nance 1994]. Researchers have identified that the World Wide Web (WWW) provides an excellent backbone to enable sharing of information and data [Fishwick 1996]. However, the complex interaction between components for modeling and simulation [Rao et al. 1999] render the raw WWW services insufficient. To address these issues a distributed, rapid prototyping environment for designing, analyzing, and deploying networked systems is required.

2. FWNS: FRAMEWORK FOR WEB-BASED NETWORK SIMULATION

In an effort to build a tool that eases the design, analysis, and verification of computer networks, a web-based framework for networks engineering called FWNS was developed. To simplify the use and maintenance of the framework, a component-based strategy was employed in the development of FWNS. Figure 2 illustrates the overall architecture of FWNS. Although flow of control and data in the system involves the interaction of a number of distributed components, the FWNS system can be easily described by functionality

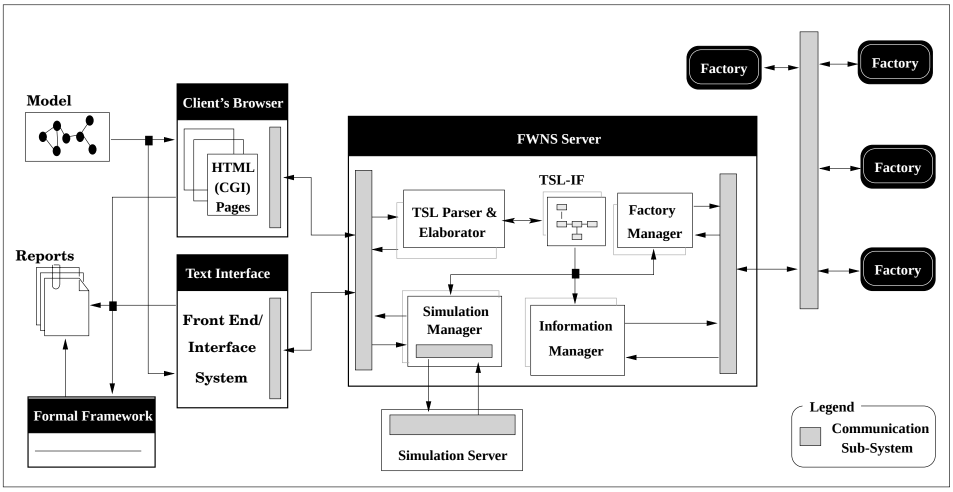

Fig. 2. Architectural overview of FWNS.

of its major components. Specifically, there are six major components (delineated by their functions) that together define the architecture of FWNS. Since the functions of these components can, at times overlap, the components are not explicitly highlighted in Figure 2. The six components are as follows: *(a)* the I-O Component, *(b)* the Topological Specification Language (TSL) Component, *(c)* the Assembly Component, *(d)* the Simulation Management Component, *(e)* the Communication Component, and *(f)* the Formal Framework Component.

As the name suggests, the I-O component is responsible for handling the input and output to/from the framework. The user interacts with the framework via the interfaces provided by the I-O component. The primary input to the framework is a detailed description of the communication network. A special input format called the Topology Specification Language (TSL) was designed and implemented to ease the specification of large and complex networks. The I-O component and the TSL component are together responsible for handling, parsing, and processing the network model descriptions written in TSL. Since the rest of the framework is dependent on the output of the I-O component, Section 3 describes the issues involved in the construction of the I-O and the TSL components.

The remainder of this paper is organized as follows. Section 4 describes the assembly component which uses the processed input information to build simulations. The primary function of the assembly component is to assemble the software components needed to generate the various derivables of FWNS. The derivables include data and statistics from the simulation, formal specifications of the network, and various documents and reports. The primary responsibility of the simulation management component (described in Section 5) is to control and coordinate the distributed simulation. Section 6 details the communication component which is responsible for providing the necessary communications infrastructure for the different distributed components of FWNS. The formal specifications that are generated by the assembly component are directed to a separate formal framework that uses the specifications for verification purposes. Section 7 describes the formal framework component of FWNS. The experiments conducted with the FWNS framework are detailed in Section 8. Concluding remarks along with future work are presented in Section 9.

3. THE I-O INTERFACE

Modeling complex networks involves specification of the topology of the network, description of the nodes in the network, source of simulation modules, and a number of parameters. The data for large networks can be voluminous. Flow of information to and from the framework need to be regulated to meet the designers requirements. The user may choose to employ a CGI (Common Gateway Interface) based HTML interface or a simple textual front end to interact with FWNS. The components used to handle the various interactions with the user along with the input parser (the TSL parser is shared between the I-O component and the TSL component) constitute the I-O component.

The I-O component is a distributed subsystem of FWNS. As illustrated in Figure 2, the I-O interface to the system is at the user's end. The user may choose a simple command-line style or a HTML-CGI based interface to the system. Depending on the choice, the corresponding FWNS modules is installed. The input system also needs to interact with the TSL parser. The TSL parser (housed in the FWNS server as illustrated in Figure 2) parses the input model description into an intermediate form (TSF-IF). The I-O component also supports simple error reporting mechanisms. This mechanism is used by the parser to report errors that may arise while processing the input configuration. To optimize the interaction between a number of users, a new TSL parser thread is invoked for every configuration submitted to the framework. This enables a number of users to concurrently work with the FWNS server.

3.1 The Topological Specification Language

The Topological Specification Language (TSL) Component is responsible for the parsing and processing of network descriptions written in TSL. This component encapsulates all TSL-related functionality. When TSL input descriptions are submitted to the TSL parser (housed in the FWNS server), it parses the network descriptions into the TSL intermediate form (TSL-IF). The TSL-IF is then used to construct a simulation. Since the TSL description is the source template for the simulation, the design and implementation of TSL is of considerable importance to the framework. In this section, we review design and implementation issues in the development of a network topological specification language such as TSL.

The two main constituents of a TSL description are: (i) information about the topology of the network model; and (ii) definition and location of corresponding software components of the network model. In TSL, the

Table I. TSL Grammar

```
topology_label: IDENTIFIER;
tsl_topology:
        {topology_label }object_definition_section
                        object_instantiation_section net_list_section;

-— OBJECT DEFINITION RULES --------------------

object_name: IDENTIFIER;
object_definition:
        object_name COLON url {parameter }SEMI_COLON;
object_definition_section:
        OPEN_FLOWER ( object_definition )*CLOSE_FLOWER;

--- OBJECT INSTANTIATION RULES ------------------

object_label: IDENTIFIER;
object_instantiation:
        object _label COLON object_name
                {NUMBER }{parameter }SEMI_COLON;
object_instantiation_section:
        OPEN_FLOWER ( object_instantiation )*CLOSE_FLOWER;

--- NET LIST RULES------------------------------

net-list_label: IDENTIFIER;
net_list:
        net-list_label COLON ( object_label )+ SEMI_COLON;
net_list_section:
        OPEN_FLOWER ( net_list )* CLOSE_FLOWER;

--- OTHER RULES--------------------------------

factory: IDENTIFIER ( DOT IDENTIFIER )*;
file_name: IDENTIFIER ( DOT IDENTIFIER )*;
url: IDENTIFIER ( DOT IDENTIFIER )*
                {COLON NUMBER DOT factory };
parameter:
        QUOTE ( NUMBER |IDENTIFIER |STRING )* QUOTE;
include_clause:
        INCLUDE QUOTE file_name QUOTE SEMI_COLON;
design_file:
        ( include_clause )* ( tsl_topology )+ ENDFILE;
```

various software components used to define a network topology are specified in the *object definition section*. The actual nodes that constitute the network along with the other components of the system being modeled are defined in the *object instantiation section*. The connectivity information between the various components is described in the *net-list section*. Table I illustrates the TSL grammar.

The object definition section is used to define a set of logical objects that will be used in the design. A set of object definitions constitute this section. Each object definition consists of a `object_name`, a Universal Resource

Locator (`URL`), and an optional list of parameters. The `object_name` associated with each object definition should be unique in a topology description. The `URL` associated with each object definition provides detailed information on the location of the simulation module (or factory1[1]) to be associated with each component of the design instantiated using the `object name`. The information contained in the URL consists of: (i) the IP address of the machine hosting the desired factory with an optional port number, (ii) the hierarchy of the factory and subfactory names housing the actual simulation module, and (iii) the name associated with the actual simulation module. An optional list of parameters may be specified with each object definition. These parameters may be used by the simulation module to tailor itself to the needs of the user. To specify an object's definition, the user must know the location of the different factories and possess information about the layout of the factory such as the name associated to the object and the hierarchy of subfactories (if any) that are responsible for the object. In addition, the user must also be aware of the syntax and semantics of the parameters expected by the different objects in the various factories. For this reason, it is the responsibility of the factory designers and developers to provide information about the location of the factory (such as the IP address of the server(s) that host the factory along with any necessary port numbers for communication) and the layout of the factory. The syntax and semantics of the parameters expected by each object must also be specified. This will enable the user to choose the appropriate object to be used in the design. The optional set of parameters can be selected by the user to tailor the object (within the scope of the object) to meet his requirements. A good place to put these details would be along with the advertisements for the various components on the WWW. Interested users can use these details for modeling and analysis. Optionally, factories can register themselves with "well known" FWNS servers increasing their visibility.

The object instantiation section consists of a list of object instantiations that is used to specify the actual set of components involved in the topology specification. Each object instantiation consists of an `object_label`, a `object_name`, and an optional set of parameters. Each `object_label` defines an entity in the model and should be unique in a topology specification. The `object_name` defines the actual software module that must be instantiated for simulation. Every `object_name` used must be defined in the object definition section. In other words, the `object_name` associates an object definition with every object instantiation. The object definition provides all the necessary information to instantiate an object. The optional set of parameters in each object instantiation can be used to further adapt each object instance to meet the specifications of the components.

The net-list section consists of a number of net-lists that are used to describe the connectivity information between the various components in

[1]A "factory" provides an uniform interface for creating families of related or dependent objects without specifying their concrete classes [Gamma et al. 1994].

Table II. Example of a TSL Configuration

```
NetworkConfiguration {
trGen: bc.ececs.uc.edu:2048.NetworkFactory.TrafficGenerator "Normal 1 5 1000";
Node1 : bc.ececs.uc.edu:2048.NetworkFactory.Node "normal 1 0";
Node2 : bc.ececs.uc.edu:2048.NetworkFactory.Node "normal 0 5";
Router: bc.ececs.uc.edu:2048.NetworkFactory.SimpleRouter;
}

* {
viking : Node1;
grog, wiley : Node2;
trGen1, trGen2, trGen3 : trGen;
zeus, thor : Router;
}

* {
trGen1 : viking; trGen2 : grog; trGen3 : wiley;
viking : thor; grog : thor; wiley: zeus;
zeus : thor; thor : zeus;
}
```

the system. Each net-list entry consists of an `net-list_label` and a list of `object_labels`. The `net-list_labels` can be used in subsequent net-list entries to incrementally build connectivity information. The `topology_label` associated with each topology may also be used (to connect hierarchies of net-lists).

The configuration information provided to FWNS consists of a set of topologies where each topology has an unique `topology_label`. These labels can be used in the net-list section of subsequent topologies to construct hierarchical networks. An *include* clause is also supported by TSL to permit inclusion of topologies specified in other files. The user can specify small and simple subnetworks and combine them to build and specify larger and more complex networks. A short example of a sample topology in TSL format is shown in Table II. The network specifications in TSL format are parsed by a TSL parser into an intermediate form (IF).

TSL-IF forms the primary input to the other modules in the FWNS server. As shown in Figure 2, the *simulation manager* and the *information manager* use the TSL-IF for extracting information about the input topology. TSL-IF is designed to provide efficient access to related data from the various nodes. It closely reflects the structure of the TSL grammar. As the current implementation of TSL-IF is in C++, object-oriented techniques have been exploited to optimize TSL-IF's design and implementation. TSL-IF is also useful for performing other transformations and analyses [Willis et al. 1996] of the network topology. Generation of configuration information in TSL format can be automated to generate several different regular and irregular internetwork topologies [Zegura et al. 1996].

4. THE ASSEMBLY COMPONENT

The functionality needed for the assembly section is jointly provided by the *simulation manager*, the *information manager*, the *factory manager*, and the distributed factories. As illustrated in Figure 2, these components are not centralized but distributed over the WWW and hence this section is by itself a distributed subsystem. The components constituting the assembly section can be broadly classified into two categories; namely:

—**Simulation-oriented components**: The simulation manager, the factory manager, and the distributed factories constitute the simulation-oriented components. These components are responsible for collaboratively building a distributed simulation. As illustrated in Figure 2, the former two are part of the FWNS server while the factories are distributed across the Internet.

—**Information-oriented components**: The information manager, the factory manager, and the distributed factories constitute the information-oriented components. These components are responsible for collating information about and from the distributed objects to generate specifications and documents.

While the various components of the assembly section possess differing functionalities, the primary input to these components is derived from the elaborated TSL-IF generated by the input section of the framework. The TSL-IF not only provides a simple structured input to the components but also standardizes the interfaces and eases interoperability between the components. The following subsections attempt to illustrate the issues involved in the design and implementation of the various components of the assembly section.

4.1 Simulation Manager

The simulation manager provides the support needed for composing a simulatable model from the distributed factories. To optimize the execution of the FWNS server, a unique simulation manager is spawned by the framework for each unique TSL configuration submitted. The simulation manager uses the elaborated TSL-IF generated by the TSL parser to construct the simulatable model. The simulation manager uses the factory manager to collaborate with the distributed factories to build the objects needed for simulation. Thus, the factory manager provides a convenient interface to interact with the distributed factories.

The simulation objects built by various factories may be local or remote objects. Remote objects must be migrated from the factory to a suitable simulation subsystem. The simulation manager is also responsible for assigning remote objects to suitable simulation subsystems. The simulatable model may be built by the simulation manager in one or two passes of the elaborated IF. If a two pass methodology is adopted, then in the first pass the existence and compatibility of the different objects in the various factories is verified. During the second pass, the actual objects are instantiated from the factories. In a single pass methodology, the objects are verified and instantiated and on errors the simulation manager undoes its work. Although the two pass technique involves more time and effort than the single pass method, a lesser amount of work needs to be undone when errors are detected. Hence, during the initial phases of the design process, the two phase approach is preferred as errors in the choice and availability of components can be quickly detected and rectified. The simulation manager plays a vital role in building the simulation. Once the simulatable model is successfully composed, the simulation manager transfers control over to the simulation subsystem modules that control and coordinate the distributed simulation.

4.2 Information Manager

The information manager provides a variety of outputs depending on the control information provided by the FWNS server. The basic functionality of the information manager is to collate information about the input configuration from the distributed object factories. The control information that determines the data collected is provided by the framework, which in turn derives the information from the user. The elaborated TSL-IF generated by the TSL parser forms the primary input to the information manager modules. Communication with the distributed factories is done via the factory manager. Similar to the factory manager, the framework spawns off an unique information manager thread for each session. Based on their functionality, the modules constituting the information manager can be grouped into two categories:

—**Report generation modules**: These modules are responsible for collating and organizing data from the distributed factories for documentation and report generation purposes. The framework provides simple templates for the various outputs. These templates are in turn obtained from the user.

—**Specification modules**: Collation and organization of data from the distributed factories for constructing formal specifications is accomplished by these modules. The elaborated TSL-IF provides all necessary information needed by these modules. The style of specification depends on the different formal mechanisms supported by the framework. Separate submodules are used to handle the intricacies of each unique mechanism. The current implementation supports construction of specifications for the Prototype Verification System (PVS) [Owre et al. 1992].

The outputs generated by the information manager play a critical role in assisting the designers of the system. Documents pertinent to the topology being modeled can be easily generated by the user. It eases the process of revising the documents due to changes in design. Depending on the information generated, the outputs may be routed to the user or to the *formal framework* module using the I-O section modules. Thus, the information manager plays a critical role in the design cycle of engineering a networking system.

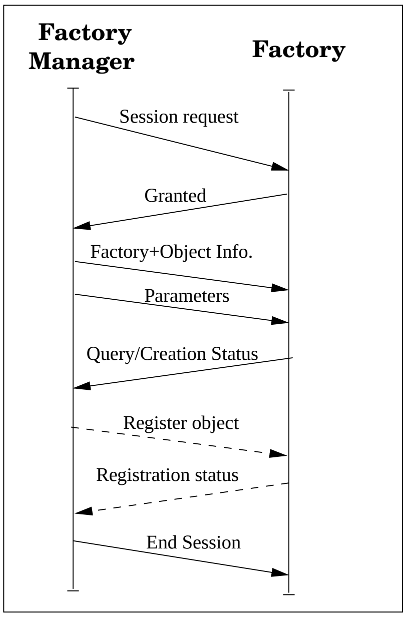

Fig. 3. Handshaking between a factory and the factory manager.

4.3 Factory Manager

The factory manager provides the communication support for the various modules of the FWNS server. It also helps to insulate the various modules from the intricacies involved in communicating with the distributed factories. The communication subsystem provides the channels between the factory manager and the distributed factories. A well defined protocol is used for communications between the manager and the factories.

Figure 3 illustrates some of the important phases in the communication process. The factory manager initiates communication with the factories. Once the channel is successfully established via the communication subsystem, it then forwards the object information along with the parameters to the factory. Based on the service requested, the factory processes the parameters and replies to the query. The asynchronous communication between the manager and the factories is complex and slow. To improve the efficiency, the FWNS server associates a unique factory manager thread with each session of the simulation manager and information manager.

One of the important extensions planned for the factory manager is to enable simple object queries. The user may specify attributes about the objects and the factory manager queries the “known” factories to locate components matching the users requirements. The query process can also be extended to perform sophisticated knowledge-based component retrievals [Penix et al. 1998].

4.3.1 *Factories*. The factories play a pivotal role in providing an uniform interface to different simulation entities and simplifies their maintenance and

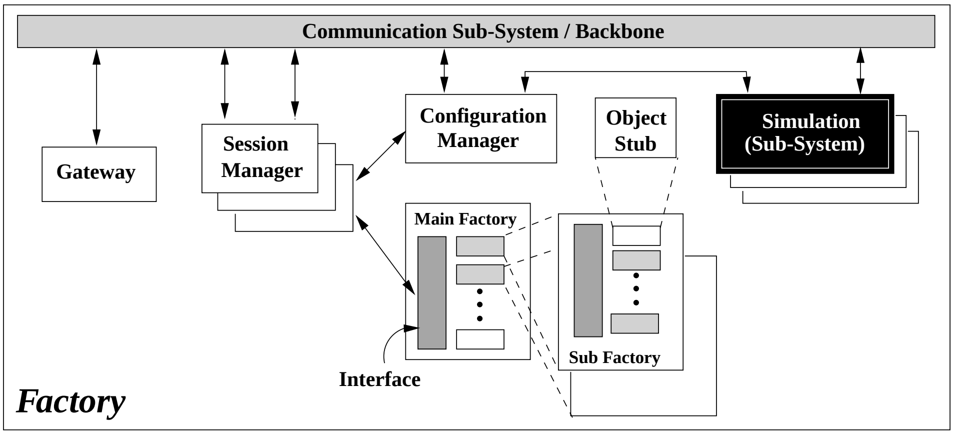

Fig. 4. Layout of a factory.

generation. Figure 4 illustrates the layout of a typical FWNS factory. The initial handle to a factory is provided by the *gateway* module. The gateway module hooks on to a specified address via the communication backbone and processes the initial requests from different simulation managers. This address is used to locate and communicate with the factory. The communication backbone provides an uniform interface to the actual communication infrastructure and the web.

The *session manager* module is responsible for interacting with a factory manager to handle queries, create objects, and to set up the simulation. A number of distinct distributed simulation managers may communicate with the same factory, via the gateway module, to setup various objects for simulation. For each such session, the gateway module spawns a new session manager thread. The different session manager threads interact and may share the same main-factory and sub-factories. Sharing the main-factory and sub-factories considerably reduces the memory requirements of a factory. The gateway model also assigns a dedicated channel to each session manager thread for communication with the backbone. This style of implementation isolates the communication channels of each session (and simulation) from others reducing the traffic on each channel. The design also permits each session to use its own dedicated channels for communication. The session manager also handles some of the specific semantics of the simulation engine.

The *configuration manager* module provides the interface between the session manager and the simulation subsystem. The simulation subsystem constitutes the actual simulation engine of the factory. The simulation subsystem also provides a uniform interface for the different simulation engines. The factory may have simulation engines that cannot handle remote (distributed over the web) objects and hence the users may be forced to instantiate all the objects from the same local factory. The factory specifications should provide the user with such details. The simulation subsystem is optional in a factory. The factory could merely build objects for simulation (for example, in Java based byte code) and the objects may be migrated to a remote *simulation server* associated with the FWNS server for actual simulation.

A factory can be built from sub-factories, sub-sub-factories or from *object stubs*. This design style provides a convenient hierarchical organization for large factories. The object stubs are the atomic components of an object factory. Object stubs provide information about the component and create the actual simulation objects. The information provided by the stubs are related to security issues, properties of the physical component (such as cost and speed) and properties of the simulation object (such as memory requirements or simulation engine requirements). These details can be used to automate the process of selecting simulation objects based on user defined criteria. This information is also used by the information manager for generating specifications and documents. The information about the locality of the actual simulation object is also contained in the stub. Objects could be *local* or *remote*. The factory is expected to be capable of simulating local objects using its simulation subsystem. The remote objects have to be migrated to a suitable simulation subsystem for performing the actual simulation. The stubs provide a conceptual link between the simulation object and the physical component. The factory provides a convenient and extensible mechanism for managing the different models. Object-oriented techniques have been successfully employed to develop basic reusable factories that can be easily extended to build new sub-factories. Since the factories are fully distributed, large simulations can be set up completely asynchronously in parallel, thereby reducing the setup time and costs. Local objects can be used to retain proprietary implementations. The factory also provides interfaces to query and search for specific components that meet certain user defined criteria.

5. THE SIMULATION MANAGEMENT COMPONENT

The different simulation subsystems in the factories along with the simulation objects and the simulation server associated with the FWNS server constitute this section. The primary responsibility of the simulation management component (SMC) is to control and coordinate the distributed simulation. The simulations are discrete-event simulations and may be parallel. *Conservative* [Bryant 1979; Chandy and Misra 1979] or *optimistic* [Jefferson 1985] synchronization mechanisms may be used for the parallel simulations.

The *simulation manager* module present in the assembly section triggers the working of the SMC. Apart from any control information passed from the simulation manager this section has no primary inputs. The outputs generated from this section are routed back to the user using the various I-O section modules. The *simulation server* is an optional but an integral part of the framework. It is used for simulating remote objects created by the factories. The remote objects are migrated from the various factories to the simulation server by the simulation manager before the simulation is set up. The simulation server is responsible for providing the necessary infrastructure required to simulate the remote object. A set of simulation servers are used to handle remote objects of different kinds. If the simulation servers associated with the framework do not support a particular kind of simulation object, then the simulation manager aborts the construction process. For each simulation session started by the simulation manager, separate threads of the necessary simulation servers are created.

The current implementation of the simulation server is based on an optimistic synchronization mechanism. The WARPED [Radhakrishnan et al. 1998] parallel discrete-event simulator is currently used to provide the necessary distributed simulation infrastructure. WARPED is a Time Warp [Jefferson 1985] based parallel discrete event simulator. The simulation server supports simulation objects built using the WARPED API. Details on developing simulation objects using the WARPED API is available in the literature [Radhakrishnan et al. 1998]. Interoperation of different synchronization domains and modules based on different simulators are currently under study. Standardization of the basic event structures required in discrete event simulators [Dahmann et al. 199] has been proposed as an ideal solution to enable collaboration between different simulators. The last two pieces of information [Dahmann et al. 1997] are the `destination` [Radhakrishnan et al. 1998] and `time stamps` [Lamport 1978] of the various events. The intricacies involved in hardware formats (such as little endian vs. big endian) are currently handled by the corresponding communication subsystem modules.

Interoperation of synchronization mechanism and modules based on different time domains is also under study. Simple protocols to exchange time and synchronization mechanisms between the various distributed entities are being designed. The time management infrastructure present in the High Level Architecture [Fujimoto and Weatherly 1996] is also being considered. The simulation management section also includes the various supporting algorithms, such as Global Virtual Time (GVT) algorithms [Bellenot 1990] that may be required. The current implementation is based on the WARPED simulator. The modules in the simulation management section can be easily extended to incorporate other simulation related optimizations. Domain knowledge can also be effectively employed for further optimizations. The simulation management sections plays an important and critical role to enable the distributed simulation feature of FWNS.

6. THE COMMUNICATION COMPONENT

The modules that provide the fundamental communication backbone for the framework and simulation subsystem constitute the communication subsystem. Current implementations of the communication subsystems in FWNS are built around Berkeley sockets [Stevens 1992]. The subsystem can be built from any communication protocol, that guarantees error-free and First-In-First-Out (FIFO) message delivery. The communication subsystem

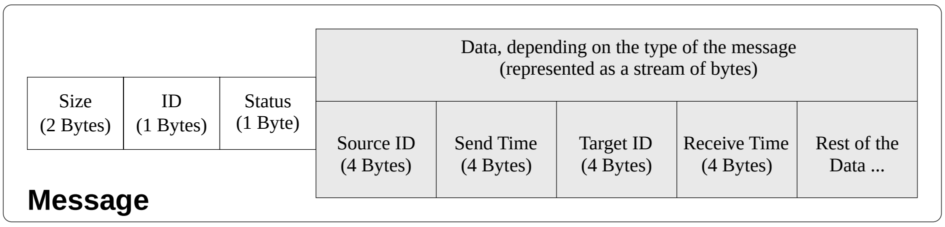

Fig. 5. Format of a FWNS Message.

modules not only insulate the various modules from the intricacies involved but also provide an uniform interface for communication between the various distributed components.

The basic unit of information transfer via the communication backbone is a *message*. The structure of the messages used in the framework is shown in Figure 5. All of the data in messages are represented in standard network format. Every message starts with 3 fields: (a) *size*: specifies the size of the message in bytes; (b) *ID*: specifies the type of the message; and (c) a *status* field. Depending on the *ID* field, the latter part of a message could be an event submessage consisting of the components shown in Figure 5. These messages are used by the simulation objects to exchange information. All other types of messages have a trailing body which is essentially a byte stream. These messages are used by the various components of the framework to interact with each other. Depending on the type of the message (as specified in the *ID* field) the *Data* field contains the specific pieces of information. The various communicating modules use a predefined communication protocol to exchange information.

The communication subsystem plays an vital role in the framework. It provides the different components a simple and transparent mechanism required for communication. It serves as an insulation from the intricacies of the underlying network. It provides a convenient filter to handle data conversions across formats (*e.g.*, conversion from big endian to little endian and vice versa) while operating on heterogenous platforms. It also provides a ideal point to implement any of the fault-tolerance that may be necessary in the system.

7. THE FORMAL FRAMEWORK

The growth in sophistication and complexity of software systems has necessitated the use of formal mechanisms to verify and validate them [Penix et al. 1998]. Automated formal methods are now employed to verify and validate designs. Solutions for verifying systems built by composition using automatic mechanisms have been studied [Penix et al. 1998]. Using formal methods, a designer can model the various aspects of a system and apply mathematical analysis/verification techniques. Formal methods help to provide higher levels of confidence in the system. The *formal framework* module supports the formal verification of the system models developed by using the FWNS environment. The modules involved in providing formal verification support constitute the formal framework.

The primary input to the the formal framework modules is collated by the *information manager* present in the FWNS server. Based on the user's specifications, the information manager selects the appropriate set of formal data from the various components in the topology. The formal data is routed back to the formal framework via the I-O section modules. The formal framework reorganizes the data to meet the requirements of the underlying formal language. The framework provides a default set of axioms and proven theorems that the specifications constructed from the user models are built on. The process of verifying the specification is currently not fully automated. The user is responsible for interacting with a theorem prover to verify the generated design specifications. The verification process also proves useful to highlight the requirements of system being modeled.

The current implementations of the formal framework in FWNS are based on the Prototype Verification System (PVS) [Owre et al. 1992]. PVS uses an axiomatic style for developing specifications. The usage of PVS language and theorem prover are available in the literature [Owre et al. 1993]. An existing set of axioms and theorems are provided with the framework that are used to compose specifications. The system requirements reflect as Type Check Conditions (TCC)s [Owre et al. 1993] during the verification process. Discharging the TCCs is analogous to providing the necessary infrastructure to the model. The process of extracting requirement specifications from the TCCs can be automated.

The formal framework plays an important role in the design and verification of the network models and components. Formal mechanisms are not only useful to extract inconsistencies in design but also help in building system requirements and to increase the confidence in the design. A number of extensions to the framework such as model checking and support for various formal languages are planned. Although the formal framework module is still in its infancy, it holds tremendous potential for automating the verification process, and is a necessary and integral component of FWNS. Preliminary studies using the PVS automated theorem proving tool have resulted in the formal specification and verification of the Time Warp synchronization protocol [Chernyakhovsky et al. 1999; Frey et al. 1999; Frey 1998].

8. EXPERIMENTS

The factory infrastructure of FWNS was used to develop a networking factory consisting of basic network components. The essential components were a traffic generator, a network node, and a router. The rate at which the packets were generated by the traffic generator was determined by a set of predefined random distributions. The user could tailor the type of traffic generated by specifying the appropriate distribution and corresponding parameters such as the *mean* and *variance* for the distribution. The

Table III. Details of Network Models Used in Experiments

S. no.	No. of nodes	No. of edges	Average degree	Total simulation objects	Lines of TSL
1	10	10	2.0	21	72
2	20	19	1.9	43	149
3	30	32	2.133	64	216
4	40	41	2.05	85	280
5	50	55	2.0	106	347

node represents a passive node on the network. The traffic generator must be used to drive the node to generate data. The size of the packets and their destination are based on a user-specified distribution. The nodes must be connected to a router to send and receive the data packets. The routers employ broadcasting to forward the data packets from one router to another. The components also generate statistics, collated during simulation, such as the number of packets generated, average packet size, and details on router traffic. The parameters to the various components, used to model a network, are provided by FWNS from their TSL specifications.

The simulation objects generated by the factory were local. Hence, the necessary support for parallel simulation was also built into the factory. The infrastructure for parallel simulation was enabled using WARPED [Radhakrishnan et al. 1998], an optimistic parallel simulation kernel. The various components in the network factory adhered to WARPED's application program interface (API). In WARPED, simulation objects or logical processes (LPs) are grouped into entities called *clusters* [Radhakrishnan et al. 1998]. The session manager module was adapted to handle the specifics of the simulation engine such as instantiating an LP and registering the requested simulation objects with the LP. A customized communication subsystem interface was developed to integrate WARPED with FWNS.

The various network configurations used in the experiments were generated using GT-ITM tools [Zegura et al. 1996]. GT-ITM tools provide techniques for generating various graphs to model real world network topologies. The network topologies are built on top of the Stanford GraphBase (SGB), a platform of data structures and routines for representing and manipulating graphs. Details on the various networking topologies, generated using GT-ITM, used in the experiments are shown in Table III. Each node in the network was associated with a traffic generation object. Hence the total number of objects in the simulation (as shown in Table III) is greater than the number of network nodes. A *SGB-to-TSL* conversion tool was developed to ease transmogrification of the graphs, generated by GT-ITM, from SGB format to TSL specifications. The generated TSL specification were suitably modified to utilize components from the network factory with appropriate parameters inorder to enable simulation.

The simulation experiments were performed on a network of shared memory multi-processor (SMP) workstations. Each workstation consisted of dual Pentium-Pro processors (166 MHz), with 128 MB of RAM, running

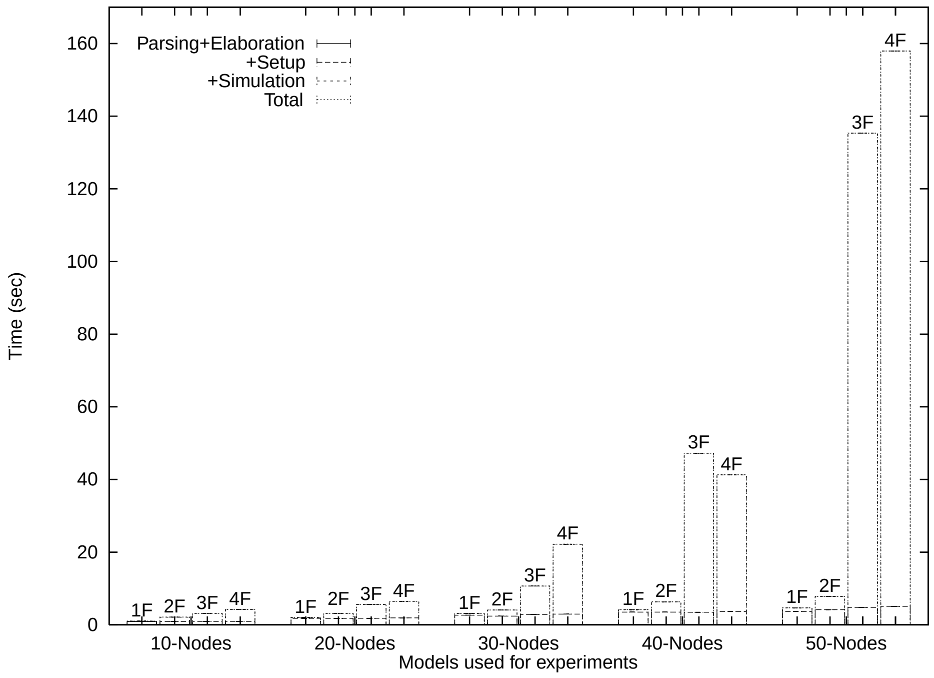

Fig. 6. Timings obtained from experiments.

Linux version 2.0.34. The workstations were connected by fast Ethernet. The parallel simulation studies were conducted by deploying two factories per workstation and employing components from the various factories by suitably modifying the TSL specifications. The traffic generators, associated with each node, were parameterized to trigger the nodes at a constant rate to generate 50 packets of size 64 bytes each. The text interface to FWNS was used in the experiments.

The timings obtained from the various experiments are illustrated in Figure 6. The cumulative time for parsing, elaboration, setting up the simulation, and simulation for the various configurations are shown. The labels (*nF*) associated with each timing bar indicate the number of factories used in the experiment. The configurations that involve more than one factory indicate the parallel and distributed simulation instances. As shown in Figure 6, the time required for the parallel simulation instances are higher due to communication and synchronization overheads. The timings for the experiments conducted with 2 factories is considerably lower, when compared to the 3 and 4 factory configurations, since the factories deployed on the same workstation were used. The time required is lower because communication between factories on the same workstation is much faster than communication between workstation due to network delays. The 4 factory simulation of the network model with 40 network nodes was faster than the 3 factory model. A scrutiny of the simulations indicated the increased performance was due to effective partitioning (allocation of objects to factories) of the various simulation objects.

Although, random partitioning was adopted in the experiments, the results indicate that considerable benefits can be accrued by using ideal partitions. Such partitions can be achieved by strategically using components from the various factories rather than on a purely random basis.

Other uses of FWNS include a demonstration of active network simulations across the web during the 1998 DARPA ITO sponsored Active Nets workshop [ANW'98 1998]. An ultra large scale remote simulation of more than two million active network nodes using the latest version of FWNS is planned for this year's Active Nets workshop. In addition, computer science graduate students at the Virginia Polytechnic Institute and State University (Virginia Tech) are currently using FWNS for a networking project.

9. CONCLUSIONS AND FUTURE WORK

The growing complexity of the different network components has made the design and construction of effective networks a complex task. Many of the networks are built from third party supplied equipments. To facilitate analysis of the different options using simulations and in order to further promote their products, the manufacturers need to provide greater access to their simulation models. The FWNS provides a flexible framework for integrating these diverse needs. The effectiveness in protection of proprietary rights, component retrieval methodology, and effectiveness of simulation make the solution ideal. Although FWNS provides a practical solution to the problem, more research is needed to handle further diversity of simulation models. Different synchronization strategies are used by different researchers, and a universal mechanism to enable all the different strategies to coexist in the same simulation is needed. More complex models at various levels of abstraction needed to be developed to study the effectiveness of the framework. The framework for web-based network simulation provides an excellent infrastructure that will prove to be a stepping stone for an ultra-scale heterogenous simulation machine that is fully distributed over the whole Internet.

REFERENCES

ANW. 1998. Active nets workshop: (ANW '98). (http://www.dyncorp-is.com/darpa/meetings/anets98jul/)

BATORY, D. AND O'MALLEY, S. 1992. The design and implementation of hierarchical software systems with reusable components. *ACM Trans. Softw. Eng. Methodol. 1*, 4 (Oct.), 355–398.

BELLENOT, S. 1990. Global virtual time algorithms. *Trans. Soc. Comput. Simul.* (Jan.), 122–127.

BRYANT, R. E. 1979. Simulation on a distributed system. In *Proceedings of the 16th Conference on Design Automation* (DAC '79, San Diego, CA, June 25-27). IEEE Computer Society Press, Los Alamitos, CA, 544–552.

CHANDY, K. M. AND MISRA, J. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng. 5*, 5 (Sept.), 440–452.

CHERNYAKHOVSKY, V., FREY, P., RADHAKRISHNAN, R., WILSEY, P. A., ALEXANDER, P., AND CARTER, H. W. 1999. A formal framework for specifying and verifying time warp optimizations. In *Proceedings of the Fourth International Workshop on Formal Methods for Parallel Programming: Theory and Applications* (FMPPTA'99). Springer-Verlag, New York, NY, 1228–1242.

DAHMANN, J. S., FUJIMOTO, R. M., AND WEATHERLY, R. M. 1997. The Department of Defense high level architecture. In *Proceedings of the Winter Conference on Simulation* (WSC '97, Atlanta, GA, Dec. 7–10), S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, Eds. ACM Press, New York, NY, 142–149.

FISHWICK, P. A. 1996. Web-based simulation: Some personal observations. In *Proceedings of the Winter Conference on Simulation* (WSC '96, Coronado, CA, Dec. 8–11), J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, Eds. ACM Press, New York, NY, 772–779.

FREY, P. 1998. Protocols for optimistic synchronization of Mixed-mode simulation. Ph.D. Dissertation. University of Cincinnati, Cincinnati, OH.

FREY, P., RADHAKRISHNAN, R., WILSEY, P. A., ALEXANDER, P., AND CARTER, H. W. 1999. An extensible formal framework for the specification and verification of an optimistic simulation protocol. In *Proceedings of the 32nd Hawaii International Conference on System Sciences* (HICSS'99, Jan.). Sony Electronic Publishing Services.

FUJIMOTO, R. M. AND WEATHERLY, R. M. 1996. Time management in the dod high level architecture. In *Proceedings of the Tenth Workshop on Parallel and Distributed Simulation* (PADS '96, Philadelphia, PA, May 22–24), W. M. Louchs and B. R. Preiss, Chairs. IEEE Computer Society Press, Los Alamitos, CA, 60–67.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Longman Publ. Co., Inc., Reading, MA.

JEFFERSON, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst. 7*, 3 (July), 404–425.

LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM 21*, 7, 558–565.

LAW, A. M. AND MCCOMAS, M. G. 1994. Simulation software for communications networks: The state of the art. In *Proceedings of the Fifth IEEE Workshop on Computer-Aided Modeling, Analysis, and Design of Communication Links and Networks* (CAMAD'94, Mar.). IEEE Computer Society, Washington, DC, 44–50.

OWRE, S., RUSHBY, J. M., AND SHANKAR, N. 1992. PVS: A prototype verification system. In *Proceedings of the 11th Conference on Automated Deduction* (CADE'92, Saratoga, NY, June), D. Kapur, Ed. Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 748–752.

OWRE, S., SHANKAR, N., AND RUSHBY, J. M. 1993. *User Guide for the PVS Specification and Verification System*. SRI International, Menlo Park, CA.

PAGE, E. H., GRIFFIN, S. P., AND ROTHER, L. S. 1998. Providing conceptual framework support for distributed web-based simulation within the high level architecture. In *Proceedings of the SPIE: Enabling Technologies for Simulation Science II* (SPIE, Orlando, FL, Apr 13-17). 287–292.

PAGE, E. H. AND NANCE, R. E. 1994. Parallel discrete event simulation: A modeling methodological perspective. In *Proceedings of the ACM/IEEE/SCS 8th Workshop on Parallel and Distributed Simulation* (Dec. '94). 88–93.

PENIX, J., MARTIN, D., FREY, P., RADHAKRISHNAN, R., ALEXANDER, P., AND WILSEY, P. A. 1998. Experiences in verifying parallel simulation algorithms. In *Proceedings of the Second Workshop on Formal Methods in Software Practice*, M. Ardis, Ed. ACM Press, New York, NY, 16–23.

PRESSMAN, R. S. 1997. *Software Engineering: A Practitioner's Approach*. 4th ed. McGraw-Hill, Inc., New York, NY.

RADHAKRISHNAN, R., MARTIN, D. E., RAO, M., AND WILSEY, P. 1998. An object-oriented time warp simulation kernel. In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments* (ISCOPE'98), D. Caromel, M. Tholburn, and R. R. Oldehoeft, Eds. 13–23.

RAO, D. M., RADHAKRISHNAN, R., AND WILSEY, P. A. 1999. FWNS: A framework for web-based network simulation. In *Proceedings of the International Conference on Web-Based Modelling and Simulation* (WebSim'99, Jan.), A. Uhrmacher, A. G. Bruzzone, and E. H. Page, Eds. Society for Computer Simulation, San Diego, CA, 9–14.

SAGE, A. P. 1992. *Systems Engineering*. Wiley series in systems engineering. John Wiley and Sons, Inc., New York, NY.

STEVENS, W. R. 1990. *UNIX Network Programming*. Prentice-Hall, Inc., Upper Saddle River, NJ.

VINOSKI, S. 1997. CORBA: Integrating diverse applications within distributed heterogenous environments. *IEEE Commun. Mag. 35*, 2 (Feb.), 46–57.

WILLIS, J. C., WILSEY, P. A., PETERSON, G. D., HINES, J., ZAMFRIESCU, A., MARTIN, D. E., AND NEWSHUTZ, R. N. 1996. Advanced intermediate representation with extensibility (AIRE). In *Proceedings of the VHDL Users' Group Fall 1996 Conference on VHDL* (Oct.). 33–40.

ZEGURA, E., CALVERT, K., AND BHATTACHARJEE, S. 1996. How to model an internetwork. In *Proceedings of the IEEE Conference on INFOCOM* (San Francisco, CA, Apr.). IEEE Computer Society Press, Los Alamitos, CA, 594–602.

Received: April 1999; revised: December 1999; accepted: January 2000