Adapting Protocols to Massively Interconnected Systems

Dennis Kafura and Marc Abrams

TR 91-27

September 9, 1991

Adapting Protocols to Massively Interconnected Systems

Dr. Dennis Kafura
Department of Computer Science
Virginia Tech
Blacksburg, VA 240610-0106
kafura@vtopus.cs.vt.edu
703/231-5568
Fax: 703/231-6075

Dr. Marc Abrams
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061-0106
abrams@vtopus.cs.vt.edu
703/231-8457
Fax: 703/231-6075

Abstract

This paper describes ongoing research focused on two critical problems posed by the interconnection of a massive number of computer systems. The interconnection may be achieved through wide area or local area networks. The two problems considered in this research are:

- performance analysis of the protocols used in an internetwork connecting thousands to millions of nodes, and
- application development in a massively distributed, heterogeneous environment where components implemented in different programming languages must be integrated and/or reused.

The performance analysis problem is addressed by employing large-scale parallel simulation, extended finite state machines and objected-oriented simulation techniques. The approach to solving the application development problem is based on an environment which exploits the synergism between object-oriented programming and layered communication protocols (specifically, OSI).

I. Introduction

In this paper we consider two important issues in the design of complex, computer based systems which impose far more stringent requirements for communication than for computation. Such systems,termed "massively interconnected systems," may employ both wide area and local area networks. The two issues are:

- how can the performance of existing and proposed communication protocols be assessed in light of the limitations of direct experimentation and analytical techniques, and
- how can applications be engineered so that an application is insensitive to the distribution of its components across heterogenous processor architectures and where the components may be implemented in differing programming languages.

In the remainder of this section the particular importance of these two issues to massively interconnected systems will be described. Subsequent sections provide an overview of the research we are conducting (Sections 2 and 3) and the current status of this work (Section 4).

1.1 Performance Analysis of Protocols for Massively Interconnected Systems

Development of any new technology is hampered without tools to predict system behavior. While tools exist to predict the behavior of individual computer communication networks, network architects desperately lack tools to predict the behavior of the combination of individual networks into a single, unified internet. A problem of immense importance is to identify what protocols best manage the data pathways of an internet. Often details of the algorithms and methods used to implement a protocol and the parameters used with the protocol dramatically affect the data capacity of the internet. Therefore network designers need to model the internet to evaluate proposed protocol algorithms and parameter settings before they are installed and unleashed on the world. The need to model is growing more critical because the rate at which data can be transmitted over a network soon will increase a hundred-fold. Increased data communication rates will permit new uses of geographically distributed computers. For example, video, voice, and data could be transmitted over a common network. Desktop workstations could display graphic images from scientific computations on remote supercomputers.

Better internet prediction tools could provide scientifically sound answers to topics of debate among internet architects. For example, standardization bodies do not know if TCP [ISI81] can efficiently handle gigabit per second transmission rates in future internet backbones. Many viewpoints exist on this issue; some examples follow.

- TCP must be implemented in hardware [KANA88].
- TCP must be replaced by a new transport protocol [CHES88].
- Congestion control must be based on rate rather than through a sliding window [CHER86].

- Algorithms that deal with network dynamics (e.g., round trip delay estimation and congestion avoidance) need to be modified [JACO88, RAMA90].
- The operating system, memory, and network adaptor rather than TCP itself limit performance [CLAR89].

Modeling internets presents a paradox: internets today connect tens of thousands of hosts, yet experiments are practical with at most dozens of hosts and gateways. Furthermore, no satisfactory analytic models exist. This leaves simulation as the only practical method. However, the speed and memory size of any single computer -- even a supercomputer -- limits the number of hosts and gateways that may be simulated. This research project addresses the critical question of whether an arbitrarily large internet model could be simulated, given a large enough parallel processor. The question is worth exploring because characteristics that have led to good parallel simulation performance in the past -- loose coupling among model components and *look-ahead* (model components can predict some future actions before receiving enabling events) -- appear to exist in internet models.

The problem addressed by this research, of developing techniques for parallel simulation of thousand to million node internets, is arguably one of the largest computational problems attempted using parallel simulation. To simulate a million node internet may well require a parallel simulation partitioned among dozens of heterogeneous parallel processors communicating over a high speed network. Therefore completion of the proposed project will lead to better understanding of the use of massively parallel computation.

This research will have several other potential impacts. First, completion of the project will give us the ability to study proposed algorithms and protocols in internets with thousands to millions of hosts and gateways. This cannot be done today, and has enormous commercial and research importance. For example, the resultant simulation techniques can be used to give new quantitative data on design alternatives as internets increase in transmission speed and size. Second, we anticipate distributing the simulator outside of Virginia Tech for other researchers to use. Third, our work is based on describing a simulation model as a set of extended finite state machines which are mechanically mapped to a parallel simulation and used to mechanically detect lookahead can be applied to parallel simulation of non-protocol applications. Fourth, the techniques developed for simulating protocols based on extended finite state machines may have application in simulation of neural nets and cellular automata.

1.2 Application Development for Massively Interconnected Systems

The software engineering practice of building massively interconnection applications can be made possible by exploiting the powerful synergism between object-oriented programming (OOP) and Open System Interconnection (OSI). This synergism can be used to simultaneously:

- empower the object-oriented paradigm by infusing it with the distributed programming capabilities inherent in open systems communication, and
- harness the power of an open system by organizing its services within the language framework of object-oriented programming.

The synergy between object-oriented programming and Open System Interconnection arises because in each there are elements which are *complementary* to *corresponding* elements in the other. The corresponding elements will first be identified and then the ways in which they are complementary will be explained.

Figure 1 identifies the corresponding elements which will be discussed at greater length in the following sections. The three elements from object-oriented programming are persistence, concurrency and communication. The three elements from the OSI environment are layered services, structured communication and remote operations. The result of integrating these elements is a computational (application development) paradigm based on persistent, communicating active objects. Careful justification is given that a synergistic effect occurs when the elements depicted in Figure 1 are combined appropriately.

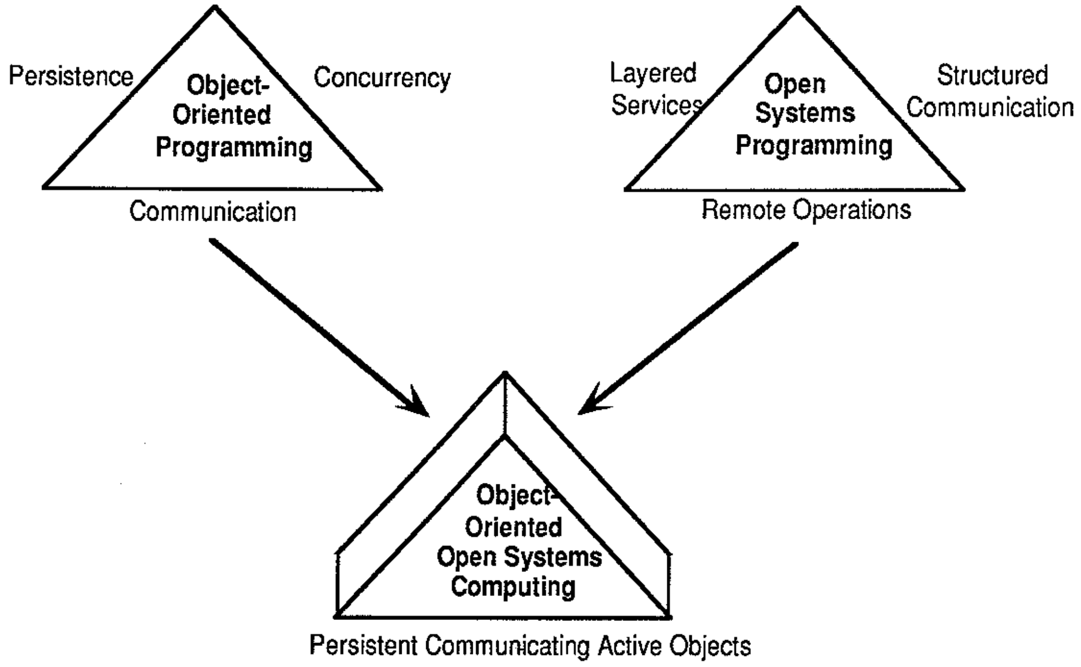

Figure 1. Elements of the Synergy

The three elements of object-oriented programming contributing to the synergy are not accidental; they seem to arise naturally from the requirements of contemporary and envisioned software systems. Wegner [WEGN90] uses the term “megaprogramming” to refer to systems which exhibit distributed concurrency, persistence and heterogeneity. The National Collaboratory is cited as an example of a system imposing these requirements. Similar requirements are given by Zdonik and Maier [ZDON90] for systems developed by “data intensive programming in-the-large”. These systems, exemplified by CAD tools, are large and complex in both function and data. Concurrent access by independent, distributed users to long-lived entities is typical for such applications. Finally, Hewitt describes the needs of an Open Information System (OIS) - a system which is open-ended, incremental and evolutionary [HEWI84]. An OIS is exemplified by an “enterprise-wide information system of the future” [HEWI90]. It is interesting to note that a prototype OIS [DEJO91] relies on concurrent, distributed objects. The requirements expressed in these three views are congruent to the elements we have identified in object-oriented programming: persistence, concurrency and communication.

To justify the claim of synergism between OOP and OSI, the major improvements brought to OSI and object-oriented programming by their integration are now briefly outlined. What does OSI gain? First, the structuring facilities of object-oriented programming can be used to hide the complexity of the OSI protocols from the application developer. By applying an object-oriented approach to OSI the formidable power of the OSI communication services become available through a clean and simple interface. Behind this interface can be hidden all of the forbidding detail which currently deters application development. Second, object-oriented programming can be used to control the complexity of the protocol implementation. We have observed, for instance, that the OSI protocol stack neatly divides into two primary inheritance hierarchies. Greater structuring permits easier experimentation and creates opportunities for performance improvement (e.g., by multithreading). Third, several experimental systems (e.g., [DIXO89, LEDD89, CAMP87]) have demonstrated that inheritance is a useful mechanism for disseminating and specializing the services (in our case OSI communication services) provided by an underlying system. What does object-oriented programming gain? First, objects are easily distributed. The object-oriented paradigm assumes the dimensions of a distributed development paradigm. Second, OSI-based communication provides interaction among objects executing on heterogeneous processor architectures, employing fundamentally different data representations and implemented in different languages. Third, the tight coupling between OSI and object-oriented programming expands the notion of persistence to include persistence of the communication state as well as the state of the object's encapsulated application data. The key point is that these substantial improvements cannot be achieved in isolation. Only by integrating the object-oriented paradigm with OSI can they be realized.

The last issue considered in this introduction is how the OOP/OSI integration relates to the remote procedure call (RPC) mechanism. RPC has two principal aims:

- preserving a familiar programming structure within a distributed computing environment, and
- providing transparent interoperability among heterogeneous architectures.

Both of these goals can be achieved better by the proposed integration than by RPC. The advantage of OOP/OSI over RPC appears in two ways. First, both OOP/OSI and RPC preserve familiar programming structures: objects and procedures, respectively. However, an object is a more robust programming structure - that is, after all, the point of object-oriented programming. Second, communication among objects, often expressed in a message-passing metaphor, is a better basis for distributed communication than that offered by RPC. RPC suggests only one (albeit, a useful one) model of interaction - a client/server model. Object-based communication, while permitting a client/server interaction, also permits interactions which are asynchronous or based on peer-to-peer communication.

2. Internet Simulation

2.1 Why Internet Simulation is Required

Experimentation with small numbers of hosts over a local area network has yielded important insights about protocols and their implementations [JACO88, BORM89]. Recently, construction of five testbeds for experimentation with gigabit rate networks have been initiated [IEEE90]. Experimentation is limited by practical consideration to relatively small numbers of hosts and to network architectures that can be built from components currently available. Therefore experimentation cannot answer all questions arising in internet design.

Promising analytic models are being developed [BOLO90, SING90], but these techniques are limited in the number of network connections that can be analyzed.

This leaves simulation. Simulation is attractive for several reasons. A simulation can model internets with arbitrary speed, size, topology, protocols, and transmission media. It may be possible to simulate internets on parallel processors or even networks of parallel processors to allow arbitrarily large simulation models. Finally, simulation has been successfully applied to *individual* networks of 1 to 10^3 nodes. Examples include MIT's Network Simulator [HEYB89], COMNET II [CACI90], and the University of Maryland's Routing Testbed [ALAE90].

2.2 Why Internet Simulation Is Hard

Can we simulate internets? In particular, can individual network simulators be modified in a straightforward manner to simulate thousand to million node internets? Currently, the answer to the first question is "not yet" and to the second question is "no," for three reasons. Existing individual network simulators are:

1. sequential programs, which are difficult to port to parallel processors;
2. large, complex programs that are difficult to understand, validate, and modify; and
3. designed for the 1 to 10^3 node range, which implies they contain unnecessary detail for modeling the 10^3 to 10^6 node range. For example, faithfully representing the media access timings by a single LAN host may be unnecessary when simulating a million nodes. However, limited buffer storage in hosts may be critical to an internet model. The challenge addressed by the proposed research is to select judiciously model features that yield output measures correct to an order of magnitude, but which do not achieve unnecessary accuracy at the expense of code complexity and long simulation running times.

2.3 Objectives and Tools

Our objectives are to:

1. develop new techniques for simulation of 10^3 to 10^6 node internets that:
   - A. require less programmer time to construct a simulation program representing a protocol, and
   - B. require less wall clock time to execute than sequential simulation requires;
2. demonstrate the validity of the techniques in objective 1 by building an internet simulation and validating its output measures using real network measurements (We also anticipate distributing the simulation for others to use.); and
3. formulate recommendations for end-to-end internet protocols algorithms and parameter settings using the simulation of objective 2. In particular, the aim is to explore parameter settings and congestion control algorithms (e.g., contrast window- and rate-

based mechanisms). One can also view this objective as investigating the physics of internets using simulation.

Our emphasis is on developing *techniques* for simulation that can be applied to any protocol and internet, rather than developing a simulation of a particular protocol running on a particular internet topology. However, to insure that the techniques developed capture a sufficient level of detail to accurately predict internet dynamics, we will use the techniques to build a simulator and validate that simulator with measured data. Finally, we will use this simulator to draw some preliminary conclusions about thousand to million node internets. We intend these conclusions to be the point of departure for my research work following completion of the project proposed here.

Accomplishment of the objectives will be done using three prior research tools that we have built.

1. *A general purpose parallel simulation system, called OLPS [ABRA88].* OLPS implements three general purpose parallel simulation algorithms (time warp [JEFF85], bounded lag [LUBA89], and Chandy-Misra's algorithm with deadlock avoidance through null messages [CHAN79]) as well as an efficient sequential simulator. The sequential simulator may be used to estimate speedup of each parallel simulation algorithm.

   A unique feature of OLPS is that all four simulators share a common programming interface [ABRA87]. Therefore, given a single simulation program written for this interface, OLPS mechanically generates four simulators. This feature is important in the proposed research, because no consensus exists as to which parallel simulation algorithm is superior. To satisfy objective 1, the performance of internet simulations based on the four simulation algorithms listed above will be evaluated using a variety of workloads. In particular, both trace driven and synthetic workloads will be constructed from two sources: traffic measurements that have been made on department networks and the backbone network within Virginia Tech, and traffic statistics that can be obtained from NSFnet. Workloads will also be constructed based on the client-server paradigm representing remote logins and file transfers.

2. *A hardware-based measurement tool for distributed systems software [ABRA87].* The tool consists of a measurement card that is inserted into each host on which measurements are performed. The hardware has been fabricated for the IBM PC-AT bus, and when funding is available it will be ported to the bus of a RISC workstation that can be equipped with a high speed network adaptor (e.g., FDDI). The tool is currently being used to examine the transient behavior of IBM's TCP/IP protocol on Ethernet and token ring connected IBM PS/2's. The measurements will be used in the validation required by objective 2.

   The measurement hardware is unique in that it provides to all hosts equipped with the card a time base synchronized to within one microsecond. The tool permits *event driven* and *time driven* measurement. Event driven measurement requires inserting calls to trace events into the source code, and generates a single log of the order and times of these user-defined events on all network hosts equipped with the measurement hardware. Time driven measurement interrupts all hosts equipped with the measurement hardware within a window of 1 microsecond at a regular interval which can be set from 4 microseconds to 1.19 hours. Time driven measurement requires writing an interrupt handler that examines and writes to a disk log important data structures of the software under study (e.g., to collect queue lengths). We are currently developing tools to visualize the performance data collected using the X Window System.

3. *A prototype TCP/IP simulation model.* The model is intended to be suitable for parallel simulation of internets containing a thousand hosts and gateways. The prototype model represents data transfer, but omits connection establishment and release. It uses a client-server workload and incorporates the algorithms proposed by Jacobson (e.g., slow start, exponential retransmission timer backoff, additive increase/multiplicative decrease window sizing, and a fast round trip delay mean and variance estimation algorithm) [JACO88]. The model is specified as a set of finite state machines, as is discussed in section 2.4.

Finally, the project will complement and utilize another tool present at Virginia Tech, which is the Model Development Environment [BALC87]. The Model Development Environment will be used to map a protocol specification mechanically to a simulator.

The combination of the OLPS parallel simulation system, the hardware-based measurement tool, the prototype TCP/IP simulation model, and the Model Development Environment provides a unique set of tools that will permit us to study the problem of simulating large internets in a way that few other researchers can.

2.4 Extended Finite State Machines

A novel aspect of this research is to map a protocol described by a set of extended finite state machines (EFSM's)[1] to a parallel simulator in a mechanical fashion. An EFSM consists of states,arcs and variables. Each arc is associated with a predicate and an action. Predicates are used if and only if the arc is one of several out of a state. The actions are code fragments that may include sending and or receiving a message and also modifying the values of variables.

The methodology of mapping an EFSM protocol description to a parallel simulator will fulfill objective 1A of overcoming the traditional problem of a protocol simulator being a large, complex program for several reasons. First, protocol designers are used to working with EFSM's. Therefore a protocol designer will have more confidence that the simulator reasonably represents the protocol if the simulator is directly synthesized from the EFSM's. Second, if the simulator needs modification, then the EFSM can be modified and remapped to a simulator.

Synthesizing a simulation model from EFSM's will also have several advantages with respect to parallel simulation. Existing parallel simulation algorithms use a computation model in which a set of *logical processes* communicate via messages. (A logical process is a scheduleable unit of code.) Each EFSM naturally maps to one logical process. A second advantage of using EFSM's is to exploit *lookahead.* Lookahead is the ability of a logical process of a parallel simulation to predict actions that occur after reception of future messages *before* the logical process receives those messages. The existence of lookahead leads to higher speedup in parallel simulations [FUJI88]. A current problem in the parallel simulation field is that there is no automatic technique to determine if lookahead exists for an arbitrary simulation model. In addition, at the moment the parallel simulation code must be hand-coded to exploit lookahead.

---

[1] finite state machines are also called state transition diagrams and finite state automata.

The lookahead problem will be alleviated by using EFSM's, which naturally reveal lookahead, as the following examples illustrate. This solution to the lookahead problem apparently has not been recognized in the literature.

1. Suppose that a protocol entity is making a transition from state S to S', and that the action associated with this transition requires waiting for a message. Further assume that state S' has one outgoing transition to state S''. Therefore, before the awaited message arrives, we can predict that the entity will enter states S' followed by S''. This type of lookahead is mechanical to deduce from EFSM's.

2. If state S'' has multiple outgoing transitions, and the predicates that select the transition depend on the data to be received in the message, then lookahead may still be possible. One would examine the sender's EFSM. From its current state, it may be possible to predict what the message to be sent will contain. This again can be automated to reveal lookahead.

3. The protocol implementation technique of predicting header contents to speed up protocols can also be employed in a simulator as another form of lookahead.

A third advantage of EFSM's is that they can be efficiently mapped to various parallel architectures. One implication is that the EFSM's may be mapped to SIMD as well as MIMD architectures. A second implication is that it will facilitate running a protocol simulator on multiple, heterogeneous parallel computers that are interconnected by a local area network. This requires a simulator to be portable to different parallel computers and still yield acceptable execution efficiency. We plan to try multiple multiprocessor internet simulations at Virginia Tech on our Sequent and Hypercube.

A fourth advantage of EFSM's is that they provide a basis for developing special purpose parallel algorithms for protocol simulation. First, EFSM's may provide a means to efficiently execute the simulation on an SIMD machine. If the number of states is much less than the number of SIMD processors, then the simulation can consist of an execution schedule that cycles through all states. For each state, the actions associated with the state are broadcast to all processors. Second, some messages that are used in general purpose parallel simulation algorithms may be omitted in a special purpose simulator, if one EFSM can predict another EFSM's transitions. A similar idea has been used in parallel simulation of chches by computer architects. Third, it may be possible to simulate two protocols on the same workload simultaneously in an amount of time that is less than the sum of the time required to simulate each protocol separately. This idea has also been used in cache simulation, where caches with different parameters are simulated simultaneously. Finally, trace reduction techniques may exist for internet traffic traces, just as reduction techniques exist for memory reference traces for cache simulations.

3. An Application Environment for Massively Interconnected Systems

Designers of distributed applications must deal with two fundamental problems: invoking the operations provided by remotely located objects (termed remote operations) and transmitting complex application data structures (termed structured communications) among heterogeneous systems. These two problems - daunting enough in small-scale distributed systems - become insurmountable in massively interconnected system unless suitable mechanisms are provided in the execution environment. The combination of object-oriented programming and the OSI protocols can be used in the execution environment to reduce these problems to manageable proportions.

3.1 Remote Operations

An open systems application is a computational process in which a relatively significant portion of the computation is concerned with establishing and managing associations with other application processes residing on different nodes in a network. Internally, an application process is a collection of *application entities*, each representing some communication aspect of the application process. Figure 2 illustrates two application processes, each consisting of two application entities. Remote communication is realized for each application entity through use of one or more *application service elements* (ASEs). ASEs provide common abstractions needed by most application entities and a higher-level interface to the services offered by the Presentation Layer. Two common ASEs are the Association Control Service Element (ACSE) and the Remote Operations Service Element (ROSE) [ISO88]. The ACSE provides a service for binding and unbinding an association with a remote entity. The entity requesting an association is called the *initiator* while the entity accepting the association is called the *responder*. The ROSE provides the basic services required to implement remote interactions.

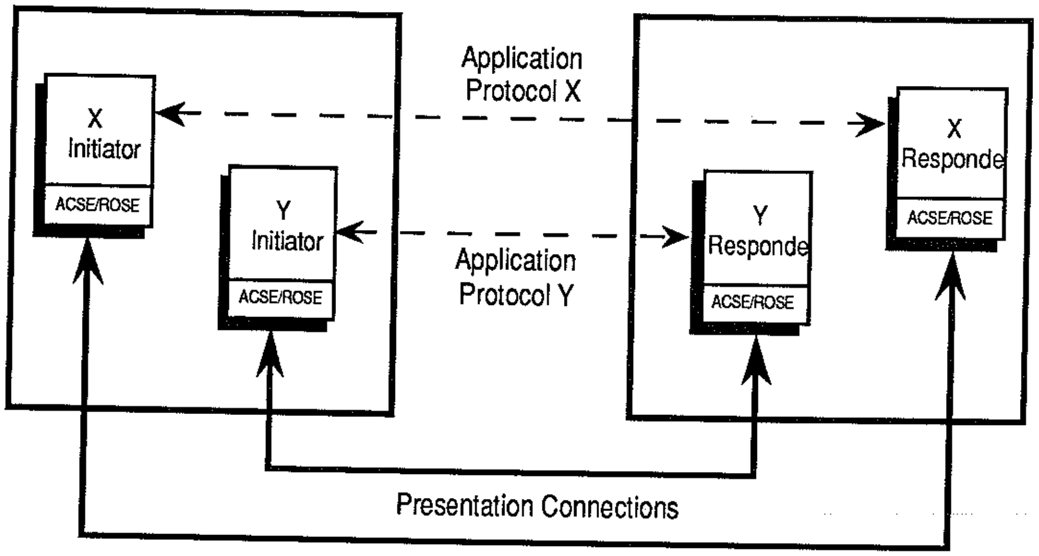

Figure 2. Communicating Peer Application Entities

From an object-oriented perspective, application entities are objects which communicate with remote objects. From this perspective, the boundary surrounding an application entity object can be exploited to encapsulate:

- complexity: the existing interfaces to application services entail lengthy argument lists containing system structures which the user must retain and supply with later uses of the service. These system structures can be better represented as objects.
- distribution: an application process may be composed of both locally communicating objects and remotely communicating objects. The application developer need make no distinction between remote and local objects.

- protocol: each pair of peer application entities uses a separate, but not necessarily unique, protocol. The protocol used in interacting with a remote object may be completely hidden from the application developer.

- language: an application entity need only be concerned with the external behavior of a peer entity whose services it uses. The internal implementation details - including the implementation language - is encapsulated, allowing entities implemented in different languages to interact.

Structuring the ASEs as objects creates an application environment with simpler, more abstract services and one which is safer as the control of arguments and the proper use of defaults can be insured by the ASE class designer.

3.2 Structured Communications

An important aspect of distributed programming is to impose the structure of application data onto the otherwise unstructured (bit stream) offered by the underlying communication service. Peer application entities, like those shown in Figure 2, use this structured communication to request operations and transmit application data values as arguments and results. The interacting entities may reside on machines whose hardware architectures differ in their representation of common values, such as integers. Such heterogeneity necessitates techniques enabling the consistent invocation of remote operations and the correct interpretation of data values.

A common approach to structured communication is to provide a remote procedure call (RPC) protocol [BIRR84] and a machine independent data representation language. Various RPC protocols and data representation languages exists; for example, Sun Microsystems provides a popular datagram based RPC [SUN88] used in conjunction with the External Data Representation (XDR) [SUN87] language for specifying C language data types.

Abstract Syntax Notation One (ASN.1) [ISO87] is defined as an ISO standard data representation language which is representationally more powerful than XDR. ASN.1 is used in conjunction with the Remote Operations Service Element to provide a remote operations service for OSI-based applications.

There are two points of synergism between the object-oriented paradigm and structured communications. First, ASN.1 and the ROSE offer sufficient mechanisms for supporting remote communication among cooperating objects. In particular, ASN.1 has the representational power to express the strongly-typed method signatures found in a typical class definition. Second, the encapsulation properties of objects allow the translation mechanics implied by XDR or ASN.1 to be concealed from the user of the object.

3.3 Project Synergy

Recently, the Synergy Project was initiated to develop a prototype system synthesizing object-oriented programming and the OSI environment. By judiciously adapting existing software, a workable prototype will be operational within approximately one to two years. The global architecture of the Synergy system is shown in Figure 3. The interaction among the parts of this architecture will be explained by following a scenario of how classes are defined by a "provider" and accessed by a "user."

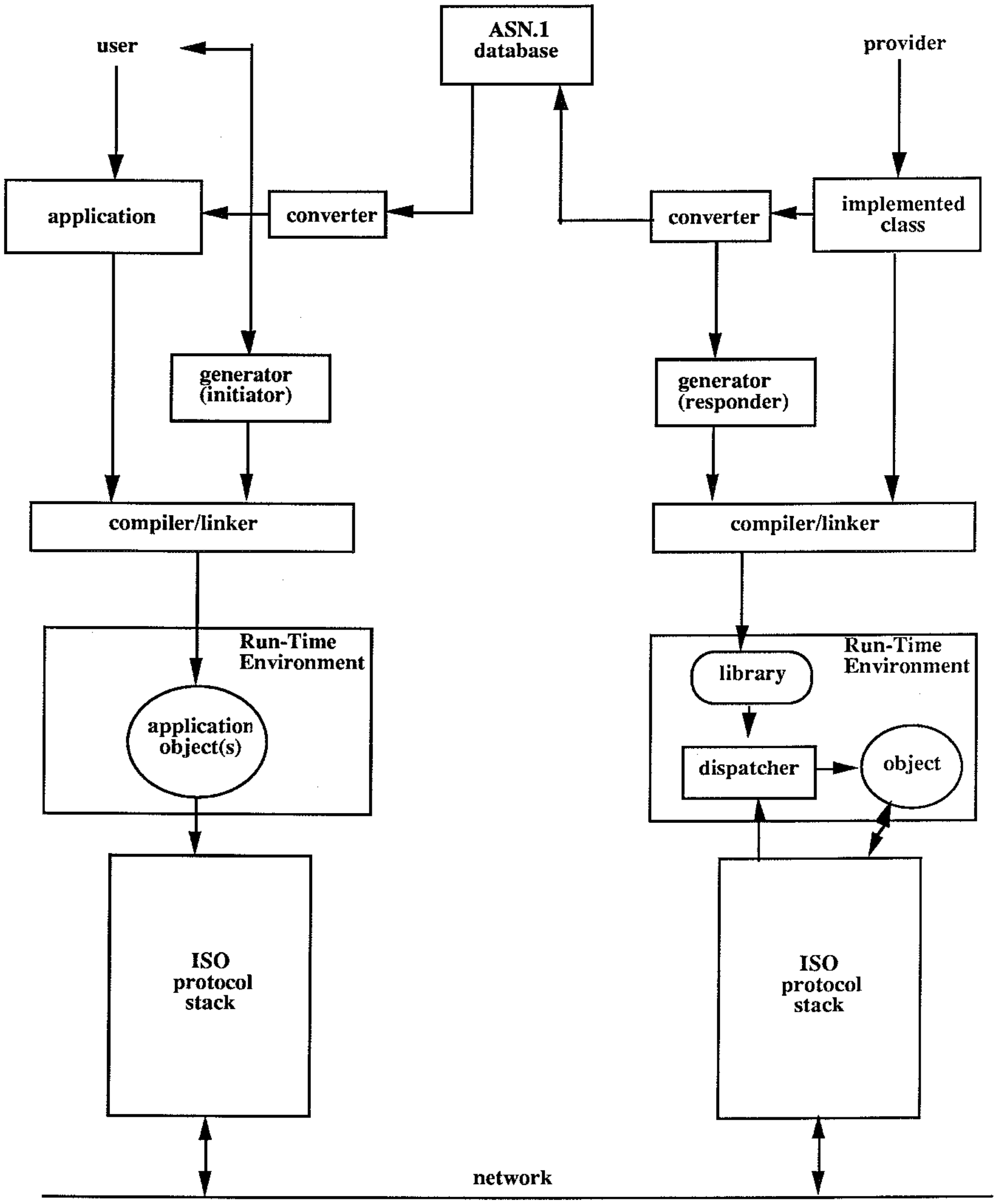

Figure 3. Architecture of Project Synergy

The developer of a distributed application, the "user", will have available a universe of existing classes (types) which have been previously implemented and made available for reuse by "providers." These classes will be defined in a programming language-independent fashion. The class definition notation we use is Abstract Syntax Notation One (ASN.1) although any other similar type definition language with equivalent expressive power could be used. We chose ASN.1 because it is part of the OSI standards and because there exist public domain tools for processing ASN.1 specifications [ROSE90].

Retrieving classes for reuse from the universe of predefined classes is an interesting question, but one which is beyond the current scope of the Synergy Project. One could foresee, however, employing the Synergy environment to build a distributed system aiding in the identification and retrieval of needed classes. The retrieval system itself illustrates the kind of application we imagine Synergy would support - systems requiring access to persistent objects (the long-lasting ASN.1 descriptions of available classes) in a distributed and heterogeneous environment (classes may be offered by other systems/organizations without prior agreement) and which may be implemented in various languages (the classes offered by a system/organization will be implemented in the developing organization's language of choice which may differ from the user's language of choice).

The provider of a class fully implements the class in the provider's language of choice. A converter examines this implementation and produces the class specification expressed in ASN.1. This ASN.1 specification is made available (ideally through a distributed data-base) to potential users. The ASN.1 specification is input to a responder generator. This generator produces "boilerplate" methods needed to interface with the OSI-based run-time environment on the provider's side. The responder methods and the methods developed by the provider are woven together by the compiler/linker to produce an entry in a library. Once installed in the library the class is available for use. This is all that the supplier need be aware of. We will consider in a moment how the instantiation of an object of this class and the invocation of its methods will take place.

A user examines the available class specifications and selects those which are appropriate for use. While the user may examine the specification in the ASN.1 syntax, in Figure 3 we show that a converter is used to generate a class definition in the user's language of choice. Selected class definitions are added to the application code under development and are also input to an initiator generator. This generator produces a "boilerplate" class whose methods interface to the ISO-based run-time environment on the user's side. The full application and the output from the initiator generator are woven together by the compiler/linker.

We now consider the events which occur when the application instantiates an object of a class provided by another node in the network. The application instantiates an object in the "boilerplate" class produced by the initiator generator. The constructor of this class uses the OSI application services to establish an association with a dispatcher on the host supplying the implementation of the desired class. Using this association, the user's object requests the dispatcher to instantiate an object of the class, execute the object's constructor and bind the association to that object. The constructor of the user's object completes when the remote object has been completely created. Thereafter, invoking a method of the user's object results transparently in the invocation of the remote object's methods using the OSI remote operations facilities. Destructing the user's object results in a termination protocol being followed to destruct the remote operation and close the association.

4. Current Status

4.1 Internet Simulation

To model internets with thousands, millions, or billions of nodes will require execution of a simulation on a parallel computer. Therefore we are studying implementation of the model on a variety of architectures using a variety of parallelization techniques. We are investigating two classes of architectures, MIMD and SIMD. We are investigating three parallelization techniques: optimistic and conservative discrete event simulation [FUJI90] as well as a numerical solution method.

We are currently producing three Internet simulation models:

- Implement the model in Sim++, a simulation programming language that runs on Time Warp [JEFF85] on a network of transputers.
- Implement the model using Nicol's conservative synchronous parallel simulation algorithm [NICO90] on hypercubes. (This is work with Dr. David Nicol at the College of William and Mary.)
- Model a window flow control mechanism running on a network that looses datagrams by a set of recurrence relations that are evaluated in parallel [GREE90] on a Connection Machine.

4.2 Application Environment - Project Synergy

The cornerstone of Project Synergy is OSI/C++ - a re-engineering of the OSI protocols in C++. Mr. Greg Lavender, a Ph.D. student at Virginia Tech, is currently re-engineering and implementing the OSI/C++ protocol stack. This work is being done by Mr. Lavender as an intern at the Microelectronics and Computer Technology Corporation (MCC) in Austin, Texas. Mr. Rajesh Khera, an M.S. student at Virginia Tech, is currently examining the ASN.1 tools in the ISO Development Environment (ISODE) in preparation for extending these tools to support object-oriented structures.

References

[ABRA87] M. Abrams (1987), Synchronous 1MHz Clock/Timer for Measurement of Network-Connected IBM PC's, Research Report RZ 1640, IBM Zurich Research Laboratory, Oct.

[ABRA88] M. Abrams (1988), "The Object Library for Parallel Simulation (OLPS)," *Proc. Winter Simulation Conference,* San Diego, CA, Dec., 210-219.

[ALAE90] C. Alaettinglu, K. Dussa, A. U. Shankar, and J. Bolot (1990), *Routing Testbed: Initial Design,* CS-TR-2475, Dept. of Computer Science, Univ. of Maryland, May.

[BALC87] O. Balci and R. E. Nance (1987), "Simulation Model Development Environments: A Research Prototype," *Journal of the Operational Research Society 38*, (8), 753-763.

[BIRR84] Andrew D. Birrell and Bruce J. Nelson. "Implementing remote procedure calls, " *ACM Transactions on Computer Systems*, 2(1), February 1984, pp.39-59.

[BOLO90] J. Bolot and A. U. Shankar (1990), *Analysis of a Fluid Approximation to Flow Control Dynamics*, CS-TR-2553, Dept. of Computer Science, Univ. of Maryland, Oct..

[BORM89] D. Borman (1989), "Implementing TCP/IP on a Cray Computer," *Computer Communication Review 19*, (2), pp. 11-15.

[CACI90] CACI Products Company (1990), *COMNET II.5 Overview*, March.

[CAMP87] Roy Campbell, Gary Johnson and Vincent Russo, "Choices (Class Hierarchical Open Interface for Custom Embedded Systems," *Operating Systems Review*, Vol. 21, Number 3, July, 1987, pp.9-17.

[CHAN79] K. M. Chandy, C. Holmes, and J. Misra (1979), "Distributed Simulation of Networks," *Computer Networks 3*, 105-113.

[CHES88] G. Chesson, B. Eich, V. Schryver, A. Cherenson, and A. Whaley (1988), *XTP Protocol Definition*, Technical report revision 3.0, Silicon Graphics, Inc., Jan.

[CHER86] D. Cheriton (1986), "VMTP: A Transport Protocol for the Next Generation of Communication Systems," In *Proc. ACM SIGCOMM* (Stowe, Vermont, Aug), 406-415.

[CLAR89] D. Clark, V. Jacobson, J. Romkey, and H. Salwen (1989), "An Analysis of TCP Processing Overhead," *IEEE Communications Magazine 27*, (6), 23-29.

[DEJO91] Peter deJong, "A Framework for the Development of Distributed Organizations," unpublished paper, 1991.

[DIXO89] G. N. Dixon, G D. Parrington, S K. Shrivastava, and S. M. Wheater. "The treatment of persistent objects in arjuna, " *Procedings: ECOOP'89 Proceedings of the 1989 European Conference on Object-Oriented Programming*, July 1989, pp. 169-189.

[FUJI88] R. M. Fujimoto (1988), "Lookahead in Parallel Discrete Event Simulation," *Proc. Int. Conf. on Parallel Processing*, St. Charles, IL, Aug.

[FUJI90] R. M. Fujimoto (1990), "Parallel Discrete Event Simulation," *CACM 33* (10), Oct., 30-53.

[GREE90] A. G. Greenberg, B. D. Lubachevsky, and I. Mitrani (1990), "Unboundedly Parallel Simulations Via Recurrence Relations," *Proc. ACM SIGMETRICS*, Boulder, CO, May, 1-12.

[HEWI84] Carl Hewitt and Peter de Jong. "Open systems, " in On Conceptual Modeling, (ed. Michael L. Brodie), Springer-Verlag, 1984, pp. 147-164.

[HEWI90] Carl Hewitt. "Towards open information systems semantics, " unpublished paper, 1990.

[HEYB89] A. Heybey (1989), *The Network Simulator,* Laboratory for Computer Science, MIT, Oct.

[IEEE90] IEEE Computer Society (1990), "Gigabit Network Testbeds," *Computer 23,* (9), 77-80.

[ISI81] Information Sciences Institute (1981). *Transmission Control Protocol,* NIC-RFC 793, Sept.

[ISO87] International Standards Organization. Information Processing --- Open Systems Interconnection --- Specification of Abstract Syntax Notation One (ASN.1), International Standard 8824, 1987.

[ISO88] International Standards Organization. Information Processing --- Text Communication --- Remote Operations part 1: Model, Notation and Service Definition, Working Document for International Standard 9072--1, 1988.

[JACO88] V. Jacobson (1988), "Congestion Avoidance and Control," In *Proc. ACM SIGCOMM,* Aug, 314-329.

[JEFF85] D. Jefferson (1985), "Virtual Time," *ACM Transactions on Programming Languages and Systems 7,* (3), 440-425.

[KANA88] H. Kanakia and D. Cheriton (1988), "The VMP Network Adaptor Board (NADB): High-Performance Network Communication for Multiprocessors," *Proc. ACM SIGCOMM,* Aug, 175-187.

[LEDD89] Bill Leddy and Kim Smith. "The Design of the Experimental Systems Kernel, " *Proceedings of the Conference on Hypercube and Concurrent Computer Applications,* Monterey, CA, 1989.

[LUBA89] B. D. Lubachevsky (1989), "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," *Comm. ACM 32,* (1), 111-131.

[NICO90] D. M. Nicol (1990), *The Cost of Conservative Synchronization in Parallel Discrete Event Simulations,* Report 90-20, Inst. for Comp. App. in Sci. and Eng., NASA Langley Research Center, May.

[RAMA90] K. K. Ramakrishnan and R. Jain (1990), "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks," *ACM Trans. on Computer Systems 8,* (2), 158-181.

[ROSE90] Marshall T. Rose. The Open Book: A Practical Perspective on OSI, Prentice-Hall, 1990.

[SING90] S. Singh, A. K. Agrawala, and S. Keshav (1990), *Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits,* CS-TR-2490, Dept. of Computer Science, Univ. of Maryland, June.

[SUN87] Sun Microsystems. "XDR: external data representation standard, " *Request for Comments 1014*, SRI Network Information Center, June 1987.

[SUN88] Sun Microsystems. "RPC: remote procedure call protocol specification version 2, " *Request for Comments 1057*, SRI Network Information Center, June 1988.

[WEGN90] Peter Wegner. "Concepts and paradigms of object-oriented programming, " *OOPS Messenger*, 1(1), August 1990, pp. 7-87.

[ZDON90] Stanley B. Zdonik and David Maier, "Fundamentals of Object-Oriented Databases" in *Readings in Object-Oriented Database Systems*, (eds. S.B. Zdonik and D. Maier), IEEE Press, 1990, pp.1-32.