Contents

1 EXECUTIVE SUMMARY

Applications that require the resources of a supercomputer are quickly reaching the point where it will be difficult to find a single computing platform on which they may be executed. It has been shown that the computing and storage requirements of large scale C3I applications and other High Performance Computing and Communications (HPCC) applications will be beyond the capacity of any single supercomputer. A network of supercomputers and high-performance workstations appears to be the only reasonable way to provide adequate computing resources for the large scale C3I and HPCC applications of the next century. Such a collection of computers and supporting software environments is called the Virtual Distributed Computing Environment (VDCE). The VDCE gives the perception of using a single, integrated computing system where users can access local or remote resources, and run processes from anywhere in the system, without being aware of the computers on which their processes are running.

The current implementation of C3I and HPCC applications using existing parallel and distributed computing software tools and environments have two main limitations. These tools require the users to define a priori the type and address of each computer that will run each task of an HPCC application. Any dynamic change in the machines involved in the application execution will incur high overhead so that it becomes in-efficient to change the execution environment at runtime. However, with large number of computers (say 100 or even 1000) involved in the computations the probability of having failures in at least one computer or in one communication link is high. Furthermore, since these applications require long execution times, in minutes or hours, it is very likely that system loads will change and thus require dynamic load balancing to improve performance and to accommodate the needs of local jobs.

We do believe that the existing parallel and distributed software tools and environments can not handle the dynamic load balancing and fault tolerance issues efficiently. Furthermore, they were not designed to handle large scale C3I and HPCC applications involving many computers interconnected by high speed local and wide area networks. There is also a need to provide each large scale application a virtual distributed computing environment (VDCE) because its computing resources can change dynamically at runtime as well as their locations. Consequently, every large scale application can have one VDCE to meet its computing and storage requirements and several VDCEs can run simultaneously.

The main objective of this project is to evaluate the current advances in computing and networking technologies and parallel and distributed computing software tools and identify those that can be used to implement the proposed virtual distributed computing environment.

We will also demonstrate and validate this approach by implementing two VDCE applications on the NYNET testbed; the applications will be chosen after consultation with Rome Laboratory personnel. Our approach to identify VDCE enabling technologies and to develop a methodology to build VDCEs and their applications can be outlined as follows:

- Evaluate potential VDCE enabling technologies that will play an important role in the development of VDCEs. The technologies to be studied and evaluated include: Parallel and Distributed Programming Software; Software Integration Tools and Environments; Web-based Software Tools, and High Speed Networks and Protocols.
- Develop a conceptual model to implement the virtual distributed computing environment. The model should answer two important questions: 1) How users can develop Efficient C3I and HPCC applications and what software tools can best be used to develop VDCE applications?; and 2) How VDCE applications are executed and how can we achieve load balancing and fault tolerance across VDCEs that might span local or wide area networks?
- Demonstrate and validate the main functions of the VDCE modules on two C3I applications that are developed based on message passing and distributed shared memory paradigms.

Our approach to investigate potential VDCE enabling technologies and implement VDCEs is based on dividing VDCE into two distinct major modules: the Software Development Module (SDM), and the Execution Module (EXM); each module will attempt to answer the questions related to how to develop and run VDCE applications. The Software Development Module (SDM) provides the tools required to produce a high-level description of a VDCE application. With SDM tools, the user can easily develop, test and evaluate the performance of a large scale VDCE application. We will investigate the required SDM functions by identifying the services and functions of three layers that are essential to develop any application: specification layer, design layer, and coding layer. Each layer is responsible for attaching specific information to the application description (e.g., application graph). The second module is the Execution Module's (EXM) that is responsible for executing and managing the execution of VDCE applications. The EXM module can be developed using the services offered by two managers: Compilation Manager and Runtime Manager. Compilation Manager is responsible for preparing the executable code for each component or task of a VDCE application. Runtime Manager is responsible for managing the execution of a VDCE application. The basic service of the runtime manager is selecting the best machines on which to run the various tasks, loading the corresponding binaries, initiating execution, and achieving efficient load balancing and and fault tolerance.

We believe that we are at a very exciting and challenging time with computing, communications, and information technologies converging as we enter the digital inforamtion age. In this proposal we bring expertise from all these diverse areas which we expect to be essential in developing the next generation VDCE. The researchers involved in this project have extensive experiences in using current parallel and distributed computing tools (MPI, PVM, ISIS, P4, EXPRESS, and HPF) to develop HPCC applications on a wide area network (NYNET). NYNET is an ATM wide area network that covers all New York State and part of Massachusetts State. We have demonstrated successfully several large scale applications (option stock pricing application, telemedicine, C3I distributed computing (Concurrent Multitarget Tracking), electromagnetic scattering simulation, multimedia distributed information systems, and video on demand applications). Each one of these NYNET applications runs on a collection of computers that range from parallel machines (Intel 860, CM 5, MasPar, NCUBE) to a heterogeneous network of computers (ALPHA DEC workstations, IBMs RS 6000, SUNs, etc.). Furthermore, we have developed and experimented with several important VDCE fucntions and services associated with SDM and EXM modules. The results of these experimental efforts have been published in several technical papers that are enclosed in this proposal Appendix.

2 AN OVERVIEW OF THE PROPOSAL

The organization of the proposal is as follows. In Section 3, we overview the main technologies that will play an important role in the development of the virtual distributed computing environment. In section 3.1, we address the trends in computer architectures and show how VDCEs will be a natural evolution of high performance computer architectures.

In Section 3.2, we review the issues related to parallel and distributed computing software. This subsection is very important because it identifies the main software issues and tools that will be viable to build the software development capability of the proposed virtual distributed computing environment. In this subsection, we identify four important approaches to develop large scale software systems: component-based software systems, parallel and distributed programming tools, network-based visualization systems, and web-based tools. These techniques need to be investigated in order to identify the software tools and environments needed to develop VDCE and its applications. In Section 3.3, we briefly describe the trends in computer network technology. Computer network is an important system and critical to the performance of VDCEs. Consequently, this project will investigate and identify the viable networking technologies and communication protocols that will be used to build the VDCE communication infrastructure.

In Section 3.4, we briefly examine the characterstics of potential VDCE applications in terms of the following attributes: computation, storage, visualization, heterogeneity, parallelism, communication, and synchronization. Understanding the characteristics and requirements of VDCE applications is an important step that will enable us to identify the required software technologies to develop VDCE applications as well as to provide an efficient, fault tolerance execution environment. In Section 4, we discuss a high-level conceptual model to develop the proposed VDCE. Based on the overview of the VDCE relevant technologies presented in Section 3, one can develop such a model in terms of two modules: Software Development Module and Execution Module. In that section, we discuss in more detail the layers and the issues that must be investigated to implement each of these two VDCE modules.

In Section 5, we give a brief description of the on-going VDCE related research projects at NPAC at Syracuse University. In that section, we discuss three different approaches currently being evaluated at Syracuse University to develop VDCE important functions and/or services: Procedural-based, Object-based, and Web-based approaches. In Section 6, we present the project tasks, milestones, and demonstrattions.

The Table 1 provides a cross-reference between the issues raised in the RFP document (PR NO. C-5-223) and the sections in this proposal where these issues have been addressed and some solutions have been outlined. For example, the task/technical requirement 4.1.1.2 that requires to develop a conceptual model of the VDCE is discussed in Sections 3.1, and 3.2.

Table 1: Cross-reference Table

Statement of work	Related Sections	Tasks	Demonstration
4.1.1.1	2.2, 2.3, 2.4, 3.1, 3.2	1, 2.1, 2.2	1
4.1.1.2	3.1. 3.2	2.1, 2.2, 2.3	1
4.1.1.3	2.2, 2.3, 2.4.8	2.1, 2.2, 2.3	1
4.1.1.4	2.2, 2.3, 2.4.8	2.3	1
4.1.2.1	2.4, 4.1.2	2.3	1
4.1.2.2	4.1.2	2.3	1
4.1.2.3	3.1, 3.2	3, 4	2, 3
4.1.3.1	3, 4	4	3

3 OVERVIEW OF VDCE RELEVANT TECHNOLOGIES

Advances in computer networking technology have introduced high speed, reliable networks capable of providing high transfer rates. Advances in software technology have provided easy-to-use tools and environments for the development of parallel and distributed applications. These advances have resulted in the proliferation of a large number of different architectural classes like SIMD computers, MIMD computers, vector computers, and data-flow computers, where each class represents a set of different trade-offs in design decisions like coarse-grain parallelism vs. fine-grain parallelism, shared-memory architecture vs. distributed-memory architecture, and circuit switched vs. packet switched. Each architectural class is tuned to deliver a maximum performance to a specific set of applications. However it remains a fact that none of the existing computing systems are general enough to address all classes of applications and provide the desired performance levels. In addition, these architectures are not scalable and their relatively narrow applicability has prevented them from being cost-effective. Consequently, there is no single existing architecture that meets all application requirements. It is this realization that has spurred intense research in virtual distributed computing environments.

We believe that the future of high performance computing lies in the integration of the plethora of specialized architectures into a single virtual distributed computing environment that allows them to cooperate in solving large scale complex applications. The VDCE to be developed in this project will capitalize on existing computer architectures and on current advances in networking technology and software to provide efficient, cost-effective, scalable, high-performance distributed computing. In what follows, we briefly outline the main issues and advances in computing and networking technologies, parallel and distributed software tools and large scale application requirements that will play an important role in the development and deployment of virtual distributed computing environments.

3.1 High Performance Computers

The advances in high performance computer technology have resulted in the proliferation of a large number of different architectural classes like SIMD computers, Shared Memory (SM) and Distributed Memory (DM) MIMD computers, etc. where each class represents a set of different trade-offs in design decisions. Figure 1 shows when these different architectures have been introduced and how they are evolving. This figure also reflects the fact that future generation of high performance computing systems will be built by utilizing the current high performance comptuers and integrate them into one virtual distributed computing environment. The main focus of the proposed research will be in this area on analyzing and assessing the current and future computing technologies and on developing a scheme to optimally integrate them into a single VDCE.

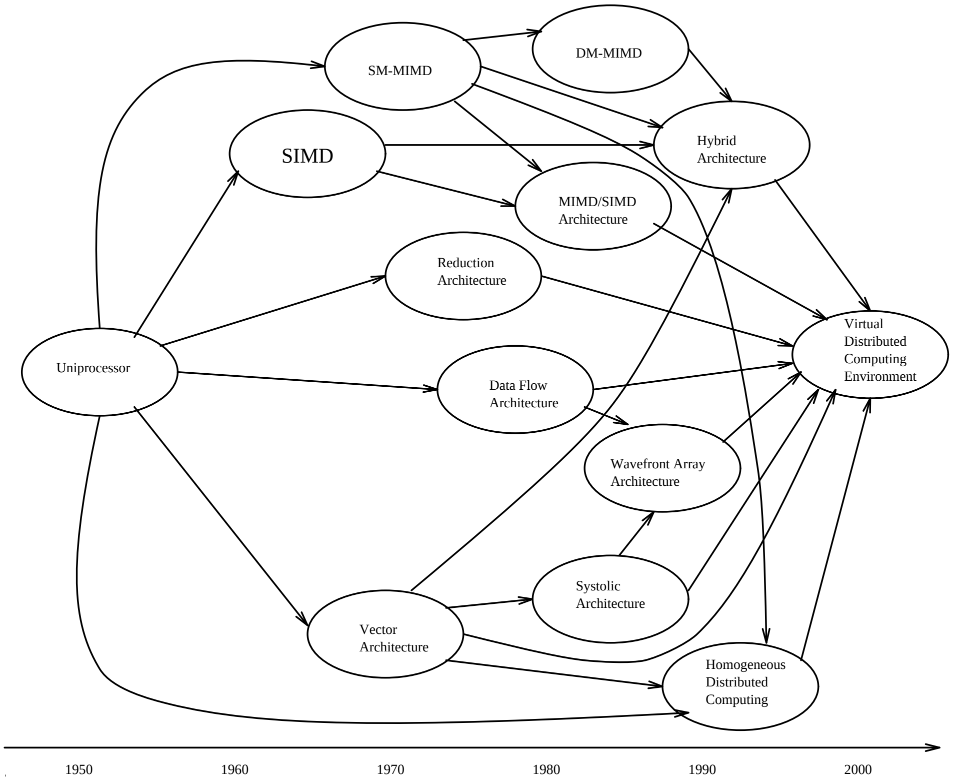


Figure 1: Architectural trends towards VDCEs

3.2 Parallel and Distributed Computing Software

In this subsection, we overview the software tools and environments used to develop parallel and distributed computing applications. Also, we identify the software tools and programming environments that will enable the development of virtual distributed computing environments and their applications. In order to provide a user-friendly and architecture-independent software programming environment, most of our research efforts shall focus on programming paradigms, programming languages, compiler technology, runtime systems, and operating systems.

In what follows, we survey four main software fields that will play an important role in the development and the implementation of virtual distributed computing environments. These areas include component-based software system, parallel/distributed programming environment, network-based visualization system, and Web-based software tools.

3.2.1 Component-based Software Systems

One of the important trends for building a large software system and exposing different characteristics within the system is based on the concept of software building blocks (also called components, modules, or objects). There are several important issues that need to be addressed in a component-based system [40]: 1)*abstraction* that refers to an abstract model for classifying all the software components; 2)*selection* that refers to a mechanism for locating a useful module from a collection of reusable components; 3) *specialization* that refers to a procedure that adapts the selected module to fit the environment; and 4) *integration* that refers to a framework that combines a set of modules in the user's application. The following is a brief description of several component-based software systems:

- *Module Interconnection Language* (MIL) [41] provides formal grammar constructs for deciding the various module interconnection specifications required to assemble a complete software system.
- *Advanced Visualization System* (AVS) [42] provides a graphical diagram for describing the interconnection of modules. Compared to MIL, which provides a formal machine-processable syntax to a user, the graphical diagram in AVS is more user-friendly.
- *Durra* [43] is a task-level description language designed to support the construction of distributed applications using concurrent, coarse-grained tasks running on networks of heterogeneous processors.
- *Conic* [44] provides support for dynamic configuration for parallel ab@layla.aub.ac.lbnd distributed programs.
- *Draco* [45] is an approach for constructing software systems from reusable software components. In particular, Draco is focused on the reuse of analysis and design information in addition to programming language code.
- *Agora* [46] is a software facility that supports the development of parallel applications written in multiple languages.

3.2.2 Parallel and Distributed Programming Environments

In this subsection, we overview several important software tools and programming environments that have been used to develop parallel and distributed applications.

PVM: Parallel Virtual Machine

PVM (Parallel Virtual Machine) is a software package developed at the **University of Tennessee and Oak Ridge National Laboratory** in collaboration with Emory University and Carnegie Mellon University. The package permits a heterogeneous collection of Unix computers hooked together by a network to be used as a single large parallel computer. Thus large computational problems can be solved more cost effectively by using the aggregate power and memory of many computers. PVM makes a user defined collection of serial, parallel, and vector computers appear as one large distributed memory computer. PVM supplies the functions to automatically start up tasks and allows the computers to communicate and synchronize with each other. Applications, which can be written in FORTRAN or C, can be parallelized by using simple message-passing constructs common to most distributed-memory computers. By sending and receiving messages, subtasks of an application can cooperate to solve a problem in parallel. The salient features of PVM are multi-language and heterogeneity support, scalability, provisions for fault tolerance, the use of multiprocessors and scalar machines, an interactive graphical front end, and support for profiling, tracing and visual analysis.

HeNCE : Heterogeneous Network Computing Environment

HeNCE (Heterogeneous Network Computing Environment) is an X-window based software environment designed to assist scientists in developing parallel and distributed programs that run on a network of computers. HeNCE provides the programmer with a high level abstraction for specifying parallelism. HeNCE is based on a parallel programming paradigm where an application program can be described by a graph. HeNCE graphs are variants of directed acyclic graphs, or DAG. Nodes of the graph represent subroutines and the arcs represent data dependencies. Individual nodes are executed under PVM. HeNCE is composed of integrated graphical tools for creating, compiling, executing, and analyzing HeNCE programs. HeNCE relies on the PVM system for process initialization and communication. The HeNCE programmer, however, will never explicitly write PVM code. During or after execution, HeNCE displays an event-ordered animation of application execution, enabling the visualization of relative computational speeds, processor utilization, and load imbalances.

MPI: Message Passing Interface

Message Passing Interface (MPI) provides a common interface for distributed memory concurrent computers and networks of workstations. MPI functionality includes point-to-point and collective communication routines, as well as support for process groups, communication contexts, and application topologies. While making use of new ideas, the MPI standard is based largely on current practice, such as IBM's EUI, Express, PVM, NX/2 Vertex, and P4.

The main advantages of establishing a message passing interface are portability and ease-of-use; a standard for message passing is a key component in building a concurrent computing environment in which applications, software libraries, and tools can be transparently ported between different machines. Furthermore, the definition of a standard provides vendors with a clearly defined set of routines that they can implement efficiently, or in some cases provide hardware or low-level system support for, thereby enhancing scalability.

The MPI standardization effort involved about 40 organizations, mainly from the United States and Europe. NPAC played major role in the MPI standardiztion activity.

Express: Parallel Processing Toolkit

Express is a collection of library routines which form a software system designed to meet the need for developing portable, and hence maintainable, parallel/distributed applications. It was based on the software originally developed by Fox's group at Caltech. It is conceptually a multi-layered system. Each of the levels are logically distinct, building only on those below it. The lowest level consists of primitive functionalities for processor allocation, program loading, and synchronous and asynchronous message passing (e.g., *send/receive*). At the next layer, global communication and synchronous routines are built on these rudimentary functions. At the highest level are the utilities, called ASPAR, designed to semi-automatically decompose problems with regular structure. As a result it is possible to port the system to a wide variety of hardware/software systems. It provides a reasonable set of tools and utilities designed for parallel processing.

P4: Portable Programs for Parallel Processors

P4 is a library of macros and subroutines developed for programming a variety of parallel machines, networks of workstations and single shared-memory multiprocessors. P4 was developed at Argonne National Laboratory. The computational models supported by P4 are: Shared memory model (monitors), the distributed memory model(message passing), and combination of the above two models.

For the shared memory model of parallel computation, P4 provides a set of primitives from which monitors can be constructed. For the distributed memory model, P4 provides typed send and receive operations, and creation of processes according to a process group file describing files to be executed by created processes and the machines that will execute them. P4 ensures easy portability to a wide range of architectures/platforms and its functions are simple and easy to comprehend. P4 does not require the presence of daemons on machines before any application can run, unlike many other tools. P4 offers features for automatic and user supervised process creation and message passing. It also allows creation of multiple processes/nodes on a single machine to be executed in a time-sharing fashion.

Cronus

Cronus is a software system that runs simultanesouly on multiple heterogeneous computers interconnected by a computer network [6]. The object model in Cronus allows to abstract system resources so the system seems like a single, flexible computing system. Cronus provides a standard representation for most common data types and several layers of standard message passing protocols that implement the object model uniformaly on all the machines in the Cronus system. The basic Cronus system services include a hierarchical name space, authentication and access control, processes, and clusters. Cronus softwre is usually implemented as a user process on top of the native operating system and this allows users to access local resources using local operating system services and remote resources using Cronus services.

HPF : High Performance Fortran/Fortran 90D

High Performance Fortran (HPF) [8] is based on the research language Fortran 90D developed jointly by Syracuse University and Rice University and has the overriding goal to produce a dialect of Fortran that can be used on a variety of parallel machines, providing portable, high-level expression to data parallel algorithms. The idea behind HPF (and Fortran 90D) is to develop a minimal set of extensions to Fortran 90 to support the data parallel programming model (defined as single threaded, global name space, loosely synchronous parallel computation). Extensions incorporated into HPF/Fortran 90D provide a means for explicit expression of parallelism and data mapping. These extensions include compiler directives which are used to advise the compiler on how data objects should be assigned to processor memories, and new language features like the *forall* statement and construct.

HPF/Fortran 90D adopts a two level mapping using the PROCESSORS, ALIGN, DISTRIBUTE, and TEMPLATE directives, to map data objects to abstract processors. The data objects (typically array elements) are first *aligned* with an abstract index space called a *template*. The template is then *distributed* onto a rectilinear arrangement of abstract *processors*. The mapping of abstract processors to physical processors is implementation dependent. Data objects are not explicitly distributed. These objects are mapped according to an implementation dependent default distribution (e.g. replication). Supported distributions types include BLOCK and CYCLIC.

Other important programming tools and environments include the following:

ISIS Distributed Programming Toolkit [51] provides high level tools for developing reliable distributed applications.

Linda [52] consists of a few simple operations designed to support and simplify the construction of explicit-parallel programs. Linda centers on an idiosyncratic memory model in which programs communicate through a logically shared associative memory called the tuple space.

Mentat [53] is an object-oriented parallel processing tool based on C++. Mentat is designed to exploit the capabilities of both humans and compilers. The user is responsible for identifying object boundaries and specifying those object classes that have sufficient computational complexity to warrant parallel execution. The Mentat runtime system is responsible for managing all aspects of communication, synchronization and scheduling for the user.

SISAL [38] is a general purpose functional language for parallel numeric computation. Constructions in SISAL are used to express scientific algorithms in a form close to their mathematical formulation with no explicit program control flow.

3.2.3 Network-based Visualization Systems

Visualization itself requires different software technology and hardware devices. As a result, combining existing visualization tools and powerful computation resources into a distributed system is one of the important trends in both academic and industrial communities. Below is a brief description of several ongoing projects.

- *An extension for AVS (Advanced Visualization System)* developed by [54] is to provide a heterogeneous remote procedure call facility that executes the computation task in a remote machine. To carry out the heterogeneity in different hardware platforms, AVS provides three services: the Universal Type System (UTS), a collection of stub compilers, and a runtime system. In this environment, a module programmer has to hardcode the server name into the module specification as well as the end-user responses to select the site at runtime.

- The *ANL Visualization Project* [55] creates a system to implement common scientific visualization toolkits on the IBM SP-1 and SP-2 system. The target application is focused on the terabyte-sized datasets produced in Grand Challenge applications. Currently, AVS is used as a development tool for this project.
- *The Advanced Visualization Research Project (AVRP)* [56] is a visualization environment that involves different research projects on parallel graphics algorithms and visualization system integration. The AVRP software system is supported on a wide variety of vendor hardware, such as Silicon Graphics, DEC, Sun, and Stellar platforms. One of the goals of this project is to integrate all the AVRP tools into a common environment through a language-based interpreter.
- *Research on Interactive Visual Environments (Rivers)* [58] is developing hardware and software systems for extending high-end 3D visualization from a batch process to an interactive process, and for visualization-based interactive steering of supercomputing simulations in a high-performance distributed environment. The modular software subsystem of Rivers is executed across multiple, heterogeneous hosts. The communication network of Rivers is based on a 50-megabit-per-second HyperChannel network, an 80-megabit-per-second ProNet-80 network, and 100-megabit-per-second FDDI backbone network.

3.2.4 Web-based Tools

Web technologies offer a viable standardization, publication and productivity platform for National Information Infrastructure (NII) as well as for VDCE software engineering. Current Web Productivity Tools include free, low cost or bundled Web browsers such as NCSA Mosaic, Netscape and Internet Explorer, quality HTML editors such as HotMetal, Internet Assistant and WebMagic, database interfaces, and early collaboration tools. New generation tools, including interactive Web extensions such as Java and VRML will offer rich functionality in the area of text processing, media authoring and GUIs.

One of the main goal of this effort aims at investigating the use of a hybrid approach to integrate World-Wide Web (WWW) technologies with the current portable scalable software systems developed by the HPCC Initiative for Engineering and Scientific simulations. This combines the parallel languages and runtime of HPCC with the software engineering, collaboration and pervasive technology base of the WWW. The resultant system will support collaborative rapid prototyping, applied to problem solving environments (PSE) at different scales. Support will extend from systems such as ELLPACK (parallel partial differential equation PSE), NWChem (parallel computational chemistry PSE), MATLAB (uniprocessor matrix and signal processing PSE), or Khoros (image processing/visualization PSE) in the simulation arena to complex National Challenges including education, manufacturing and healthcare. Roughly we consider the world-wide metacomputer as a set of future generation WWW servers, each of which (individual workstation, cluster, SMP, or MPP) is implemented internally (in the closely-coupled homogeneous environment) using appropriate HPCC technologies (e.g. HPF, MPI, PVM, Fortran-M, pC++, CC++, parallel Oracle, etc.). Then we link these servers together using generalized WWW technologies to allow executable program components to be published as services, and so create a distributed problem solving environment for prototyping large scale software.

In Section 5, we discuss an on-going project at NPAC, WebWork, as an important approach that can be used to develop virtual distributed computing environments. WebWork is a collaboratory multi-user multi-server problem solving environment on the Internet, based on the existing HPCC technologies for local computational backends and the evolving WWW technologies for human interfaces, system-wide coordination and world-wide dissemination. It supports collaborative rapid prototyping, applied to programming environments at different scales ranging from support from a single PC running a WWW server and client, to the enterprise computing systems of a corporation or a university department, to the NSF Metacenter and even multi-national, world-wide collaborations.

3.3 Communication Networks

One basic requirement of distributed computing is high-speed interprocess communications over the underlying network. Fortunately, the transmission speed in communication networks increased by several orders of magnitude over the past decade as shown in Figure 2. One recent trend is focused on building networks in which the physical communication medium has a peak bandwidth on the order of 1 Gb/s or higher.

Communication networks with high latencies and low application bandwidths have limited the applications that can efficiently run on network-based parallel/distributed computing environments in the last decade. The current advances in fiber technology have been able to stretch transmission rates from 8 Gbits/sec for 30 km distances in 1985 to over 10 Gbits/sec for 350 km distances in 1990 (a 16-fold improvement in 5 years). These advances have lead to aggressive research in the development of high speed computer networks and transport protocols.

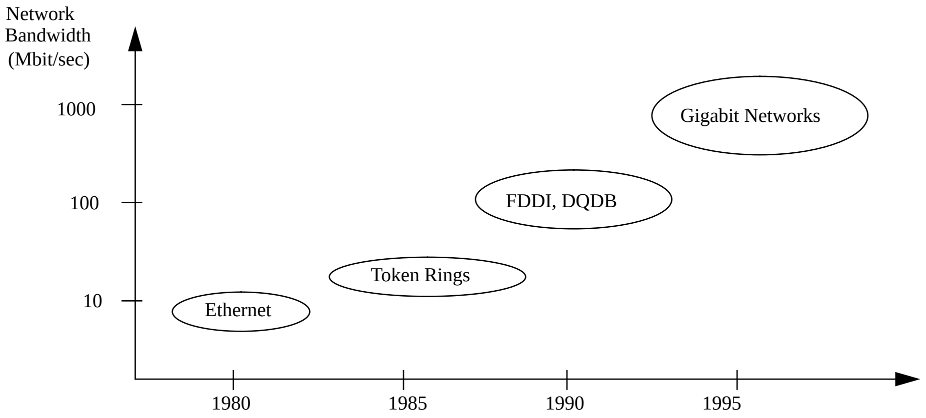


Figure 2: Trends in network technology

Furthermore, with the current movement towards the standardization of Gigabit LANs (ATM, Fiber Channel, and HIPPI), high-speed networks have received increasing industrial support. These trends will make high speed networks affordable and widely available. The advances in high-speed network technologies and their standardization will be a major driving force for virtual distributed computing environments in the future.

3.4 VDCE Application Requirements

One important class of applications requiring VDCEs includes C3I applications, High Performance Computing applications and National and Grand Challenge problems. In general, these applications are characterized by massive data sets and complex operations that exceed the limits of any single high performance computer. In what follows, we first describe the requirements of C3I applications and then briefly examine the characteristics of VDCE applications in terms of the following attributes: computation, storage, visualization, heterogeneity, parallelism, communication, and synchronization.

3.4.1 C3I Applications Requirements

The role of Command and Control with a dynamic combat environment is of critical importance to successful mission accomplishment. In a crisis situation, the very short times available for military personnel and national leaders to react and make decisions has necessitated the use of what is known generically as Command and Control systems. The Command and Control model consists of two major components: command elements that issue orders and control elements that execute them. The command element includes the authority and responsibility for effectively using available resources and directly controlling and coordinating the mission forces.

Input data are feed into the command elements and are filtered, refined and processed to yield an estimation of the state of the data. The control element is the specific fundamental activities designated to accomplish the mission. The estimated data is utilized as feedback to the control elements for subsequent analysis, execution or direction to perform the appropriate actions or responses. This Command and Control process is iterative in nature, and as new data or information enters this process, yields an increasingly better data estimation.

As an example of the data input task, consider a battlefield situation where a three-dimensional action is occurring. The first dimension is distance, the second, and more significant, dimension is time. The third dimension is operational. In order to extend the battlefield forward in time and distance, control systems are placed at appropriate operational facilities supporting the battlefield functional areas. Both horizontal and vertical information flow is accommodated within the concept. Intelligence support is the foundation on which the operational effort is built. The first objective is to keep the commander informed of the characteristics of the area of operation. A second objective is to assist the commander in planning and implementing activities. The result of the Command and Control activity is an ability to rapidly convey decisions and to direct alternative courses of action.

As the amount of input data increases and time requirements become more stringent, the input function has become automated and is referred to as data fusion, multisource correlations or multisensor integrations. In today's complex Command and Control application systems which include applications such as air traffic control, missile attack warning, and vessel traffic control, it is essential to introduce faster and more powerful computing environments to process the ever increasing magnitude of input data. The proposed virtual distributed computing environment to be investigated in this project will provide a powerful, cost-effective, fault-tolerant computing environment for large scale C3I applications.

3.4.2 Characteristics of VDCE Applications

Computation Requirement

In general, solving large scale VDCE applications is expected to require raw computational power between 100 and 1000 billions of operations per second. For example, [62] shows the estimated computer requirements for several types of problems in turbulence physics research. To compute a flow over an entire aircraft using the complete Naiver-Stokes equations without approximation, a computer with a processing speed of 10^{19} operations per second is required to reduce the simulation time down to within 200 hours. Even in the approximation approach of flow analysis, such as large-eddy simulation, the expected computer speed is still in a teraflop per second range.

Storage Requirement

VDCE applications usually generate and/or might require to access a terabyte of data. For example, a mathematical model could produce large scale numerical data and graphical output for the purpose of simulation. However, an image understanding system could demand 2 or 3 dimensional, colored, high-resolution pixel data for lower-level processing. Consequently, two basic storage requirements for solving these applications are large-scale storage capacity and high-performance access mechanisms. Large-scale storage capacity in main memory and extended memory is used to keep pace with the I/O data size. High-performance access mechanisms (a low latency for scalar processing and a high bandwidth for vector processing) are used to keep pace with the CPU speed.

Visualization Requirement

Visualization of scientific data sets has become an important tool that allows scientists and engineers to efficiently and compressively understand the system behavior. In addition to the need of high-performance computing and storage, visualization itself requires special software technology and hardware devices. For example, most technology for displaying medical data fall into the three broad categories of surface-based rendering, binary voxel rendering, and volume rendering [63]. Each technology will involve different intermediate geometrical representations and different algorithms. Furthermore, specified hardware devices, such as stereo viewers, varifocal mirrors, cine sequences, real-time image-generation systems, and head-mounted displays are required for displaying the images. Obviously, it is impossible, or inefficient, to integrate all the software technology and hardware devices into a single physical machine.

Heterogeneity Requirement

The nature of the computations and communications characteristics of large complex applications is heterogeneous. For example, an image understanding system [64] [65] is usually classified into three levels (low, intermediate, and high) based on the data processed at each level. Basically, image data, or corresponding arrays of numerical data, are processed in the lower level. Extracted image event, which is represented as symbolic description, is processed in the intermediate level. Finally, the high level will handle the knowledge data, such as semantic networks. Because the requirements at each level are very different in computation, communication, and control, each level demands different types of computer architecture, such as SIMD architecture for image processing in the lower level and MIMD architecture for semantic network traversal in the high level. Heterogeneity requirement has also been demonstrated in applications, such as computer robot, speech recognition, and C^3I (command, control, communication, and intelligence).

Parallelism Requirement

Parallelism has been exploited (from coarse-grain to fine-grain) into *five* different processing levels [66]: a) job or program level; b) subprogram or job step level; c) procedure, subroutine, task, or coroutine level; d) non-recursive loop or unfolded iteration level; and e) instruction or statement level.

Each parallelism level (or granularity) has different characteristics and requires different approaches to exploit parallelism at that level. For example, fine-grain parallelism (levels (d) and (e)) usually requires a parallelizing compiler and SIMD computer. On the other hand, coarse-grain parallelism (levels (a) and (b)) usually requires effective operating system and MIMD computer. However, a combination of these levels may be required to achieve the best performance. Consequently, a general-purpose parallel computer should be able to support all the features used to exploit parallelism that range from fine-grain to coarse-grain.

Communication Requirement

Communication patterns in different applications (*regular* or *irregular*) not only provide a strong impact on selecting the best computer architecture, but also affect the performance of partitioning, mapping, and scheduling tasks. Regular communication patterns, such as butterfly operation in FFT, nearest neighbor communication in Laplace's equation, or all-pairs communication in N-body problem, can be easily satisfied by mesh, ring, hypercube or other such architectural topologies. On the other hand, irregular communication patterns (sparse matrix computation, combinatorial searching, and discrete event simulation) are more difficult to map to existing architectures.

Synchronization Requirement

Fox [67] presents a classification for problem architecture which can be used to identify appropriate hardware and software paradigms. Fox's classification is based on a break up of each problem into *spatial* (data) and *temporal* (control) aspects. The temporal structure of a problem is analogous to the hardware classification into SIMD and MIMD. The spatial structure of a problem is analogous to the topology of the hardware. The problem structures can be classified into five categories: 1) synchronous, 2) loosely synchronous, 3) asynchronous, 4) embarrassingly parallel, and 5) loosely synchronous complex.

Each problem structure requires different hardware and software to solve it. For example, synchronous problems are inherently data parallel and therefore it maps naturally into a SIMD architecture. Asynchronous problems are inherently functional parallel and thus they map naturally into a MIMD architecture.

3.5 Discussion

Advances in software technology have produced tools and environments for the development of parallel and distributed applications. Also, the advances in high performance computer technology have resulted in the proliferation of a large number of different architectural classes like SIMD computers, MIMD computers, vector computers, etc., where each class represents a set of different trade-offs in design decisions. Each architectural class is tuned to deliver maximum performance to a specific set of applications, however, none of the existing high performance computing systems are general enough to address all classes of applications and provide the desired performance levels. Applications that require the resources of a supercomputer are quickly reaching the point where it will be difficult to find a single computing platform on which they may be executed. As shown in Figure 3, large scale HPCC applications of 1995 and beyond, will require a memory capacity of 1000GB and system speeds of 1 Tflop [68]. Given the current pace of technological advancement it is unlikely that a single supercomputer will be able to provide the resources required for these problems. A network of supercomputers and high performance workstations integrated into one single virtual distributed computing environment appears to be the only reasonable way to provide adequate computing resources for these large applications. The main goal of this project is to identify and evaluate VDCE technologies and develop a conceptual model to implement VDCEs and their applications.

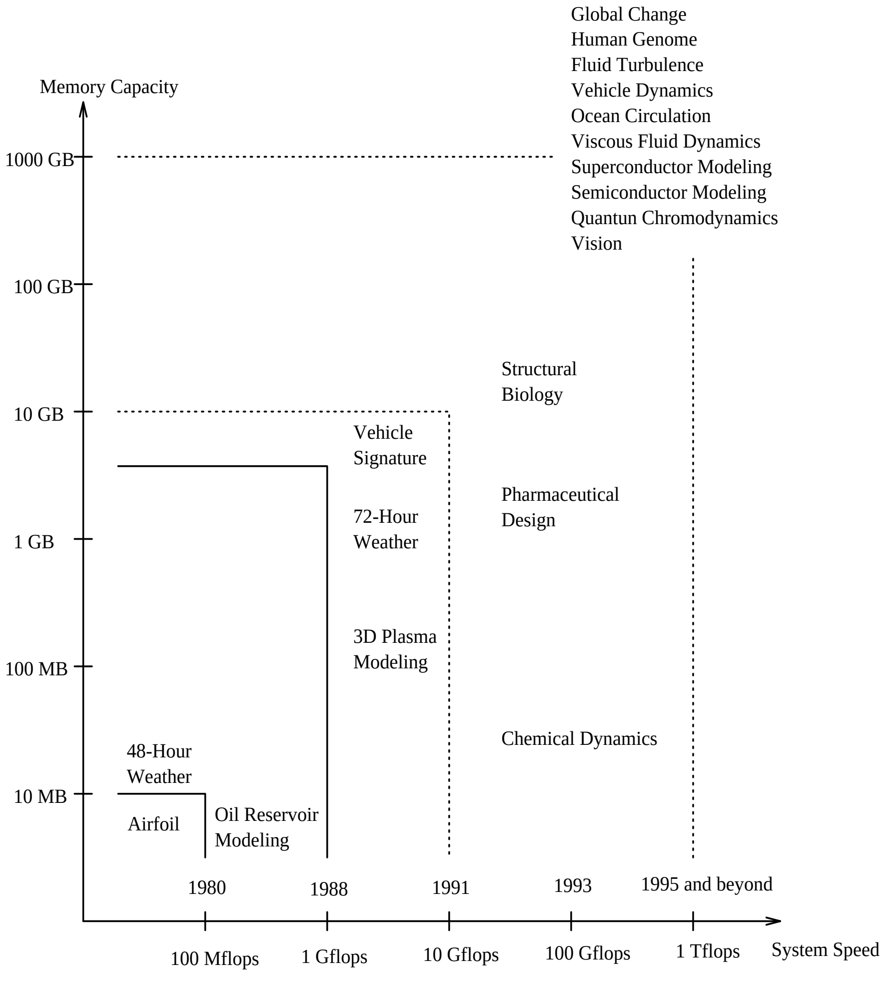


Figure 3: Computing and Storage Requirements of Future Applications

4 VDCE Conceptual Development Model

The main design philosophy used in creating the VDCE is to utilize existing programming paradigms, programming languages, computer architectures, and physical machines, to provide high performance computing for a variety of applications. By using existing technology whenever possible system designers will be able to focus on providing a new computing environment rather than reimplementing basic functions. Running existing tools and system software will eliminate compatibility problems and will make it easier to use existing programs as modules in a VDCE application. The VDCE will allow users to develop and run applications that require more resources than can be reasonably provided by a single computer. At the highest level, the VDCE can be divided into two major modules: the Software Development Module (SDM), and the Execution Module (EXM). Each module, in turn, is broken down into several layers as shown in Figure 4. In what follows, we describe the main functions and the services of each of these two modules.

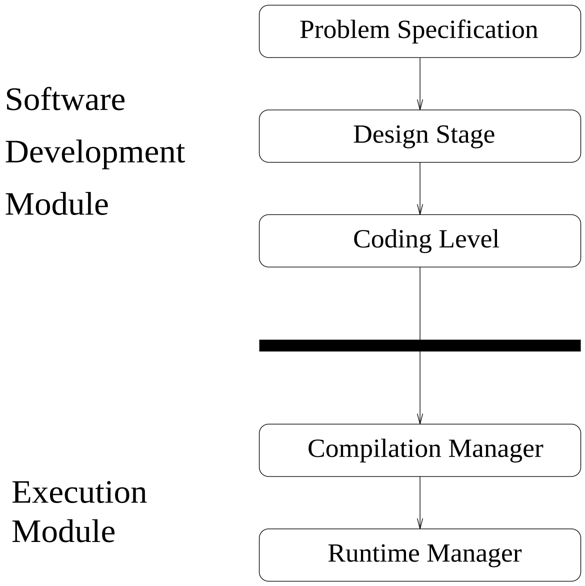


Figure 4: VDCE Conceptual Development Model

4.1 The Software Development Module

The SDM provides the tools required to develop a VDCE application. The primary purpose of this module is to develop, test and evaluate the performance of the VDCE application. The tools provided by the SDM are similar to the CASE tools that are available for standard programming environments. The SDM consists of three layers, each of which is responsible for attaching specific information to the application graph. The information contained in the completed application graph will include: 1) Implementation language; 2) Input and output requirements; 3) Hardware requirements; and 4) Optional user supplied information.

The *problem specification* layer is responsible for extracting the requirements of the VDCE application to be solved and formalizing its functional flow. This is accomplished by creating the initial application graph.

The *design stage* is responsible for analyzing the computational needs and the existing dependencies for each task in the application graph. The analysis performed by the design stage is based on Fox's [69, 70] work on the architecture of problems and portable parallel software systems. The parallel software design methodology used in the design stage concentrates on the architecture of the problem and not the machine. The machine architectural features will be captured by other lower stages. There are three broad classes of problem architectures: synchronous, loosely synchronous, and asynchronous, which describe the temporal (time or synchronization) structure of the problem. The temporal structure of the problem is analogous to the hardware classification into SIMD and MIMD. Other classes that capture the nature of the application, such as graphic or interactive, will be used to assist the lower layers in determining the best application to map tasks onto available machines in the VDCE.

The graph produced by the design stage feeds into the *coding level*. In this stage, the application is parallelized using architecture independent languages that allow tasks to be implemented using functional, data parallelism or both. The software tools and languages to code and parallelize the application at this level will be based on emerging standards (High Performance Fortran [71], High Performance C++ [72], etc.). Support for architecture independent communication between tasks will be provided via standard communication libraries (e.g. MPI) and the semantics of object-oriented method invocation.

As the application graph is annotated in the various layers of the SDM, the user will be able to record information about the nature of the application that will be useful to the compilation and runtime managers. These *hints* will allow the execution module to do extra optimization. For instance, suppose a particular application has three functionally parallel modules and the user expects one to run much longer than the combined running times of the other two. If the system is aware of this, dispatching of the longer job can be given higher priority so opportunities for parallel execution will be maximized. The user will not be required to supply hints, since the execution module will have access to information about an application such as implementation language, executable size and input and output requirements.

4.1.1 Illustrative Example: Lake Effect Snow Modeling

During the winter in central New York, lake effect snow storms are of great concern. Lake effect snow occurs when cold northern air blows over Lake Ontario. As the air passes over the lake it picks up moisture. When the moisture laden air reaches land, the difference in temperature between the lake and the ground forces the moist air to rise. As the air rises it cools which reduces its ability to hold moisture. The moisture then returns to the ground in the form of snow. Lake effect snow storms can be quite severe, creating as much as two feet of snow in one hour. Lake effect storms typically cover a very small area and can be very unstable. The slightest change in wind direction or speed can change the intensity of a storm and the area which will receive the greatest amount of snow. Many utility companies and emergency services are especially interested in the ability of computers to model lake effect storms.

Several computer models of lake effect snow have been developed. One model, currently under development at Syracuse University, will be used to illustrate our approach to develop the VDCE. At the heart of the model is a data parallel program that uses current weather conditions to predict the behavior of lake effect snow. The output of the model is displayed on a graphical display where the effects of the storm can be viewed. Several other computers analyze the current weather conditions and present this information to the prediction engine. The lake effect snow model is an example of a typical VDCE application, where several high performance workstations and supercomputers are involved in its solution. In what follows, we show how various layers of the VDCE SDM can assist in the designing and the development of a VDCE application.

Problem Specification

In the problem specification stage the initial application graph describing the application is created. In this application, the application graph consists of five tasks as shown in Figure 5. Three tasks collect data about current weather conditions. This information is fed into a fourth task that represents the lake effect snow prediction engine. The output from the prediction engine is processed by a fourth task which is responsible for displaying the predicted snow fall on a graphical display. Note that only descriptive information is presented in this application graph. The graph serves to break the application into functional components and illustrate how the components interact.

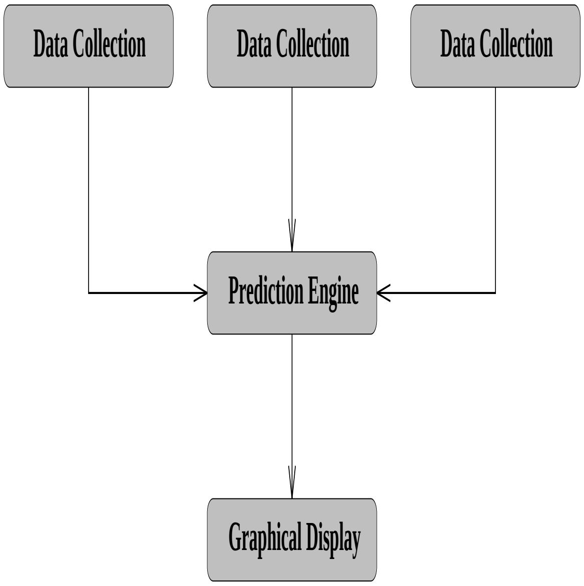


Figure 5: Lake Effect Simulation Application Graph

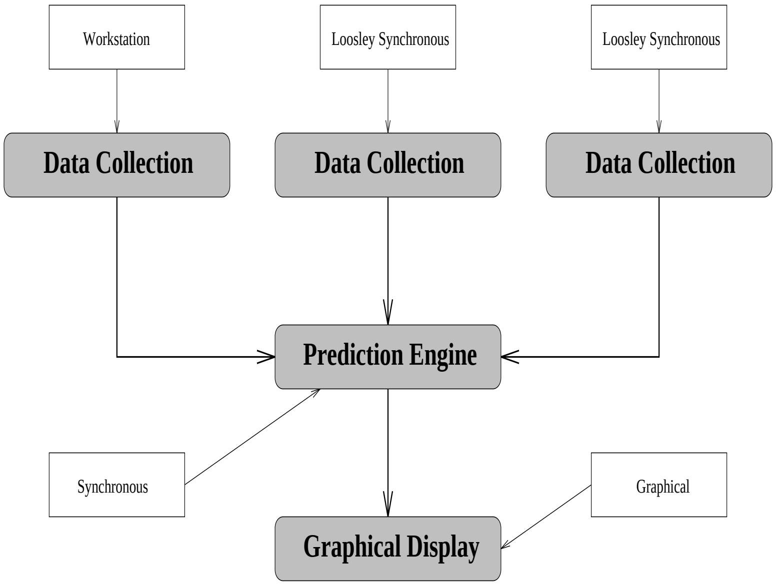


Figure 6: Annotated Design Stage Application Graph

Design Stage

At this level the annotated application graph, which is shown in Figure 6, contains additional information which describes the nature of the computation performed by each of the tasks in the application graph.

This application uses several different classes of machines. The data collection tasks can

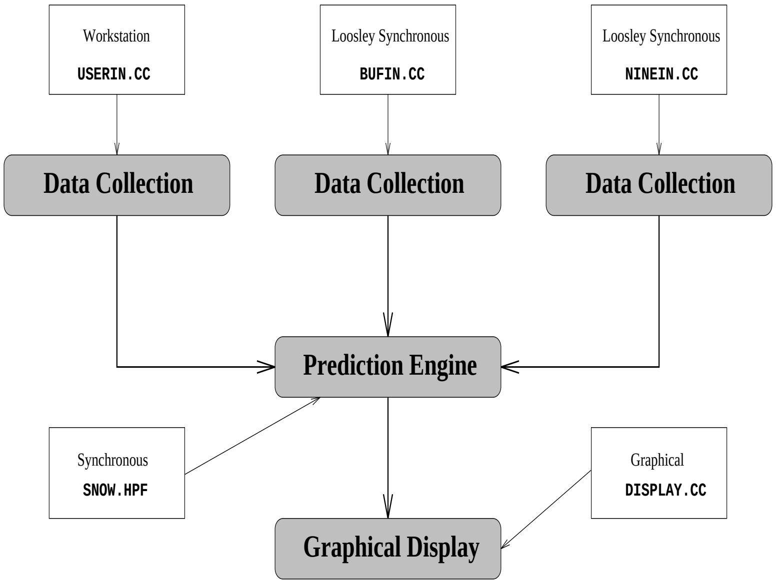


Figure 7: Snow Model Coding Level Application Graph

be divided into two categories. The first includes a task that can run on a workstation. Presumably this task is interactive in nature and requires the graphical user interface provided by most workstations. The remaining two data collection tasks require machines capable of supporting loosely synchronous parallelism. Although not explicitly stated, these tasks themselves may also be VDCE applications. The snow simulation runs on a data parallel machine, and the display requires the use of a machine capable of displaying high-quality graphics. Again note that although the description of the application has been refined, it is still not dependent upon any specific computer architecture.

Coding Stage

In the last stage of the software development process the programmer attaches code to the application graph. In Figure 7 the names of the files containing the source code are given. Information regarding compilation instructions and “suggestions” for running the application may also be specified at this level.

4.2 The Execution Module (EXM)

Once a VDCE application is developed and described according to the coding layer of SDM, the next stage involves the execution and the management of the VDCE application. The primary purpose of the EXM module is thus to run and manage the execution of VDCE applications. Consequently, the EXM tasks consist of two distinct steps: preparing executables, and managing the execution of the VDCE application. Hence, the EXM module can be described in terms of two managers: Compilation and Runtime managers. The compilation manager handles all the issues related to preparing executables required to run VDCE applications. The runtime manager is a distributed application whose components are running on each of the machines in the VDCE network. These components accept both local and remote execution requests from VDCE users. In what follows, we identify the main issues that must be investigated in the design and the development of the EXM compilation and runtime managers.

4.2.1 Compilation Manager

An important issue that needs to be addressed at the EXM module level is when compilation of the source code should take place. Part of the power provided by the VDCE lies in its ability to dynamically select machines on which to run tasks, the ability to migrate processes from one architecture to another, and support for fault tolerance. Most systems that provide similar features do so for homogeneous environments. The VDCE is designed to use heterogeneous platforms which makes these features harder to support. A single VDCE task can be executed on several different hardware platforms, each requiring different binaries. The compilation manager must select the machine, or machines, on which each task should or could run but there is no guarantee that every task will map to exactly one machine. In fact in most cases several different machines may be used to execute a particular task. In this case the compilation manager prepares executable images for all possible machines. The choice of which machine to run a given task will be made by the runtime manager. By preparing all possible executables before the execution of an application, the runtime manager will be able to move a given task among various machine architectures without the need to compile a task while the application is running.

4.2.2 Runtime Manager

The *runtime manager* is responsible for managing the execution of a VDCE application. The basic service provided by the runtime manager is selecting the "best" machines on which to run the various tasks, loading the corresponding binaries, and initiating the application execution. The selection of the target machines is based on the current load and the machines availability. While the application is running, the runtime manager will migrate tasks to less loaded machines, and provide fault tolerance, if required or requested by the user. In what follows, we discuss the main issues that must be addressed by the VDCE runtime manager.

Task Placement

The scheduling mechanisms used in each computer will seek to achieve two, sometimes conflicting, goals: 1) Schedule work on idle machines to maximize hardware utilization; and 2) Execute tasks on the best available platform. Where best available platform means the computer best equipped to satisfy the task's requirements in terms of processor requirements, architecture requirements, file requirements, etc.

The runtime scheduling algorithms will give preference to schedules that maximize overall performance of one or more VDCE applications. Krueger [81] showed that processor utilization rates in a distributed system could be improved significantly through careful use of non-preemptive process placement techniques. Such techniques may not be appropriate in virtual ditributed computing environments. Virtual distributed computing applications may provide enough tasks to consume all the excess capacity in a system. The philosophy of Clark [82], Ju [83], and Krueger [81], is to suspend remotely initiated tasks when resource requirements of locally initiated processes increase. Execution of remote tasks is resumed when activity of locally initiated tasks diminishes. This approach reduces the frequency of process migrations and yields high resource utilization without requiring an efficient process migration mechanism.

Communication

Communication between tasks will take place either through primitives defined in a message passing tool or via object-oriented method invocation semantics. The compilation manager will provide a number of different libraries that will map application communication primitives to the proper communication tools available in the VDCE. In addition to providing a map between the abstract and low-level communications primitives, these libraries will provide the runtime manager with the ability to monitor, redirect, and move connections between tasks.

The channel model can be used to connect VDCE tasks. A channel is a logical transport medium that connects possibly many tasks sending and receiving messages. Channels are distinct from the tasks that are connected to them, and thus readily support messaging directed to groups and/or single tasks without requiring that clients use different forms of message addressing. In channel-based systems, clients may be unaware of whether messages are being received by groups or individuals. The runtime manager may split channels, interposing other tasks between senders and receivers to deal with issues such as authentication or data conversion. Channels will be connected to tasks through *ports*. The runtime manager will be responsible for the creation, placement, and destruction of ports.

In systems where objects reside on a single processor, communication is typically imple-

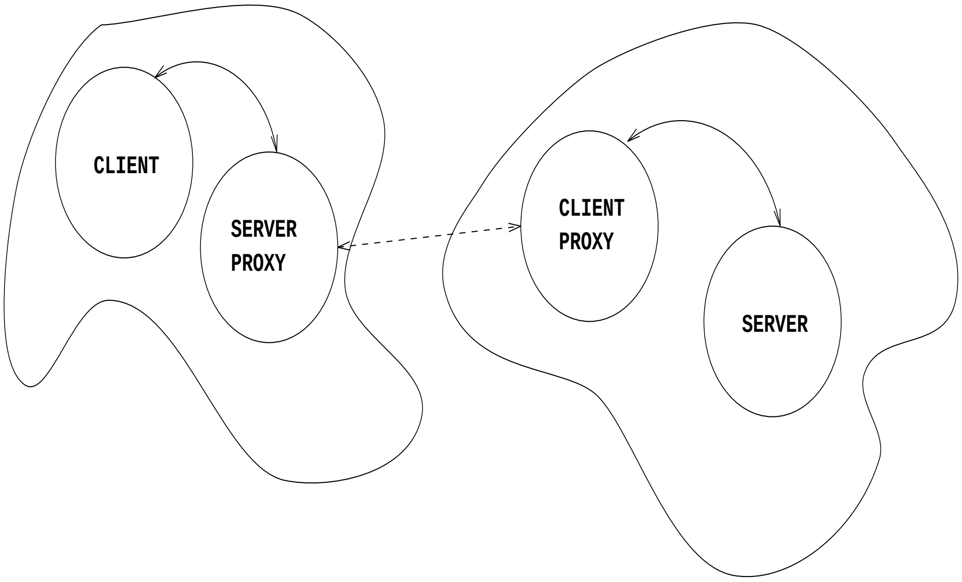


Figure 8: Communication via Proxies

mented using standard procedure calling conventions. For objects residing on different processors, the communication between objects could be implemented using message passing. In this case the client object and a *server proxy* would be placed on one processor, and the server object and a *client proxy* on the other. The role of the proxy is to receive messages, translate information into architecture independent form, and forward the result to the corresponding proxy on the other processor, as shown in Figure 8. Proxies will be generated by the compilation manger when needed, using a tool such as the IDL compiler provided by the Object Management Group [80]. The generated code will use VDCE channels to exchange information with proxies running on other machines.

Load Balancing

The ability of the runtime manager to maximize resource utilization and system throughput will be severely, perhaps fatally, restricted if task placement decisions must be made before task initiation and cannot be altered later. To achieve the system objectives, we do need to support: 1) Load balancing techniques that determine which tasks should be running and on what machines; 2) Remote execution mechanisms to dispatch and relocate tasks on the machines that can best accommodate them.

Many papers have studied these two issues, but the significance of their results is undermined by its lack of generality. Many remote execution systems are built to support a particular load balancing philosophy [81] [84] [85]. Furthermore, they also tend to make restrictive assumptions about the homogeneity of the underlying network and the nature of the tasks being migrated. For instance, Litzkow [84] describes a system that can migrate long running, compute intensive, non-communicating jobs across a homogeneous network. Such a system would not be robust enough to provide the functionality required by the VDCE runtime manager.

The literature on load balancing does not address the remote execution problem. Load balancing algorithms are often validated on homogeneous systems using tasks that are easily migrated (like parallel Monte Carlo simulations [85] or big batch jobs [84]). Load balancing algorithms are rarely tested on heterogeneous networks running heterogeneous applications that will be the typical applications to run on VDCEs.

To provide the most robust VDCE execution environment and provide flexibility to implement any workload balancing scheme, the VDCE runtime manager should implement a variety of process migration schemes. The following list outlines four process migration approaches that will be considered in the VDCE runtime manager:

- **Process migration through redundant execution.** Dispatch the same task on several idle machines. If one of those machines gets busy with other work then kill the incarnation of the redundant task on that machine. This approach eliminates process migration overhead; by killing a task and using an already running redundant copy on another computer avoids the communication overhead of moving that process and its state information over the network. Furthermore, this approach provides high performance and fault tolerant execution as the VDCE application we design takes the results from the fastest machine.

- **Process migration through checkpointing.** Migratable tasks are checkpointed regularly. To migrate a task we first stop that task and start it somewhere else by instantiating the new incarnation from the last checkpointed state. This approach is expensive and may require the cooperations of all the application tasks [86]).

- **Straight Process migration.** To migrate a task we dump the contents of the address space, copy it to a new machine and then restart it at that new machine. The main drawback of this approach is the homogeneous requirement [84].

- **Process migration through recompilation.** This is an expensive approach but could be robust. This approach has been studied by only few researchers and it is believed to have a difficult implementation [87].

The VDCE runtime manager should have several of these techniques in its repertoire. Which of these will be used for any particular migration will depend on the state of the system and the characteristics of the VDCE task(s) involved.

Processor Utilization

Two related concepts should be employed by the VDCE runtime manager to achieve high processor utilization: *free parallelism* and *anticipatory processing.*

Free parallelism is based on the observation that when parallel processes are running on otherwise idle machines, efficiency is not an important measure of performance, only speed-up needs to be considered. If 100 idle machines are available and the only way to use them is to distribute a single application over all 100 machines to realize a 10% speed-up; it is still advantageous to adopt this approach because the 10% speed-up comes for "free". Many distributed computing environments contain significant numbers of idle or nearly idle computers most of the time. In these environments the runtime manager should attempt to allocate userful work to these idle machines.

Anticipatory processing is a method for using idle processors to increase system throughput even when there are no dispatchable VDCE tasks available to exploit them. It involves using idle processors to perform processing that may or may not be required in the next phase of the application execution. For example, suppose there is a VDCE application consisting of two modules where the second cannot start until the first completes. If there are lots of idle resources in the system they can be used to do some computations that might be needed when the second module runs. If the second module is a C program we could compile it on several computers that can run that task if the executables are not available on these candidate machines. Also, if the second module requires input files other than the ones produced by its predecessor first module, we could use idle processors to read those files at the candidate machines that might run the second module when it becomes dispatchable.

5 NPAC VDCE RELATED RESEARCH PROJECTS

The researchers involved in this project have also been involved in several on-going VDCE related research projects that aim at investigating different technologies to implement the main building blocks of a virtual distributed computing environment. We have been evaluating three approaches to develop VDCEs: Procedural-based, Object-based and Web-based approaches. We have implemented two prototypes of the VDCE execution modules and evaluated two types of scheduling schemes: Selective and Redundancy Task Scheduling. The results of these research projects have been published in several research papers that are included in the attached Appendix. In what follows, we highlight the main features of these research projects or prototypes.

5.1 Procedural-based Approach

One of the main goals stated in our early effort to study VDCEs was to use existing distributed computing tools as much as possible. This approach was taken to minimize the time and effort needed to construct prototypes and reduce their complexity. As a result, we have developed two prototypes that used commercially available message passing tools to perform the required communications. One prototype used ISIS message passing tool while the second prototype used P4 message passing tool. The ISIS-based prototype is a workstation based system which used ISIS to implement several VDCE constructs including *group leaders*, *execution processes*, *load measurement*, and *redundant low-priority scheduling*. The P4-based prototype focused on integrating all the high performance computers available at NPAC into one VDCE. In this prototype, we have developed *application description framework* and facilities for performing *redundant normal priority scheduling* on supercomputers and high performance workstation clusters.

5.1.1 ISIS-based Prototype

The first VDCE prototyping effort was performed in the summer and early fall of 1994. The goal of the effort was to gain experience implementing some key VDCE components and try to quantify the benefits of redundant scheduling. Because this was the very first VDCE prototype project, and would have to be built entirely from scratch, the project's scope was limited to a single Sun workstation cluster. This prototype fully implemented the concepts of *group leaders*, *execution processes* and *scheduling daemons* described in [88]. That is, each workstation participating in the VDCE ran a daemon process which monitored the load on that machine. When the load on both the I/O system and CPU of a machine were low the daemon notifies the group leader. The group leader maintaines a list of lightly loaded workstations. The execution process would manage execution of an application by obtaining the list of lightly loaded machines from the group leader. The execution process would then directly contact the appropriate daemons and instruct them to run the application tasks. The daemons had the capacity to begin and cancel the application tasks. They were also programmed to notify the execution process when a task is completed normally.

Because this prototype ran on a network of SUN workstations which were normally used by students and faculty, all the VDCE components, except the execution process, ran at the lowest possible priority. Experimentation with the prototype showed that limited use of redundant scheduling achieved good performance in spite of the low dispatching priority. The principal limitation of the ISIS prototype was that the environment supported only SUN workstations and could not include the high performance parallel computers available at NPAC. Another limitation of the system was that it did not provide any communication infrastructure for use by VDCE applications. This limitation is shared by the P4 prototype. More research is needed to develop an efficient VDCE communication infrastructure. In a previous paper [88], we describe a channel based communication system that can be used in VDCE.

5.1.2 P4-based Prototype

The second VDCE prototyping experiment was performed in the first half of the spring semester of 1995. This prototype used most of the parallel computers (CM-5, IBM SP2) and high performance workstations (DEC ALPHAs, SGIs, and SUNs) available at NPAC. During the HPDC Demonstration Session of the Fourth IEEE International Symposium on High Performance Distributed Computing (August 1995), we have demonstrated successfully how this prototype can optimize the execution of a C3I application (Concurrent Multitarget Tracking) using two scheduling techniques: Redundancy and Selective Scheduling.

In this prototype, we developed an *application description framework* that allows computing tasks to be specified in terms of their inputs, outputs, node requirements and executable files. Figure 9 shows the syntax of task descriptions. Task descriptions can then be used to build a large scale application according the the application syntax shown in Figure 10. This application description syntax allows different node requirements and executable files to be specified for each platform where the task can be executed on. These task specifications are collected in a file called a *task description.* A separate *application description* indicates how a set of tasks can be managed by the VDCE to realize an application. Most of the information in the task description (except the executable file names) can be overridden in the application description. In other words, the task description sets out default values which can be refined

[**task:** task_name]

[**executables:** architecture: path_name
<architecture2: path_name2>
< ... >]

<**nodes:** architecture: number_of_nodes
<architecture2: number_of_nodes2>
< ... >>

<**command:** comand_line_parameter <command_line_parameter2> < ... >>

<**input:** input_file_name
<input_file_name2>
< ... >>

<**output:** output_file_name
<output_file_name2>
< ... >>

<**hints:** <replicateable | non-replicateable>
<duration: short | medium | long>>

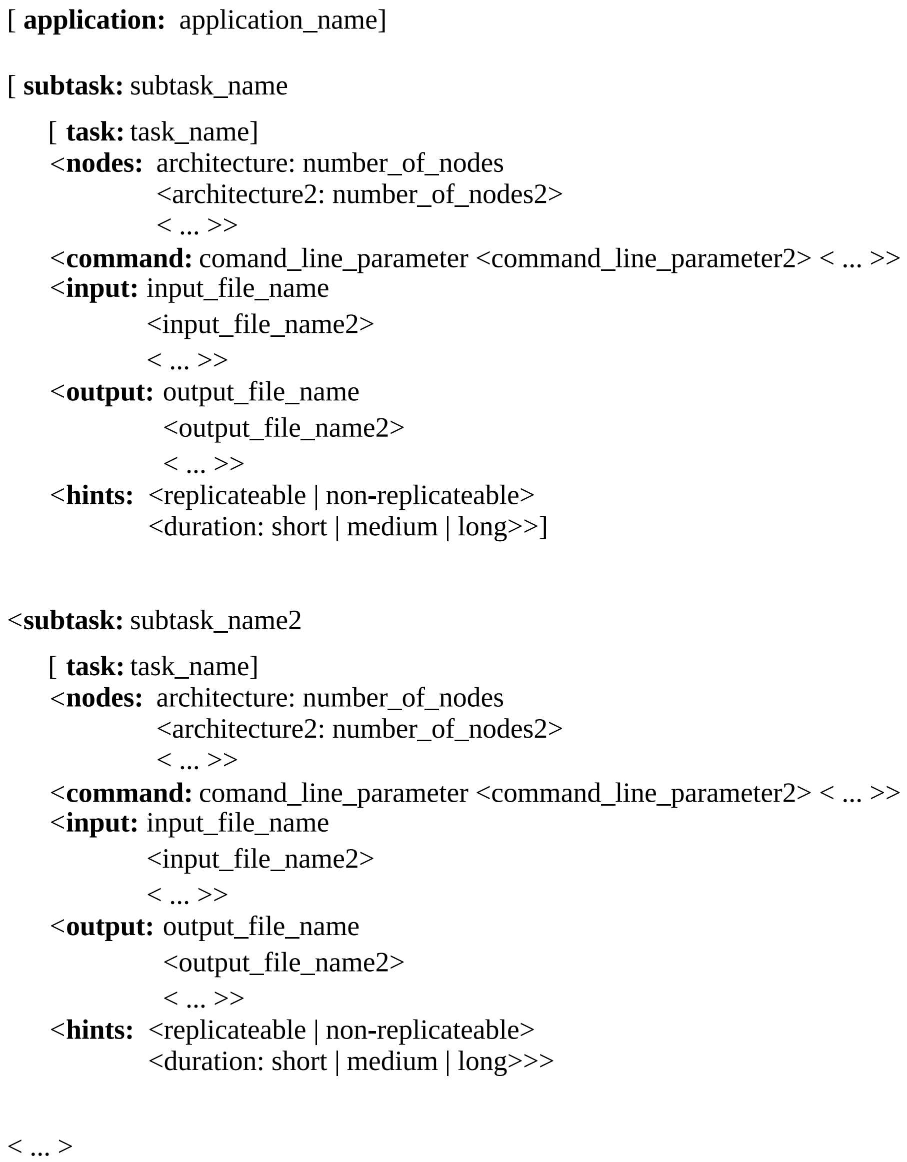
[ ] indicates required parameters
< > indicates optional parameters
bold face indicates keywords

Figure 9: VDCE Task Description Syntax

when an application is defined. Figure 7 shows how the task description syntax can be used to define a set of tasks which perform common linear algebra operations. These tasks can be used to construct a linear equation solver application, as shown in Figure 12.

5.2 Object-based Approach

Incorporating object oriented methodologies into the VDCE model will give VDCE users a natural way to express relationships between application components. Object oriented design emphasizes building software components with well defined and intuitive interfaces in order to facilitate component reuse. These are the same criteria for VDCE tasks, so the use of object oriented methodologies in the VDCE is a natural evolution of both technologies. The techniques used to dispatch large grain executable files will be the same whether those executables implement procedures or object methods. In fact, the earliest designs of the VDCE [88] described the use of proxies and interface design languages (IDLs) to implement an RPC based communication scheme. These same concepts, proxies and IDLs, are used to construct most distributed object systems [78]. Hence, object oriented applications can easily be accommodated in our conceptual model for developing VDCEs.



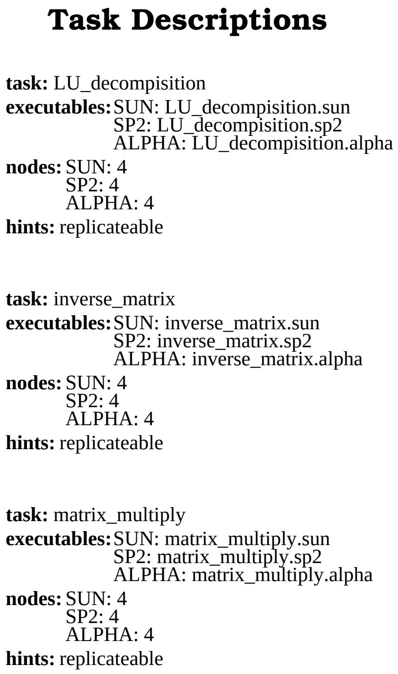
```
[ application:  application_name]

[ subtask: subtask_name
     [ task: task_name]
     <nodes:  architecture: number_of_nodes
              <architecture2: number_of_nodes2>
              < ... >>
     <command: comand_line_parameter <command_line_parameter2> < ... >>
     <input: input_file_name
             <input_file_name2>
             < ... >>
     <output: output_file_name
              <output_file_name2>
              < ... >>
     <hints:  <replicateable | non-replicateable>
              <duration: short | medium | long>>]


<subtask: subtask_name2
     [ task: task_name]
     <nodes:  architecture: number_of_nodes
              <architecture2: number_of_nodes2>
              < ... >>
     <command: comand_line_parameter <command_line_parameter2> < ... >>
     <input: input_file_name
             <input_file_name2>
             < ... >>
     <output: output_file_name
              <output_file_name2>
              < ... >>
     <hints:  <replicateable | non-replicateable>
              <duration: short | medium | long>>>


< ... >
```

Figure 10: VDCE Application Description Syntax



Task Descriptions

task: LU_decompisition
executables: SUN: LU_decompisition.sun
SP2: LU_decompisition.sp2
ALPHA: LU_decompisition.alpha
nodes: SUN: 4
SP2: 4
ALPHA: 4
hints: replicateable

task: inverse_matrix
executables: SUN: inverse_matrix.sun
SP2: inverse_matrix.sp2
ALPHA: inverse_matrix.alpha
nodes: SUN: 4
SP2: 4
ALPHA: 4
hints: replicateable

task: matrix_multiply
executables: SUN: matrix_multiply.sun
SP2: matrix_multiply.sp2
ALPHA: matrix_multiply.alpha
nodes: SUN: 4
SP2: 4
ALPHA: 4
hints: replicateable

Figure 11: Task Descriptions of Linear Algebra Operations

The object oriented extensions to the VDCE application description framework are straight forward and can be explained using the same linear equation solver example discussed previously. Now suppose that instead of using the procedurally oriented tasks of Figure 12 the user is considering the use of a CORBA compliant [80] matrix class. There are several methods to implement each class. One method involves *one program per method* [80]. The result of this approach would be similar to the procedural approach. For each task in the application the user would provide the VDCE with a series of executables which implement the task on the various hardware platforms. In the object oriented system the executables would implement class methods rather than conventional procedures. The user would have to alert the VDCE to this difference, therefore a new keyword, `object-implementation`, is added to the **hints** section of the text description.

The changes to the application description of the linear equation solver would be more significant than those to the task description. In the **input** and **output** fields of the procedural linear equation solver of Figure 12 the operands are treated as file names. In the object oriented application description, the **input** and **output** parameters are object references rather than file names.

In order to support the new functions described in the previous subsection, an object oriented communication system will need to be incorporated into the VDCE runtime manager. As noted above, a proxy based communication system was outlined in [88]. This approach

Application Description

subtask: LU_decompose
task: LU_decompisition
input: A.matrix
output: L.matrix , U.matrix
command: A.matrix L.matrix U.matrix

subtask: inverse_matrix_1
task: inverse_matrix
input: L.matrix
output: L^{-1} .matrix
command: L.matrix L^{-1} .matrix
synchronization: after LU_decompse completes

subtask: inverse_matrix_2
task: inverse_matrix
input: U.matrix
output: U^{-1}.matrix
command: U.matrix U^{-1}.matrix
synchronization: after LU_decompse completes

subtask: first_matrix_multiply
task: matrix_multiply
input: L^{-1} .matrix , B.vector
output: temp.vector
command: L.matrix B.vector temp.vector
synchronization: after inverse_matrix_1 completes

subtask: second_matrix_multiply
task: matrix_multiply
input: U^{-1}.matrix , temp.vector
output: X.vector
command: U.matrix temp.vector X.vector
synchronization: after first_matrix_multiply completes
after inverse_matrix_2 completes

Application Graph

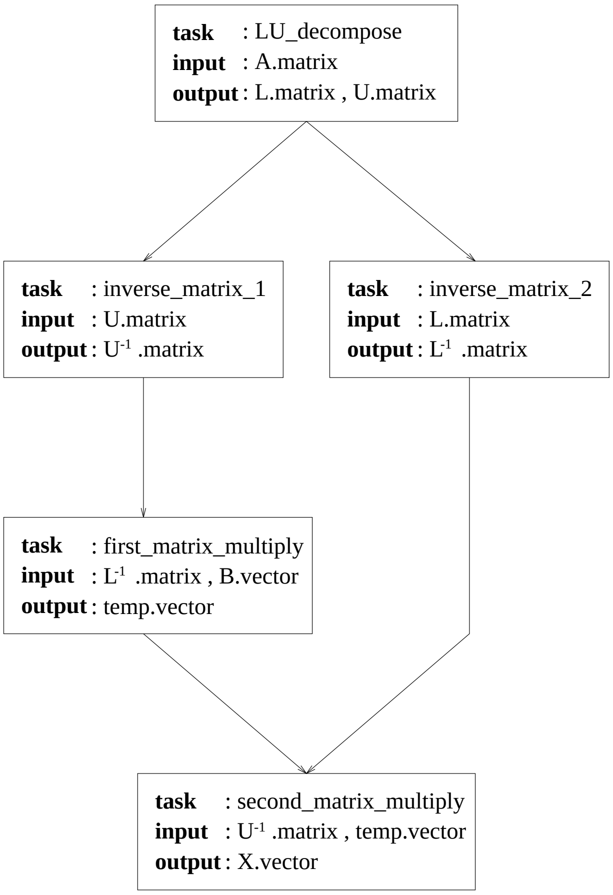


Figure 12: Application Description of a Linear Equation Solver

could be expanded to allow the proxies to provide CORBA compliant Object Adapter services [80] to object implementations. This would enable object implementations designed for other distributed object systems, such as IBM's SOM/DSOM [78], to be used in VDCE applications. The proxies would have to support the VDCE's redundant execution model and collaborate with VDCE execution processes to invoke and cancel object methods as needed.

An alternative approach would involve restructuring the VDCE runtime manager to run on top of, and exploit the capabilities of, a distributed object system. As an example, the VDCE execution process could be implemented as a DSOM client program which would interact with the DSOM Object Request Broker to invoke and manipulate object implementations across the network. The main advantage to this approach is that DSOM includes a *Replication Framework* which "allows an object to be mirrored in multiple address spaces" [78]. The Replication Framework might provide the redundancy support needed by the VDCE.

5.3 Web-based Approach

The goal of this activity is to investigate the use of Web-based technology to build Problem Solving Environment for C3I applications and National and Grand Challenges. We refer to this environment as WebWork and to build WebWork, we will use today's server technology and publicly available components of Web Productivity Tools. This approach has been adopted by NPAC WebTools to bootstrap the initial software process that will prototype Virtual Software Laboratory (VSL). We will then use VSL to prototype World-Wide Virtual Machine (WWVM), and then use both to prototype, test and demonstrate a set of C3I applications. We will initially employ the top-down process methodology which facilitates quick formation and publication of coarse grain modules using the existing C3I software components. Such modules are then linked in terms of dataflow channels to form compute-webs, accessible by clients. Visual compute-web editor will be prototyped in the early stage of the project and then used to publish both problems and solutions for the selected set of C3I applications. This will provide framework for testing our concept of the agents aided collaborative problem solving environment. We will start with the most intuitive WebFlow paradigm in which users directly employ visual editors to build compute-web connectivity graphs. A few application domains will be addressed concurrently and simple instances of problems and solutions will be identified by cross-cutting these disciplines in terms of emergent WebWork standards. In the next stage, we will extend this approach to other programming paradigms and we will address base HPCC support for intra-modular computation in terms of the High Performance Fortran Coordination Language (HPFCL) model. Here we will be also able to initiate and test the bottom–up process by importing HPCC runtime libraries to the WebWork framework.

Aspects of such a process, aimed at building consistent and complete module libraries for selected domains, will be addressed throughout the project and we will attempt to formalize it in the HPCC domain in terms of the WebScript protocol. Such an evolutionary common intermediate form would provide an integration platform for a growing collection of middleware protocols and a powerful, high level programming language based framework for publishing problems and solutions. With HPFCL, we will initiate the process of building the HPCC layer on top of pervasive Web technologies, testing it in the specific application context, prototyping parallel I/O in terms of server databases, and collecting specification requirements for the next generation multithreaded object-oriented compute-servers, driven by the WebWork paradigm.

Our approach to build Web-based VDCE prototype, we refer to it as WebWork, will be based on three layers. More detail about WebWork can be found in [79] (see also Appendix). WebWork represents a world-wide virtual machine (WWVM) model for heterogeneous and distributed high performance computation. Layer 1 is an extensible World-Wide Virtual Machine (WWVM) for heterogeneous and distributed high performance computing; it defines computation and communication primitives, initially based on existing Web standards (MIME, HTTP, CGI, CCI), to provide a publication model of computation. Layer 3 contains the user level programming and problem-solving environments (PSE's). These domain-specific PSE's range from NWChem and Global Arrays for computational chemistry at Pacific Northwest laboratory, ELLPACK (parallel numerics PSE), MATLAB (uniprocessor PSE), and Khoros (signal processing/visualization PSE) in the simulation arena, to complex National Challenges including education, manufacturing and healthcare. This minimal two-layer model supports only traditional forms of client-server interaction; that is, computation is performed by a web of servers, while client tools act as simple presenters for the published output of the computation. As WebWork grows, rather than building PSE's directly atop the WWVM, Layer 2, or middleware, will add more advanced client paradigms to this basis. In Layer 2, advanced client codes called agents may take on server functionality, becoming autonomous participants in the compute-web. Increases in client-code mobility and flexibility will also require the addition of "brokers" to mediate the interchange of different data formats and "little languages" (VRML for 3D graphics, PERL for text processing, HPFCL for coarse-grained control-parallel HPF modules, AVS for visualization and dataflow, ELLPACK/PDELAB for PDE specifications, and so forth) between participants in a compute-web. This interpolating environment, WebScript, will be an evolving collection of competing approaches with agent based "translators" to implement interoperability.

5.4 Discussion

The research results of these three on-going NPAC projects and their experimental results will enable the researchers involved in this project to successfully address all the research challenges of developing a conceptual model for virtual distributed computing environments. Furthermore, the availability of high performance computers at NPAC and at the Theorey Center at Cornell University that can be accessed via the NYNET testbed will provide us with all the required resources to develop and demonstrate the C3I applications on the NYNET VDCE as requested by the RFP document.

6 PROJECT TASKS AND MILESTONES

In this proposal, we have surveyed the current trends in computer hardware, computer software, network technology and application characteristics. The results of this survey showed the important role of these technologies to develop the virtual distributed computing environment. We also presented an approach for creating a prototype Virtual Distributed Computing Environment that can efficiently utilize multiple programming paradigms, programming languages, computer architectures, and physical machines for solving complex applications. These applications have complicated characteristics in computation, storage, visualization, heterogeneity, parallelism, communication, and synchronization, that exceed the capability of a single programming paradigm, programming language, computer architecture, and physical machine. In what follows, we outline the main tasks we shall perform to to develop a conceptual model to build and implement VDCE prototype.

6.1 Project Tasks

- **Task 1:** Survey and evaluate VDCE enabling technologies and identify shortfalls:
  - **Task 1.1:** Parallel and distributed programming software and hardware
  - **Task 1.2:** Software integration tools and environments
  - **Task 1.3:** Web-based software tools
  - **Task 1.4:** High speed networks and protocols
  - **Task 1.5:** Write a report that summarizes the results of this task survey and evaluation of VDCE enabling technologies.
- **Task 2:** Develop a VDCE conceptual development model. This model shall define the operational characteristics, data flow and structure of the VDCE components.
  - **Task 2.1:** Identify VDCE main software components required to develop VDCE applications
  - **Task 2.2:** Identify the main software components required to run VDCE applications
  - **Task 2.3:** Evaluate different schemes and technologies to implement the components identified in Task 2.2 and Task 2.3. This shall include the performance of a risk assessment for each of the candidate technologies.

  - **Task 2.4:** Conduct a cost/manpower assessment for the implementation of a fully functional VDCE.
  - **Task 2.5:** Write a report that summarizes the approach to be used for the VDCE conceptual development model.

- **Task 3:** Develop a C3I VDCE Application
  - **Task 3.1:** Identify a C3I application in consultation with Rome Laboratory personnel
  - **Task 3.2:** Extend the VDCE software development model to C3I applications. Analyze the utilization of the required technologies in an operational military environment and identify technical deficiencies. Develop a methodology to overcome these deficiencies.
  - **Task 3.3:** Identify required software tools to implement VDCE SDM for C3I applications
  - **Task 3.4:** Develop the selected C3I application using the software tools identified in Task 3.3
  - **Task 3.5:** Identify required software tools and algorithms to run C3I applications in VDCEs
  - **Task 3.6:** Implement and demonstrate the selected C3I application on the NYNET testbed. This will include a minimum of 3 nodes.
  - **Task 3.7:** Write a summary report describing our approach to develop and implement the selected C3I VDCE application.

- **Task 4:** Develop a methodology and implement a HDSM C3I VDCE Application
  - **Task 4.1:** Identify a HDSM C3I application in consultation with Rome Laboratory personnel
  - **Task 4.2:** Conduct a technical risk assessment for a HDSM implementation which includes data conversion, thread management, page size, uniform access, programming languages and interhost communication.
  - **Task 4.3:** Extend the VDCE Software Development Model to develop HDSM C3I Applications
  - **Task 4.4:** Identify Required Software Tools to develop HDSM C3I applications over VDCEs

  - **Task 4.5:** Develop the selected HDSM C3I application using the software tools identified in Task 4.4
  - **Task 4.6:** Identify Required software tools and algorithms to run HDSM C3I applications in VDCEs
  - **Task 4.7:** Implement and demonstrate the selected HDSM C3I application on the NYNET testbed. This will include a minimum of 3 nodes. This prototype HDSM implementation shall demonstrate the satifactory resolution of the high risk technical issues.
  - **Task 4.8:** Conduct a cost/manpower assessment for an implementation of an n-node HDSM.
  - **Task 4.9:** Write a summary report of our approach to develop and implement the selected HDSM C3I application.
- **Task 5:** Prepare Monthly Program Progress, CFSRs, Interim & Final Technical Reports and Present Results.

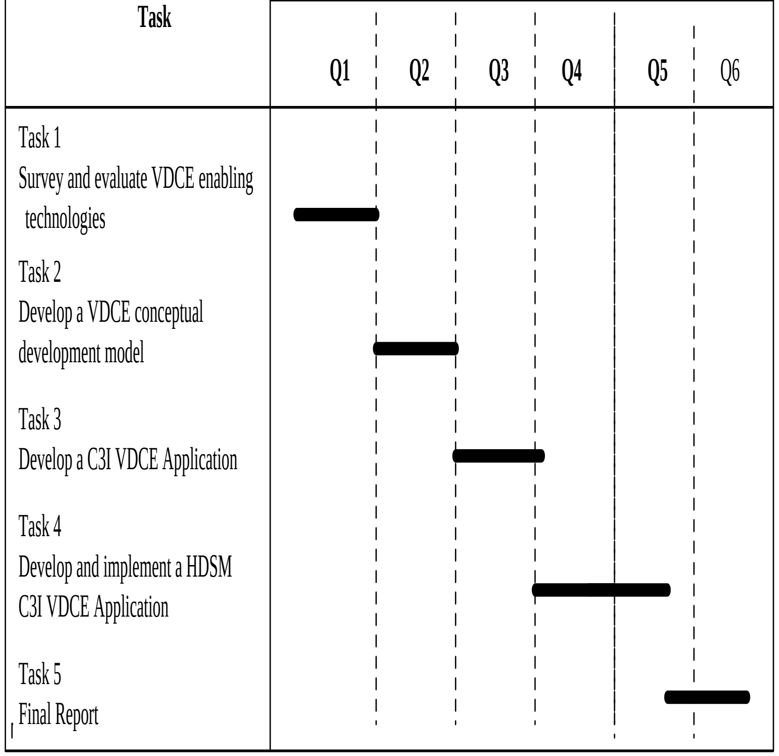
6.2 Task Milestones

Figure 13 shows the time schedule for performing the tasks discussed in the previous subsection.

7 DEMONSTRATIONS

- **Demonstration 1:** Demonstrate the tools that can be used to develop C3I applications over VDCEs
- **Demonstration 2:** Demonstrate the tools that can be used to run the selected C3I application over the NYNET testbed
- **Demonstration 3:** Demonstrate the tools required to develop and run the selected HDSM C3I application on the NYNET testbed

Figure 14 shows the time schedule for performing each of the above mentioned presentations.



Q1-Q5: 3 month periods

Q6: one month period

Figure 13: Main Task Table

Demonstration	Schedule					
	Q1	**Q2**	**Q3**	**Q4**	Q5	Q6
Demonstration 1 Tools to develop C3I applications over VDCEs		▬				
Demonstration 2 Tools to run the selected C3I application over the NYNET testbed			▬			
Demonstration 3 Tools to develop and run selected HDSM C3I application on the NYNET testbed				▬		
Final Demonstration						▬

Q1-Q5: represent 3 month period

Q6: represents one month period

Figure 14: Presentation Table

8 Program Organization

Professor Salim Hariri will serve as the principal investigator for the subject effort. Professor Hariri received a BSEE, with distinction, from Damascus University, Damascus, Syria, in 1977; an MSc from The Ohio State University, Columbus, Ohio, in 1982; and a Ph.D. in computer engineering from the University of Southern California in 1986. Currently, he is an Associate Professor in the Department of Electrical and Computer Engineering at Syracuse University. He has worked and consulted at AT &T Bell Labs in designing and evaluating reliable distributed computing environments between 1989-1992.

Professor Hariri's current research focuses on high performance distributed computing, high speed networks design, expert systems for managing performance of high speed networks, benchmarking and evaluating parallel and distributed software tools, software development methodology for heterogeneous high performance computing systems, He has co-authored over 100 journal and conference papers and he is the author of a book "High Performance Distributed Computing: Concepts and Design" to be published by Artech House Inc, in 1996. Professor Hariri was the project leader for the benchmarking project of parallel and distributed software tools for BMC3I applications and the virtual machine project sponsored by Rome Laboratory. Dr. Hariri is currently the project leader for the Rome Lab MultiMedia Transfer Protocols effort.

Professor Geoffrey Fox will be the co-principle investigator. He is currently a Professor of Computer Science and Physics, and Director of NPAC (Northeast Parallel Architecture Center) at Syracuse University. He was a Professor of Physics at Caltech from 1972-1990. He also held the positions of Executive Officer of Physics, Dean for Educational Computing and Associate Provost for Computing at Caltech. Prior to his tenure at Caltech, he has held various positions at Institute for Advanced Studies at Princeton, Lawrence Berkeley Laboratory, Cavendish Laboratory, Cambridge, England, Brookhaven National Laboratory, and Argonne National Laboratory.

Professor Fox has led a team which developed the initial use of hypercube in a large variety of scientific fields and has expanded study to other supercomputer architectures. His expertise is in concurrent algorithms and software. He has published over 250 research papers in journals and conferences in the areas of high energy physics, parallel computers, application of parallel computers for computational science. At Syracuse, he leads the ACTION program aimed at integrating parallel computers into industry. He is part of the NSF science and Technology Center CRPC (Center for Research in Parallel Computation led by Ken Kennedy). He has led the Syracuse group developing the FORTRAN 90D (High Performance FORTRAN) compiler. He is also the chair of the ARPA Parallel Compiler Runtime Consortium.

Professors Hariri and Fox will have the primary responsibility for providing leadership and direction in order to achieve the technical objectives of this program. Supporting Professors Hariri and Fox in this effort will be:

Dr. Wojtek Furmanski is a research professor in Physics and conducts research and development in computer and computation science at NPAC. He has 10 years of experience in high performance computing, starting from the early days of the Caltech Concurrent Computation Program. At Caltech, Furmanski also worked on neuroscience and computer vision applications. He prototyped the MOVIE (Multitasking Object-oriented Visual Interactive Environment) system. This has been further developed at NPAC as a core technology for the televirtuality operating environment. Current work in this area is focused on integrating MOVIE with the World Wide Web technologies. The first product in this area - WebTools - is in the process of being developed and packaged for public release.

Mr. Stephen Warzala will be the NPAC program manager for this effort. He currently serves as the primary management interface with Rome Laboratory personnel on NPAC's on going contract activities. He will be responsible for monitoring and maintaining the program schedule and assuring the delivery of quality products as required. Mr. Warzala brings 14 years of DOD Research and Development (R&D) oriented project experience to our team. He has served as a technical management consultant for several DOD contractors including Northrup, Grumman Data Systems, Texas Instruments and COMSAT. Mr. Warzala began his career as a computer scientist/project engineer at the Rome Air Development Center (RADC).

9 COMPUTATIONAL RESOURCES AT NPAC

NPAC computational resources offer a variety of most recent HPCC technologies. The systems installed at NPAC provide testbeds for a multitude of computationally intensive projects, and can be broadly divided into 4 categories:

- Parallel Computers
- Clusters of high-performance workstations
- Desktop workstations for program development and data visualization
- Networking infrastructure

9.1 Parallel Computers

The parallel machines currently installed in NPAC include a CM5, an iPSC 860, two DECmpp machines, an SP2, and an nCUBE 2. They represent the major current architectural trends for MPP platforms. The configuration and utilization data for these machines are as follows:

The CM5 is a 32 node parallel system from Thinking Machines Corporation. Every node has 32MB of memory, a SUN Sparc 1 scalar CPU, and four vector units. The nodes are connected by TMC's proprietary network with a quad-tree topology. The peak computational power of our CM5 is estimated at 3 GFLOPS. The CM5 is used in a number of ways at NPAC: it supports educational projects, serves as a compute server for a number of research projects, both internal and external, and for experiments in distributed computing. In the context of the current project, CM5 will be used to run weather simulation and computational electromagnetic codes.

The iPSC 860 from Intel is a 16 node parallel system with 256 MB of RAM total, and a hypercube interconnect topology. It features a small scale concurrent file system. The iPSC is used as the main resource for NPAC's High Performance FORTRAN and Parallel C++ development projects. It will be used for C++ code development for the current project.

The DECmpp machines is a large SIMD systems with 16K processors, operating in a data parallel model. DECmpp, manufactured by MasPar Corporation and distributed by Digital, is fully configured and is equipped with a high speed I/O channel and a HiPPI interface. Its architecture makes it well suited for regular problems. The aggregate peak computing power of the DECmpp is approximately 1.5 GFLOPS. The DECmpp platform has relatively sophisticated and user friendly software support. For the current project the machine will support signal and data compression applications.

The SP2 machine is the newest HPCC product of the IBM Corporation. The SP2 consists of a 12 very powerful nodes. The node processor of the SP2 is an IBM RISC/6000 Power2 processor running at 66 MHz, one of the most powerful RISC processors available. Available software includes Oracle Para llel Server and DB2 Parallel Edition RDBMS. For current project the SP2 will provide parallel database, parallel Web server and simulation services.

The nCUBE 2 installed at NPAC is a 64 node, 2 GB RAM machine with a MIMD hypercube architecture. The salient feature of the NPAC machine is its scalable parallel I/O subsystem. The subsystem, working in parallel with the computational processor array, consists of 32 nodes forming its own hypercube, with a SCSI controller and a disk array connected to every node. The current configuration supports 96 disks with total storage capacity of nearly 0.2 TB. The I/O subsystem supports also networking processors and controllers. Network hookup of the NPAC's nCUBE consists of multiple Ethernets multiplexed to the ATM backbone. nCUBE's design represents the state-of-art scalable I/O system, probably the most advanced one in the industry. The nCUBE platform in NPAC supports the Parallel Oracle Database Server, a high performance, high capacity relational database management system, and the Oracle Parallel Text Server. The Text Server provides high performance free text store and retrieval engine. In addition, the nCUBE supports an advanced Parallel File System used as a storage vehicle in the ongoing Video on Demand project in NPAC.

9.2 Clusters of high-performance workstations

The distributed computing facilities in NPAC include two clusters of high performance workstations: DEC Alpha cluster and a heterogeneous cluster of ATM connected workstations.

The DEC Alpha cluster consists of 8 Alpha model 400 compute servers. The cluster is supported by a high performance networking backbone consisting of dedicated, switched FDDI segments. This solution provides full FDDI bandwidth and low latency switching to every workstation in the cluster. The model implemented at NPAC exemplifies Digital's approach to distributed computing.

The ATM cluster consists of a number of Sun and SGI workstations. The cluster forms an interface between the NYNET and the rest on NPAC facility. The cluster is used to develop new multimedia networking protocols, video on demand and collaboratory applications and to support demonstrations.

9.3 Networking Infrastructure

All NPAC facilities are networked. The main networking FDDI backbone connects NPAC file and compute servers, the Alpha cluster, the ATM cluster (via an SGI Challenge network server as a gateway) and is interfaced to a 100+ farm of desktop workstations and personal computers via a DECnis router. NPAC connectivity to Internet is provided by a dedicated Ethernet segment directly connected to a router feeding the T3 Internet link.

As seen from the above review, NPAC's facilities represents a state of art HPCC center providing its partners with access to the newest technology. The unique capabilities of NPAC computational infrastructure offers a number of benefits: A number of different parallel architectures are gathered in one place. This allows for extensive experiments and for identification of the most suitable architecture for a particular project. In addition to diverse parallel systems, loosely coupled distributed systems are supported as well. This extends the overall center capability and allows for integration of the technologies required to solve extremely irregular problems. The integration capability is supported by the efficient networking backbone. This capability includes applications running on geographically remote computers by interfacing NPAC internal LAN to NYNET high speed network.

NYNET will also support applications that require very high bandwidth for data exchange between supercomputers. Both traditional "number crunching" and data processing platforms are available. This allows us to test distributed applications that have both intensive data processing and computational components. Both database applications (parallel database server) and I/O intensive applications (parallel I/O subsystems) are supported. NPAC's heterogeneous environment offers considerable computing power and an advanced development environment for the current project. Resources Provided Through NYNET

NYNET links NPAC facilities to those at Rome Laboratory and Cornell. Currently, this link is based on two OC3 lines. At NPAC, a dedicated ATM cluster is directly connected to the NYNET OC3 links. The ATM infrastructure in NPAC consists of the two FORE-200 switches with OC-3c single mode, OC3 multimode, and 140 Mbit/s TAXI interfaces, and of the LAX-20 LAN access switch. In addition, the GTE switch supports TAXI, DS3, T1, and HiPPI interfaces. Switching devices enable internetworking with almost any kind of a network. Sun and SGI workstations in the cluster have full video conferencing capability, and digital video capture, compression, and retrieval support. The cluster provides hardware platform for both commercial and public domain collaboratory packages.

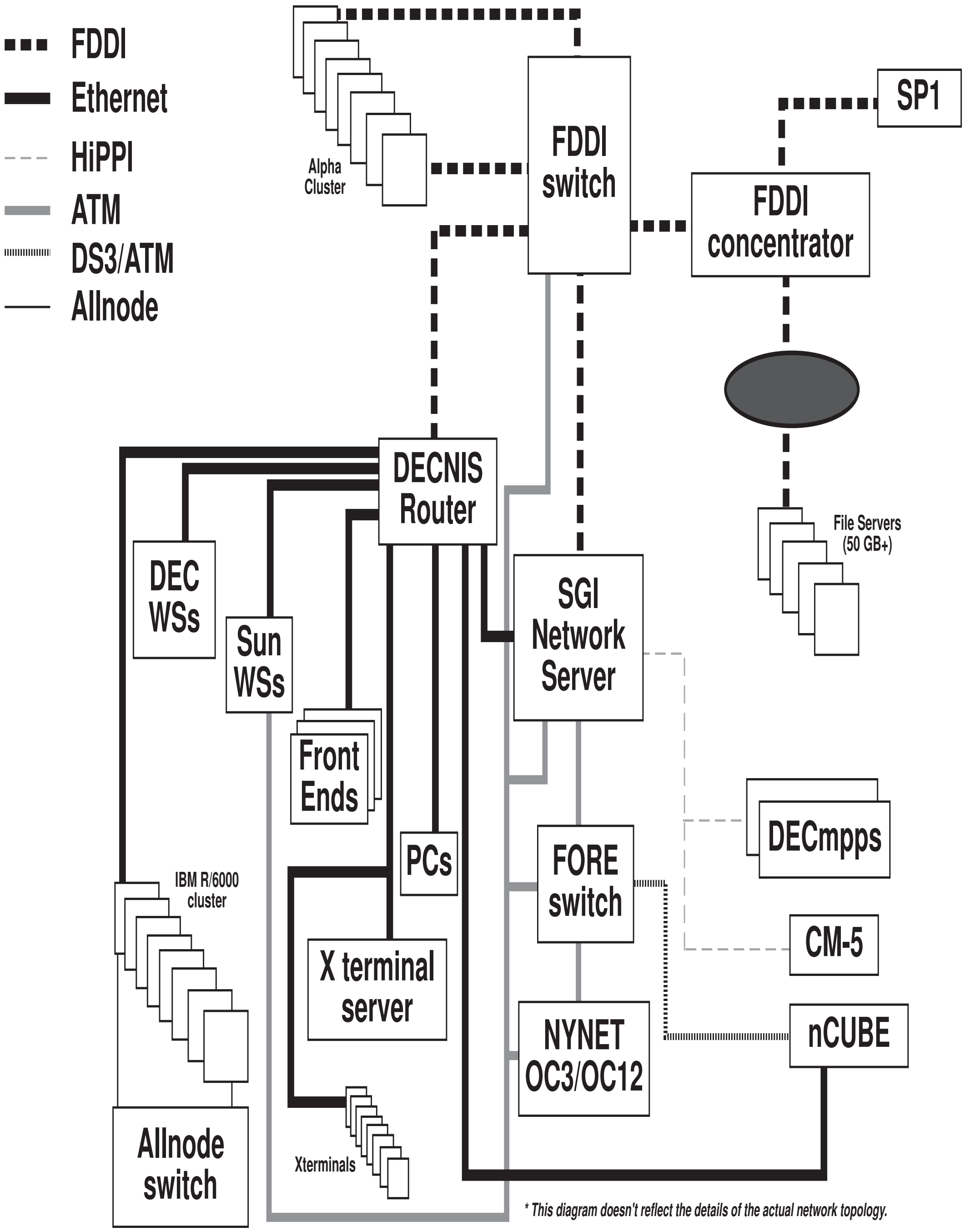
The backbone of the network is comprised of broadband ATM switches interconnected via SONET trunks. To provide unified network management and control capabilities across the public network, the switches are interfaced with the ATM Forum specified Public NNI (Node-to-Node Interface) and an OSI CMIP-based network management system is operated. The NYNET participants are provided access to the ATM backbone via 2 OC-3c SONET links per site. The purpose of providing two links per site is as follows:

The interface between the participant sites and the public network conforms to the ATM Forum specified UNI (User-to-Network Interface), and an SNMP-based customer network management system is being developed in collaboration with Rome Laboratory. This management system will allow the users to monitor and collect network configuration and traffic statistics, as well as providing limited end-to-end management and control capabilities. To interconnect the New York upstate participants to the downstate participants (different LATAs) requires a NYNET interface to a third party network. This is accomplished by means of the ATM Forum specified B-ICI (Broadband Inter-Carrier Interface). The same interface will be used to connect NYNET to the national computing environment network. The NYNET public network provides permanent virtual circuit (PVC) cell relay service (CRS). In the NYNET PVC CRS, virtual circuits are provisioned based upon the peak user burst rate, which may be spec! ified as large as the line access rate (OC-3 in this case).

NYNET connectivity is being extended. At present the network supports the Living Textbook projects with a number of schools directly connected to the OC3 ATM network. This network extension is in production mode and is being supported jointly by NPAC and Rome Laboratory. While not directly relevant to the current project, the Living Textbook extension testifies to the high-performance network expertise of NPAC team and to the NPAC abilities as system integrator. Living Textbook project is also an example of efficient and mutually beneficial cooperation of NPAC and Rome Laboratory staffs. With HPCC facilities being currently installed in Rome Laboratory (nCUBE, Paragon) we expect to promote the role of NYNET to a link supporting HPCC applications between two supercomputing sites. We also expect to extend the outreach of the technology developed in the current project by using NYNET as a vehicle to connect NPAC supercomputing facility to remote sites using Rome Laboratory connectivity to nation wide military ATM networks. Figure 9 shows a block diagram of the computing and networking resources available at NPAC.

9.4 CASE Distributed Multimedia Information System Laboratory

CASE Center at Syracuse University will be receiving a Shared University Research (SUR) Grant in 1995 from IBM to build the proposed distributed multimedia information system (DMIS) laboratory shown in Figure 10 based on IBM ATM technology. The DMIS laboratory will be connected to other Syracuse University Laboratories via ATM switches. The laboratory will have one IBM ATM switch (2220 Nways Switch) with two OC-3 ports where one of



** This diagram doesn't reflect the details of the actual network topology.*

Figure 15: NPAC computing and networking facilities.

them will be used to provide connectivity to the NYNET testbed, three ATM HUBS (8260-A) with several interfaces to other legacy networks (Ethernet and FDDI) and IBM PC cluster connected to the HUBS through 25 Mbps ATM adapters developed by IBM. The DMIS laboratory network management software includes the ATM Campus Manager and the Intelligent HUB manager based on NetView AIX system. The DMIS laboratory is expected to be fully operational by September of 1995.

-bb-error =

Figure 16: Distributed Multimedia Information Systems Laboratory.

References

[1] Ruben Prieto-Diaz, and James M. Neighbors, *Module Interconnection Languages,* The Journal of Systems and Software, No. 6, pp. 307-334, 1986.

[2] Craig Upson, Thomas Faulhaber, Jr., David Kamins, David LaidLaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam, *The Application Visualization System: A Computational Environment for Scientific Visualization,* IEEE Computer Graphics and Applications, pp. 30-42, 1989.

[3] Mario R. Barbacci, Charles B. Weinstock, Dennis L. Doubleday, Michael J. Gardner, and Randall W. Lichota, *Durra: a structure description language for developing distributed applications,* Software Engineering Journal, pp. 83-94, Mar 1993.

[4] Jeff Kramer, "Configuration Programming - A Framework for the Development of Distributable Systems", *IEEE*, pp. 374–384, 1990.

[5] James M. Neighbors, "The Darco Approach to Constructing Software from Reusable Components", *IEEE Trans. on Software Engineering*, vol. SE-10, pp. 564–574, September 1984.

[6] James C. Berets, Natasha Cherniack, and Richard M. Sands, "Intorduction to Cronus," BBN Systems and Technologies, Report No. 6986, January 1993.

[7] Roberto Bisiani and Alessandro Forin, "Architectural Support for Multilanguage Parallel Programming on Heterogeneous Systems", *ACM*, pp. 21–30, 1987.

[8] Massively Parallel System Group, *High Performance Fortran*, Digital Equipment Corporation, report ml01-5/u46 edition, 1992.

[9] Jeff Kramer, *Configuration Programming - A Framework for the Development of Distributable Systems,* Proc. IEEE Int. Conf. on Computer Systems and Software Engineering (CompEuro 90), pp. 374-384, 1990.

[10] *Express Reference Manual,* Parasoft Corporation, 1988.

[11] V. S. Sunderam, *PVM: A Framework for Parallel Distributed Computing*, Oak Ridge National Laboratory, 1991.

[12] Adam Beguelin, Jack J. Dongarra, G. A. Geist, Robert Manchek, V. S. Sunderam, *Graphical Development Tools for Network-Based Concurrent Supercomputing,* Proceedings of ACM Supercomputing 1991, pp. 435-444, 1991.

[13] Geoffrey C. Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, and Min-You Wu, *Fortran D Language Specifications*, Northeast Parallel Architectures Center, Syracuse University, 1990.

[14] K. Birman, R. Cooper, T. Joseph, K. Kane, and F. Schmuck, *The ISIS System Manual.*

[15] S. Ahuja, N. Crriero, and D. Gelernter, *IEEE Computer*, No.8, Vol.19, pp. 26-34, Aug 1986.

[16] Andrew S. Grimshaw, "An Introduction to Parallel Object-Oriented Programming with Mentat", *TR-91-07, Department of Computer Science, University of Virginia*, 1991.

[17] Andrew S. Grimshaw, *An Introduction to Parallel Object-Oriented Programming with Mentat* TR-91-07, Department of Computer Science, University of Virginia, 1991.

[18] Patrick T. Homer, and Richard D. Schlichting, *Supporting Heterogeneity and Distribution in the Numerical Propulsion System Simulation Project,* Proc. of the Second Int. Symposium on High Performance Distributed Computing, pp. 187-195, Jul 20-23, 1993.

[19] *Acquisition of a Highly Parallel Research Computer,* Argonne National Laboratory, Feb 1993.

[20] Richard L. Phillips, *Distributed Visualization at Los Alamos National Laboratory,* IEEE Computer, pp. 70-77, Aug 1989.

[21] Richard L. Phillips, *Distributed Visualization at Los Alamos National Laboratory,* IEEE Computer, pp. 70-77, Aug 1989.

[22] Brian Cabral, and Carol L. Hunter, *Visualization Tools at Lawrence Livermore National Laboratory,* IEEE Computer, Aug 1989, pp. 77-84.

[23] Brian Cabral, and Carol L. Hunter, *Visualization Tools at Lawrence Livermore National Laboratoty,* IEEE Computer, Aug 1989, pp. 77-84.

[24] Robert B. Haber, *Scientific Visualization and the Rivers Project at the National Center for Supercomputing Applications,* IEEE Computer, Aug 1989, pp. 84-89.

[25] Ira Pramanick, *Distributed Computing Solutions to the All-Pairs Shortest Path Problem,* Proc. of the Second Int. Symposium on High Performance Distributed Computing, Jul 20-23, 1993, pp. 196-203.

[26] T. F. La Porta and M. Schwartz, *Architectures, Features, and Implementation of High-Speed Transport Protocols,* IEEE Network Magazine, May 1991, pp. 14-22

[27] A. N. Netravali, W. D. Roome, and K. Sabnani, *Design and Implementation of a High-Speed Transport Protocol,* IEEE Trans. on Communications, November 1990.

[28] N. Jain, M. Schwartz and T. R. Bashkow, *Transport Protocol Processing at GBPS Rates,* Proceedings of the SIGCOMM Symposium on Communications Architecture and Protocols, Aug 1990, pp. 188-198.

[29] R. H. Perrott, *Parallel Language Developments in Europe: An Overview,* Concurrency: Practice and Experience, Vol. 4(8), Dec 1992, pp. 589-617.

[30] Doreen Y. Cheng, *A Survey of Parallel Programming Languages and Tools,* RND-93-005, NAS Ames Research Center, Mar 1993.

[31] Charles W. Krueger, *Software Reuse,* ACM Computing Surveys, Vol. 24, No.2, Jun 1992, pp. 131-182.

[32] Geoffery C. Fox, *The Architecture of Problems and Portable Parallel Software Systems,* Northeast Parallel Architectures Center, Syracuse University.

[33] Angel L. DeCegama, *Parallel Processing Architectures and VLSI Hardware, Vol.1*, Prentice-Hall, Inc., 1989.

[34] Kai Hwang and Faye A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, Inc., 1984.

[35] H. Siegel and L. Siegel, "PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition," *IEEE Transactions on Computers*, vol. 30, pp. 934–947, December 1981.

[36] Kai Hwang, "Advanced Parallel Processing with Supercomputer Architectures," *Proceedings of the IEEE*, vol. 75, pp. 1348–1379, October 1987.

[37] Doreen Y. Cheng, "A Survey of Parallel Programming Languages and Tools," *RND-93-005, NAS Ames Research Center*, March 1993.

[38] R. H. Perrott, "Parallel Language Developments in Europe: An Overview," *Concurrency: Practice and Experience*, vol. 4(8), pp. 589–617, December 1992.

[39] Patrick T. Homer and Richard D. Schlichting, "Supporting Heterogeneity and Distribution in the Numerical Propulsion System Simulation Project", *Proc. of the Second Int. Symposium on High Performance Distributed Computing*, pp. 187–195, July 20-23 1993.

[40] Charles W. Krueger, "Software Reuse," *ACM Computing Surveys*, vol. 24, pp. 131–182, June 1992.

[41] Ruben Prieto-Diaz and James M. Neighbors, "Module Interconnection Languages," *The Journal of Systems and Software*, vol. 6, pp. 307–334, 1986.

[42] Craig Upson, Thomas Faulhaber Jr., David Kamins, David LaidLaw, David Schlegel, Jeffrey Vroom, Robert Gurwitz, and Andries van Dam, "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, pp. 30–42, 1989.

[43] M. R. Barbacci, C. B. Weinstock, and J. M. Wing, "Durra: Language Support for Large-Grained Parallelism," *Parallel Processing and Applications, ed. E. Chiricozzi and A. Damico, North-Holland*, pp. 371–379, 1988.

[44] Jeff Kramer, "Configuration Programming - A Framework for the Development of Distributable Systems," *IEEE*, pp. 374–384, 1990.

[45] James M. Neighbors, "The Darco Approach to Constructing Software from Reusable Components," *IEEE Trans. on Software Engineering*, vol. SE-10, pp. 564–574, September 1984.

[46] Roberto Bisiani and Alessandro Forin, "Architectural Support for Multilanguage Parallel Programming on Heterogeneous Systems," *ACM*, pp. 21–30, 1987.

[47] Parasoft Corporation, "Express Reference Manual," 1988.

[48] V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing," 1991.

[49] Adam Beguelin, Jack J. Dongarra, G. A. Geist, Robert Manchek, and V. S. Sunderam, "Graphical Development Tools for Network-Based Concurrent Supercomputing," *Proceedings of ACM Supercomputing*, pp. 435–444, 1991.

[50] Geoffrey C. Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, , and Min-You Wu, "Fortran D Language Specifications," *Northeast Parallel Architectures Center, Syracuse University*, 1990.

[51] K. Birman, R. Cooper, T. Joseph, K. Kane, and F. Schmuck, "The ISIS System Manual".

[52] S. Ahuja, N. Crriero, and D. Gelernter, "Linda with Friends," *IEEE Computer*, vol. 19, pp. 26–34, August 1986.

[53] Andrew S. Grimshaw, "An Introduction to Parallel Object-Oriented Programming with Mentat," *TR-91-07, Department of Computer Science, University of Virginia*, 1991.

[54] Patrick T. Homer and Richard D. Schlichting, "Supporting Heterogeneity and Distribution in the Numerical Propulsion System Simulation Project," *Proc. of the Second Int. Symposium on High Performance Distributed Computing*, pp. 187–195, July 20-23 1993.

[55] Argonne National Laboratory, "Acquisition of a Highly Parallel Research Computer," February 1993.

[56] Brian Cabral and Carol L. Hunter, "Visualization Tools at Lawrence Livermore National Laboratory", *IEEE Computer*, pp. 77–84, August 1989.

[57] Richard L. Phillips, "Distributed Visualization at Los Alamos National Laboratory," *IEEE Computer*, pp. 70–77, August 1989.

[58] Robert B. Haber, "Scientific Visualization and the Rivers Project at the National Center for Supercomputing Applications," *IEEE Computer*, pp. 84–89, August 1989.

[59] T. F. La Porta and M. Schwartz, "Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, pp. 14–22, May 1991.

[60] A. N. Netravali, W. D. Roome, and K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol," *IEEE Trans. on Communications*, November 1990.

[61] N. Jain, M. Schwartz, and T. R. Bashkow, "Transport Protocol Processing at GBPS Rates," *Proceedings of the SIGCOMM Symposium on Communications Architecture and Protocols*, pp. 188–198, August 1990.

[62] Victor L. Peterson, John Kim, Terry L. Holst, George S. Deiwert, David M. Cooper, Andwer B. Watson, and F. Ron Bailey, "Supercomputer Requirements for Selected Disciplines Important to Aerospace," *Proceedings of the IEEE*, vol. 77, pp. 1038–1055, July 1989.

[63] Henry Fuchs, Marc Levoy, and Stephen M. Pizer, "Interactive Visualization of 3D Medical Data," *IEEE Computer*, pp. 46–51, August 1989.

[64] Charles C. Weems, "Architectural Requirements of Image Understanding with Respect to Parallel Processing," *Proceedings of the IEEE*, vol. 79, pp. 537–547, April 1991.

[65] Charles Weems, Edward Riseman, and Allen Hanson, "The DARPA Image Understanding Benchmark for Parallel Computers," *Journal of Parallel and Distributed Computing*, vol. 11, pp. 1–24, 1991.

[66] Kai Hwang, *Advanced Computer Architecture with Parallel Programming*, McGraw-Hill, Inc., 1993.

[67] Geoffrey C. Fox, "The Architecture of Problems and Portable Parallel Software Systems," *Northeast Parallel Architectures Center, Syracuse University*, 1990.

[68] Hwang, K., and Briggs, F.A., "Computer Architecture and Parallel Processing", McGraw-Hill, Second Edition.

[69] Geoffery C. Fox, *The Architecture of Problems and Portable Parallel Software Systems,* Northeast Parallel Architectures Center, Syracuse University.

[70] Geoffery C. Fox, *Software and Hardware Requirements for Some Applications of Parallel Computing to Industrial Problems,* NPAC Technical Report SCCS-717, June 1995, Syracuse University.

[71] Geoffrey C. Fox, Seema Hiranandani, Ken Kennedy, Charles Koelbel, Uli Kremer, Chau-Wen Tseng, and Min-You Wu, *Fortran D Language Specifications*, Northeast Parallel Architectures Center, Syracuse University, 1990.

[72] Cheng, D.Y., *A Survey of Parallel Programming Languages and Tools*, RND-93-005, NASA Ames Research Center, March 1993.

[73] Hariri, S., Park, J., Yu, F., Parashar, M., and Fox, G., *A Message Passing Interface for Parallel and Distributed Computing*, Proceedings of the Second International Symposium on High Performance Distributed Computing, pp. 84-91, July 1993.

[74] Walker, D., Standards for message passing in a distributed memory environment. *ORNL/TM-12147*, August 1992.

[75] Ira Pramanick, "Distributed Computing Solutions to the All-Pairs Shortest Path Problem," *Proceedings of the Second Int. Symposium on High Performance Distributed Computing*, pp. 196–203, July 20-23 1993.

[76] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors, Vol. 1*, Prentice-Hall, Inc., 1988.

[77] Rousselle, P., Tymann, P., Hariri, S., and Fox, G., *The Virtual Computing Environment,* Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing, pp. 7-14, August 1994.

[78] Bertz, M. *Interoperable Objects*, Dr. Dobb's Journal, Vol. 19, No. 11, pp. 18-32, October 1994.

[79] Geoffrey C. Fox, and Wojtek Furmanski, "The Use of the National Information Infrastructure and High Performance Computers in Industry," Northeast Parallel Architecture Center Technical Report, Syracuse, NY, August 1995.

[80] The Object Management Group, *The Common Object Request Broker: Architecture and Specification*, OMG Document Number 91.12.1, December 1991.

[81] Krueger, P., and Cawla, R., *The Stealth Distributed Scheduler,* Proceedings of the 11th IEEE International Conference on Distributed Computing Systems, May 1991.

[82] Clark, H. and McMillin, B., *DAWGS a distributed compute server utilizing idle workstations,* Journal of Parallel and Distributed Computing, pp. 175-186, Vol.14, No.2, 1992.

[83] Ju, J., Xu, G., and Tao, J., *Parallel Computing Using Idle Workstations,* Operating Systems Review, Vol. 27, No. 3, 1993.

[84] Litzkow, M. and Solomon, M., *Supporting checkpointing and process migration outside the UNIX kernel,* Proceedings of the Winter 1992 USENIX Conference 1991, pp. 283-290, 1991.

[85] Waldspurger, C.A., Hogg, T., Huberman, B.A., Kephart, J.O., and Stornetta, W.S., *Spawn: a distributed computational economy,* IEEE Transactions on Software Engineering, pp. 103-117, Vol.18, No.2, 1992.

[86] Kramer, J., *Configuration Programming - A Framework for the Development of Distributable Systems,* The Proceedings of the IEEE International Conference on Computer Systems and Software Engineering (CompEuro 90), pp. 374-384, 1990.

[87] Theimer, M., and Hayes, B., *Heterogeneous Process Migration by Recompilation*, Proceedings of the 11th IEEE International Conference on Distributed Computing Systems, May 1991.

[88] Rousselle, P., Tymann, P., Hariri, S., and Fox, G., *The Virtual Computing Environment,* Proceedings of the Third IEEE International Symposium on High Performance Distributed Computing, pp. 7-14, August 1994.

A APPENDIX

A.1 Resumes

A.2 Selected Technical Papers