Chapter 3

Overview of the VCE Architecture

The VCE is a suite of tools and technologies which support the development and execution of large distributed applications on networked high performance computing platforms [91]. This chapter will present a high level description of the VCE components and discuss their interfaces. The VCE design is modular and thus new tools and methodologies can be incorporated into the design over time. Some individual components of the system may have considerable value to users apart from their role in the larger VCE scheme. The modular design of the VCE allows subsets of its functionality to be deployed if other portions are not implemented or not needed in a particular situation.

3.1 VCE Building Blocks

The VCE design can be represented hierarchically by the components (building blocks) shown in Figure 14. At the highest level, the components of the VCE are divided into a *software development suite* and an *execution suite*. Application development and execution are largely independent activities except for areas such as problem determination and performance monitoring. The VCE design seeks to minimize the dependencies between the tools in the software development and execution suites. In the future new hardware architectures and development methodologies will

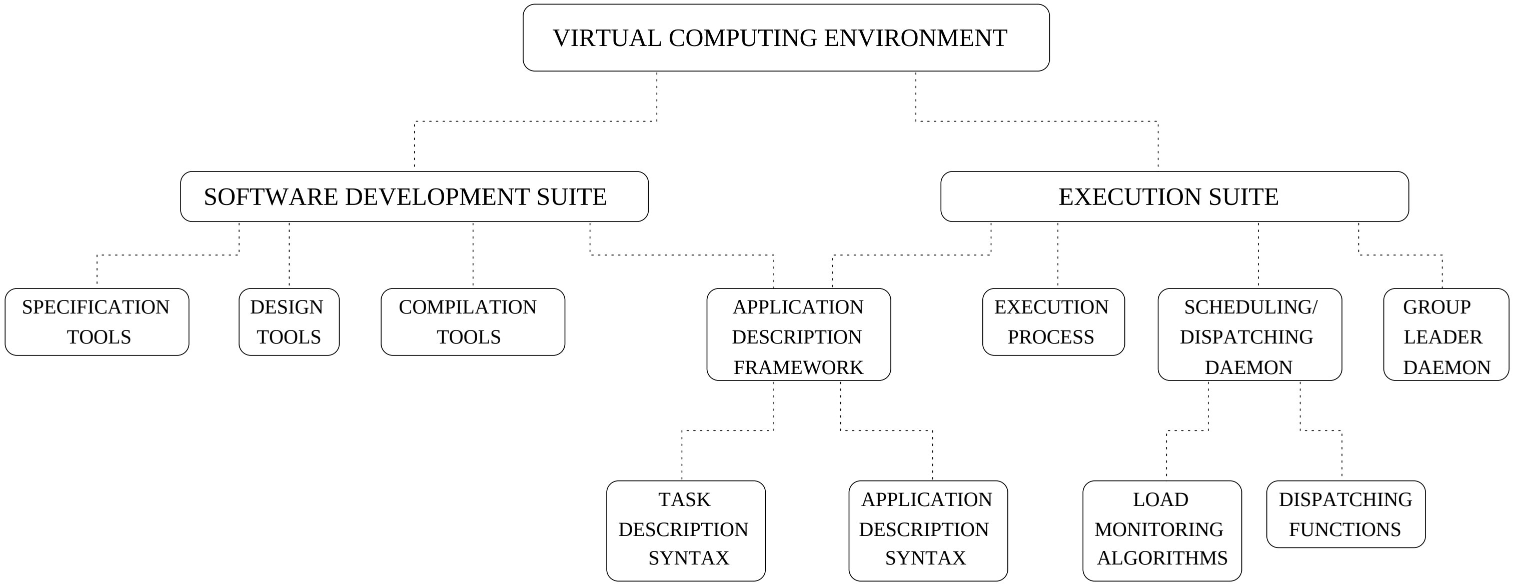

Figure 14: Structure of the VCE

inevitably arise. By constructing a general, extensible interface between the VCE development and execution environments, support for new software engineering tools and hardware platforms can be provided quickly with minimal effort and user disruption.

Figure 14 shows a software development suite that contains *specification tools*, *design tools* and *compilation tools*. This particular set of tools reflects a stepwise refinement approach to procedurally oriented application development. There are currently many mature procedurally oriented development tools (such as AVS [2]) which can easily be adopted to fill these roles in the VCE. These procedural tools can be replaced or augmented by object oriented tools which support object class development or class hierarchy development. Design tools specifically geared towards implementing high performance applications across multiple World Wide Web attached high performance computers [45] can be accommodated in the VCE software development suite as well. Section 3.2 will describe a VCE software development suite which uses an AVS-like graphical user interface. Chapters 4 and 5 discuss how object oriented and Web based functions can be supported by the VCE execution suite. The design of the software development suite is flexible and can be implemented using procedural, object oriented or Web techniques.

The *VCE Application Description Framework* is the primary interface between the VCE development environment and the execution environment. The tools of the software development suite produce application and task description files (Figure 2). These description files, along with executable and data files they refer to, are input to the execution suite. The description files, which use the syntax of the application description framework (Chapter 4), contain all the information the execution suite needs to dispatch the application's executables and manage its data files and intertask communications. Figures 2 and 14 both convey that the definition of the application description framework is an integral part of both the software development suite and the execution suite. The characteristics of an application are captured by the software development suite and communicated to the execution suite using the application description framework syntax.

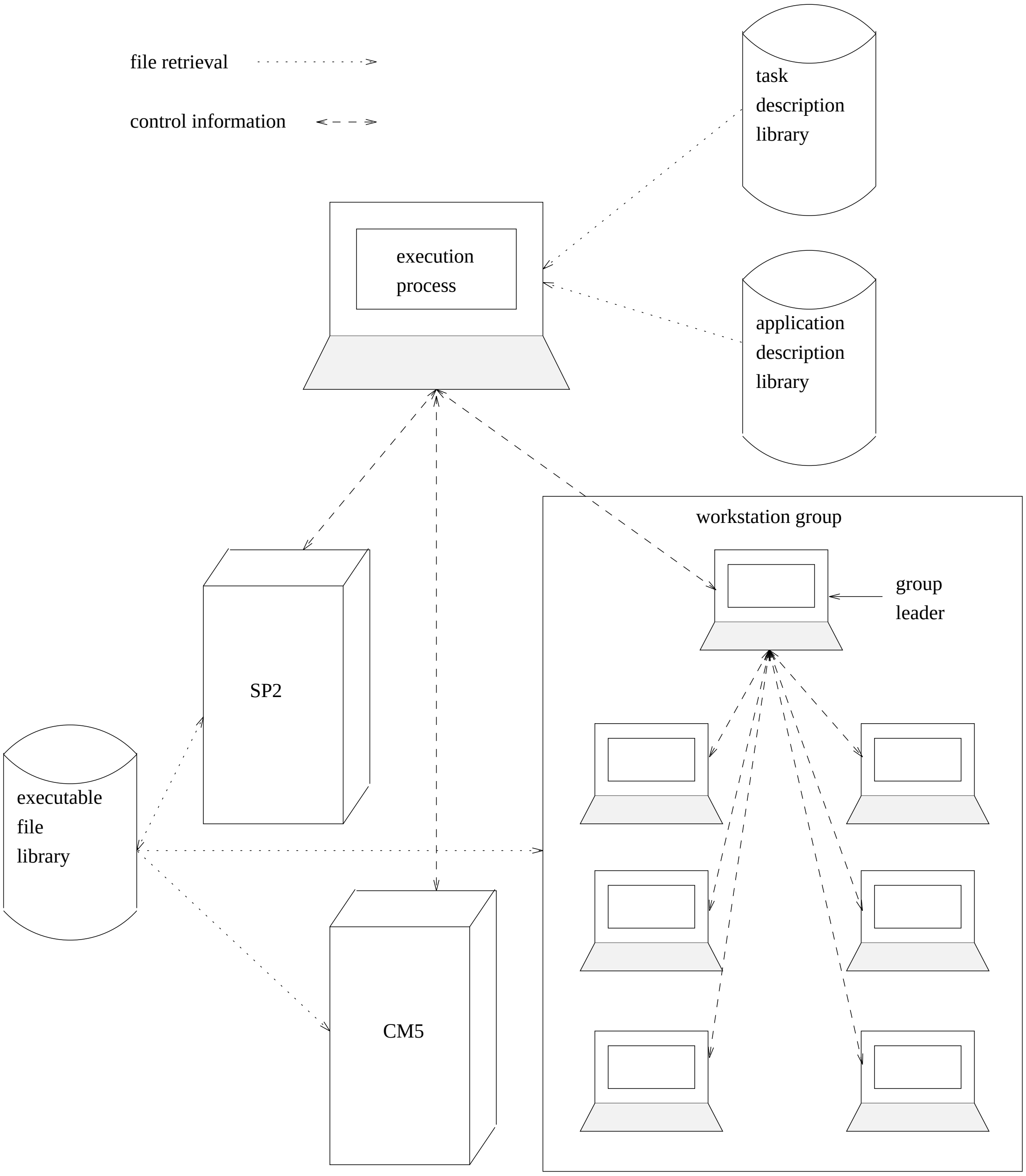

Figure 15: The VCE Execution Environment

The execution suite components shown in Figure 14 represent the components of the VCE's unique automated distributed application control system. Research on managing the execution of distributed applications over multiple high performance computers (as opposed to multiple workstations) is a new but active area of inquiry. The VCE execution suite is an early attempt to provide such capability. Figure 15 shows how control information flows in the VCE execution suite. Task and application descriptions are only used by the execution process. It makes task placement decisions based on load information provided by individual machines and group leaders. Task implementations are loaded by individual machines under the control of the execution process.

The function of the execution process is to manage the execution of a single application. From the information in task and application description files it determines the precedence constraints between the application tasks. When a task is ready to be dispatched, the execution process solicits load information from group leaders whose groups contain machines which are capable of running the task (based on the architecture of the machines, the availability of executable files, and the placement of data files and communicating tasks). The scheduling and dispatching algorithms used by execution processes are outlined in Section 5.1.

The implementation of an execution process is straight forward. It reads an application description and builds data structures that indicate the synchronization and communication requirements of each task. The application description will indicate that some tasks cannot be run until others have completed (i.e. a precedence constraint exists). When all such constraints are satisfied the execution process determines (from a task description) what executables are available for the task. It then contacts eligible group leaders to determine how many machines are available to run the task. Finally, it sends dispatching commands to the scheduling/dispatching daemons on the available machines. Once all eligible tasks are dispatched the execution process waits for notification from a scheduling/dispatching daemon that a task instance has completed. When such a notification arrives, the execution process cancels any redundant instances of the task and checks to see if the completion satisfies the precedence constraint of any other task or completes the application. The execution process also copies data files needed or produced by task instances and instantiates intertask communication facilities as needed (Section 5.2).

The execution process is the most complex element of the execution suite. While the execution process runs on the workstation of the application user, group leaders and scheduling/dispatching daemons run on widely dispersed host machines. They must not place a significant load on these machines. A scheduling/dispatching daemon periodically samples the load on its host, notifies its group leader if the load changes significantly, and executes dispatching instructions sent by execution processes. Group leaders maintain a list of lightly loaded group members and furnish it to execution processes upon request.

Modularity is a goal of the execution suite as well as the software development suite. Among execution suite components, only execution processes are effected by changes to the application description framework. Refinements to the scheduling/dispatching daemon's load monitoring algorithms, which determine when a machine is eligible to accept VCE work, can be made without impacting other components.

3.2 The Software Development Suite

The decomposition of the software development suite shown in Figure 14 can be achieved by adapting a modular application builder [103] such as AVS [2] for use in the VCE.

AVS is a tool which is used to construct scientific applications from vendor supplied or user written modules [73]. AVS provides a set of data types and functions for receiving or transmitting data between modules. Well designed AVS modules accept an input data type, perform some operation on it and produce output data (such modules are often called filters).

Applications are constructed with AVS using a graphical user interface. The user can select modules from a menu, place them on a palette and draw connections between them to indicate how the output of one module is to be supplied as input to others. This forms a dataflow pipeline. In most AVS applications the last module in such a pipeline renders the data using AVS supplied graphic functions.

AVS provides some support for distributed systems. Specifically, in a workstation cluster environment the user can indicate that different modules are to be placed on different machines within the cluster. However, no load balancing capabilities are supplied.

AVS support for supercomputing platforms is also rather limited [103]. The most common way to enable a parallel computer to participate in an AVS pipeline is to write a special module that runs on the user's workstation and uses sockets connections to communicate with a remote supercomputer and run a program there. Recent releases of AVS have included some facilities to allow AVS modules to be implemented on some types of parallel machines (such as the CM5). Because the number of parallel architectures supported is small, and only a limited number of AVS data types can be used on them, AVS is still not suitable for constructing applications which involve processing of large data sets on multiple, geographically dispersed supercomputers [103].

The goal of the VCE software development suite is to provide a software engineering environment which is as intuitive and usable as AVS but which serves as a front end to a much more robust execution environment. AVS's emphasis on visual user interaction, modular application development, function encapsulation and program reuse are retained in the VCE. In fact, it is likely that AVS will be used to prototype the VCE software development suite. AVS captures the application specification supplied by the user in a data structure called an *AVS network* [2]. Implementing an AVS based front end to the VCE software development suite simply entails constructing a translation program to convert the AVS network syntax into VCE application description framework syntax. Similar translators can be used to include other modular application builders, or development tools based on other paradigms, into the software development suite.

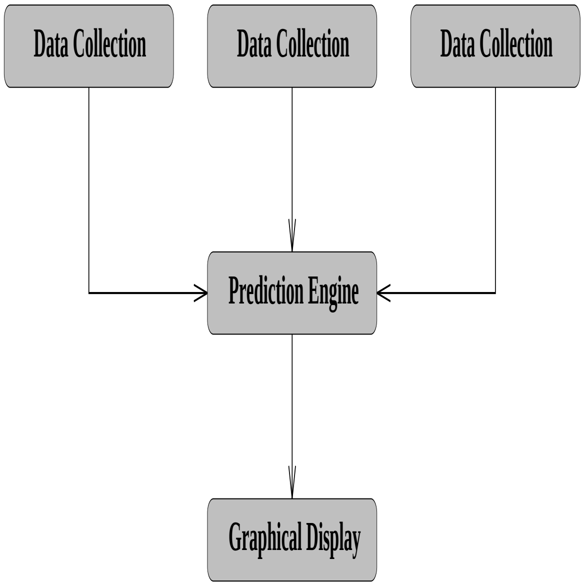

Figure 16: Task Graph of a Meteorological Application

3.2.1 Example Application Development Activities

To demonstrate the capabilities of the software development suite an example is presented showing how a meteorological application which processes observation data to predict storm activity is developed using one particular development approach.

The application is represented in the software development suite as a *task graph*. Problem specification tools are used to formalize the functional flow of the application and create the initial task graph. Figure 16 shows the preliminary task graph.

Once the initial task graph is complete, design tools are used to identify the temporal (time or synchronization) characteristics of the tasks within the application. For this example the we consider design tools based on Fox's [43] architecture of problems and portable parallel software. The information supplied at this stage helps the execution suite to select suitable hardware architectures to host each task. Figure 17 shows the task graph of the weather forecasting application after the design stage analysis is complete.

Finally, in the coding stage of development, the user supplies an architecture

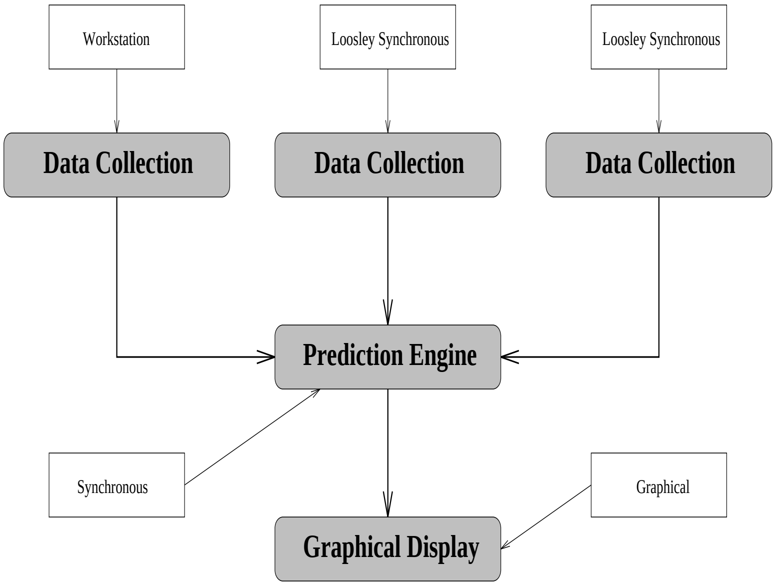

Figure 17: Annotated Design Stage Task Graph

independent source module to implement each task. Tools based on emerging standards such as High Performance Fortran [46] and High Performance C++ [29] will be included among the software design suite's compilation tools to express intratask parallelism. Support for communication between tasks will be provided by the VCE communication system which is discussed in Sections 3.6 and 5.2. Specifying the coding phase information for the weather forecasting application results in the task graph shown in Figure 18.

The completed task graph contains the input, output, and synchronization requirements of each task and its implementation language. As the task graph is annotated in the various stages of the software development suite, the user is able to record information about the nature of the application that may be useful to the execution suite. These *hints* will allow the execution module to do additional 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 task can be given higher priority and opportunities for parallelism will be maximized.

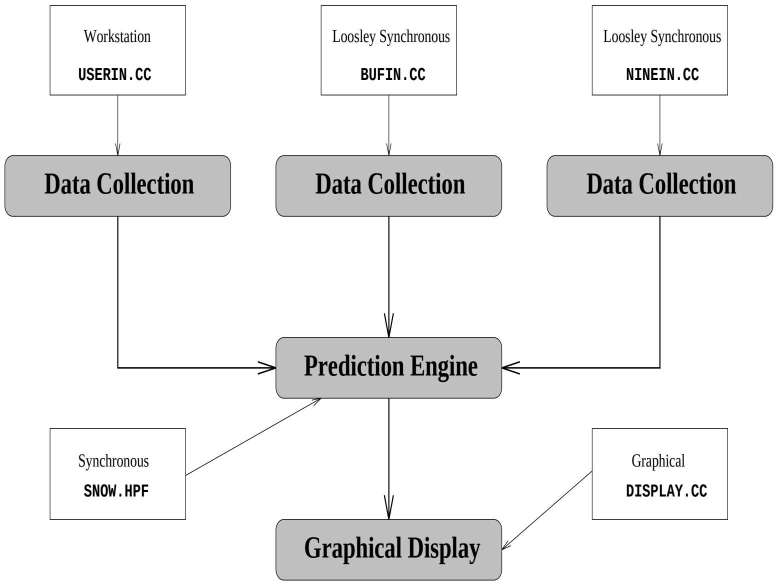

Figure 18: Coding Level Task Graph

3.3 The Application Description Framework

The application description framework provides a textual methodology for encoding all information about a distributed application that the VCE execution suite will need to efficiently dispatch it and manage its communications. The framework is described in detail in Chapter 4.

Since compilers are a component of the software development suite, executable files are one of its output products. A single program produced by the software development suite may be compiled on several different hardware platforms, resulting in a set of executable files. Similarly, a problem might be attacked using different algorithms on different machines, resulting in a set of programs that yield functionally equivalent executable files for a variety of platforms. Managing these programs and executable files is the responsibility of the tools in the software development suite. However, the execution suite needs to be aware that a set of executables can be used interchangeably when dispatching an application. Therefore, the software development module constructs a *task description file* which describes such a set of executable files. The task description file includes an indication of which executable file is to be used on which platform. It also indicates which platforms are preferable to others and documents the type of input data consumed by the task and the type of output data produced. Other dispatching hints can be supplied by the user and encoded in the task description file.

Each task constitutes one application component available to developers. As described in the previous section, a user can employ an AVS-like tool to construct an application by selecting tasks and graphically indicating their communication and synchronization requirements. The user can also use the application construction tool to specify the source of application input and the destination of its output. The result of this activity is an *application description file* which captures the structure of the application.

The application description framework specifies the syntax for task and application description files which, along with executable files, are produced by the software development suite and used by the execution suite to realize applications.

3.3.1 Supporting Object Oriented and Other Development Methods

The specification, design and development tools described in Section 3.2 are procedurally oriented. That is because most current AVS-like modular application builder tools, which are the motivation for the current software development suite design, are procedurally or dataflow oriented [103]. However, the application description framework contains capabilities to manage object oriented applications as well. Tasks can be specified as providing method implementations and an application description can lay out the flow of data and control within a class hierarchy. In the future, as visual tools for designing and developing object oriented applications become more common, they can be integrated into the software development suite. The application description framework and the execution suite already have capabilities to describe

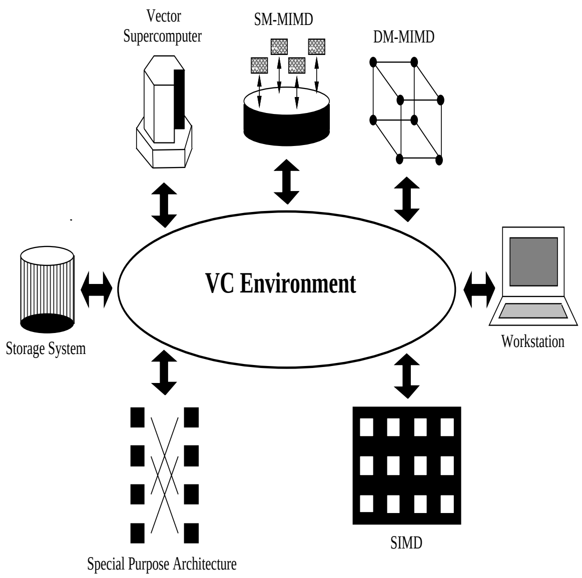

Figure 19: The Virtual Computing Environment

and execute object oriented applications, so the addition of an object oriented development environment will be straight forward. Also, the syntax of the application description framework is simple, intuitive and human-readable. This will give users the option of preparing task and application description files manually if an application is very simple or if it is being developed using a methodology (such as object oriented) which is not supported by a visual programming environment.

3.4 The Execution Suite

The VCE execution suite coordinates execution of user applications on a collection of heterogeneous network attached computers as illustrated in Figure 19. Participating computers may be widely geographically separated. All computers used by the VCE are shared with conventional users. The VCE seeks to use only excess capacity on participating machines and minimize any possible impact to non-VCE users.

Most of the user's interaction with the VCE will be with the software development suite. Once a user is satisfied that an application is correctly specified and implemented he or she will invoke the execution suite to run it. Therefore, the requirements of the execution suite can be stated succinctly. They are to take an application, which is defined by the information contained in a fully specified application description (as shown in Figure 18) and the modules referenced by the description, and manage the execution of the tasks so as to allow the efficient realization of the application in a heterogeneous distributed environment exemplified by Figure 19.

Some of the functions which must be performed by the execution suite include:

- Managing the intertask communications of VCE applications.
- Locating hosts which are architecturally suitable for application tasks and dispatching the work.
- Monitoring the load on participating machines to prevent the scheduling of VCE work on machines that are already busy.

While all of these activities are discussed in detail in subsequent chapters, the remainder of this section provides an overview of the techniques used in the VCE for these functions.

3.4.1 Components of the Execution Suite

Machines with a wide variety of capabilities participate in the Virtual Computing Environment. A *VCE daemon* running on each machine contributes to global scheduling and remote execution functions. The machines participating in the VCE are organized as groups, where the members of a group share similar architectural features. In a particular configuration there might be an Alpha workstation group, a Sun workstation group, and an SP2 parallel computer group. One machine within each group acts as group leader and coordinates the activities of the group.

The group concept lends considerable flexibility to the VCE and will be discussed in more detail in Section 5.1. Individual machines send load information to the group

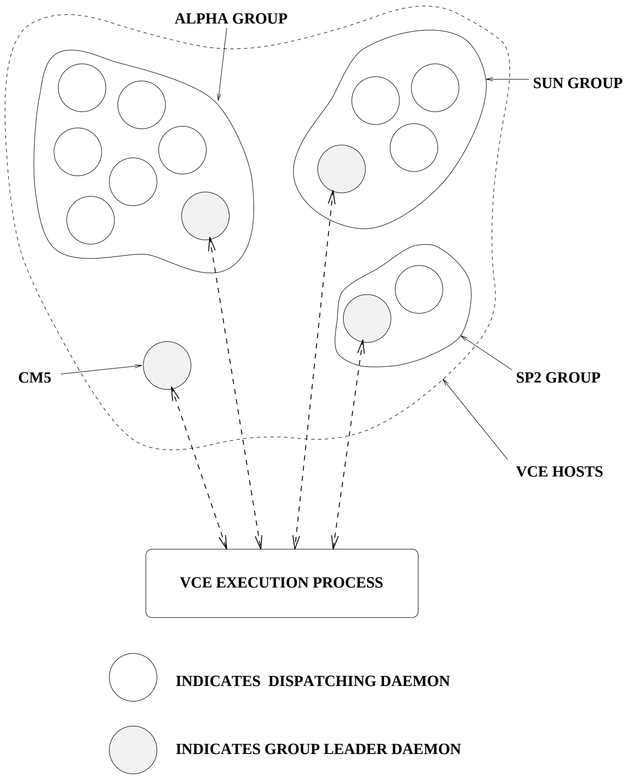

Figure 20: Organization of Machines in the VCE

leader. In current prototypes this information is simply a message indicating that the load on the machine has changed and now is or is not low enough that the machine can host VCE tasks. The group leader is therefore a repository of load information for the entire group and can be consulted when placement decisions need to be made rather than communicating with all the group members.

Execution of a VCE application utilizes an *execution process* running on the user's workstation. The execution process uses information produced by the software development suite. The application and task description files are parsed to determine the types of machines required to run the application. The execution process then sends request messages to leaders of groups that are candidates to host portions of the application (see Figure 20). Groups are chosen based on the availability of executable files and the preferences and hints supplied by the user. Each group leader returns a list of lightly loaded group members. The execution process then sends task dispatching instructions to the machines identified by the group leader. On some types of machines VCE work is dispatched at low priority to minimize interference with non-VCE processing. Internet and World Wide Web technology is used to allow the execution process to locate group leaders.

3.5 Load Balancing

Much of the literature on load balancing is not useful in designing a load balancing methodology for a system like the VCE. Load balancing algorithms are often validated on homogeneous systems using tasks that are easily migrated (like parallel Monte Carlo simulations [104] or big batch jobs [21]). Load balancing algorithms are not tested on large networks composed of multiple supercomputers.

To provide the most robust possible execution environment and provide the greatest possible flexibility, the execution suite may implement a variety of process migration schemes. The following list outlines four process migration approaches that may be useful the VCE.

- **Process migration through redundant execution.** Dispatch the same task on several idle machines. If one of those machines gets busy with other work the other instances of the redundant task will still perform well.

- **Process migration through checkpointing.** Migratable jobs checkpoint regularly. To migrate a job kill it and start it somewhere else by instantiating the new instance from the checkpoint record. This is expensive and may require the cooperation of the task involved (the task may need to be coded to write and read checkpoint data [63]).

- **Process migration the old-fashioned way.** To migrate a job dump the contents of the address space, copy it to a new machine and restart it. This has many drawbacks, one being that it requires homogeneity [71].

- **Process migration through recompilation.** This is very expensive but may be very robust. It is likely to be difficult to implement [102].

Ideally, The VCE execution suite would include implementations of all these algorithms. When a migration is needed, the most efficient technique would be chosen. For instance, if a task was to be migrated across compatible hosts, dumping and migrating its execution image would be feasible. For migrations between heterogeneous machines, checkpointing or recompilation methods would be needed.

So far, VCE research has focused on the use of redundant execution as the most general load balancing technology. It permits the use of heterogeneous hosts and does not require any special processing on the part of the task being run. Redundant execution does require special support from the run time system. Communication among redundant instances of application tasks must be managed carefully to insure that the redundancy is transparent to the application developer and communication coherence is maintained even if some task instances progress considerably faster than others. Details of how communication is handled in the face of redundancy is discussed in the next section and in Section 5.2.

A central contribution of the current work is to demonstrate the value of redundant execution as a load balancing tool in high performance computing environments. Techniques for scheduling redundant execution are outlined in Section 5.1. In the absence of redundant scheduling or the more complex strategies mentioned above, the only method for scheduling task executions involves *selective scheduling*. That is, placing tasks on a single lightly loaded host. Chapter 6 will describe a series of prototype and simulation experiments which were performed to gain insight into the characteristics of both redundant and selective scheduling. These experiments showed that relatively low levels of redundancy (three instances of each task) dramatically improved performance and were sufficient to overcome the performance penalty of low priority dispatching.

3.6 Communication

The VCE communication system is discussed in detail in Section 5.2. Many distributed computing environments use some variation of a proxy based system (Figure 21) for communication among distributed application components [59] [89]. That is, if two portions of an application are to run on different machines, the distributed computing system supplies proxy processes which present the same interface as the remote component but actually communicate with their counterparts to allow remote communication using the semantics of local communication. This isolates the distributed application from the complexities of network communication and greatly simplifies distributed application development.

In the VCE these communication proxy functions are performed by *communication processes*. Once placement decisions are made by the execution process, it starts a communication to support each task instance. The communication process runs on the same machine as the task it supports.

The communication pattern of the application is described in the application description file and is conveyed to the communication process by the execution process. This enables the communication process to know the types and destinations of the

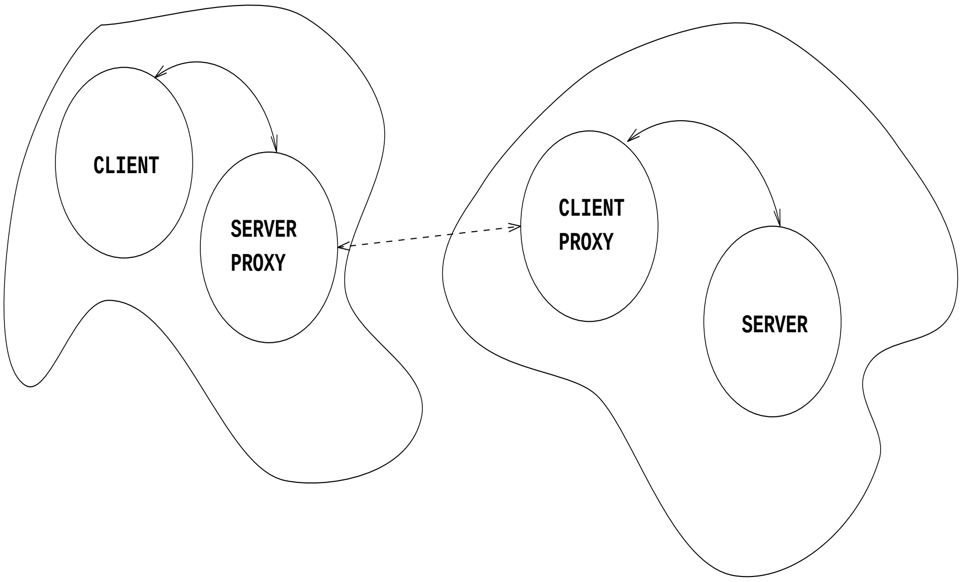

Figure 21: Communication via Proxies

data it will be responsible for. The communication process insulates the user task from the management of redundancy by automatically duplicating output data for all instances of a replicated destination task and discarding redundant input received from replicated source tasks.

3.7 Design Attributes

The VCE design is more robust than related systems described in Chapter 2. The software design suite allows a user to design an application from existing components without requiring any special knowledge of the programming languages, algorithms or hardware they employ. All these issues are addressed by task implementers, who similarly need not be knowledgeable in the particulars of the applications in which the tasks will be used. This enables scientists and engineers to exploit high performance computing facilities more easily than systems which focus on parallel programming language extensions (Sections 2.7 and 2.9). The modularity of the software development suite will permit the use of procedural, object oriented or other (perhaps as yet undiscovered) software engineering technologies.

The VCE's execution suite can coordinate application tasks executing simultaneously on multiple high performance computers and heterogeneous workstations. This will accommodate the execution of Grand Challenge applications which require multiple heterogeneous high performance computers to acheive their best performance [74]. Previous remote execution systems were often limited to coordinating application execution across multiple workstations (Section 2.6). The VCE retains the ability of these systems to detect idle hosts and use the resources of these machines. These benefits are obtained without operating system modifications.

Bibliography

[1] Adler, R. M., *Emerging Standards for Component Software*, Computer, Vol. 28, No. 3, pp. 68-76, March 1995.

[2] Advanced Visual Systems Inc., **AVS Developer's Guide (Release 4.0)**, Part No. 320-0013-02, Revision B, Advanced Visual Systems Inc., Waltham Mass., May 1992.

[3] Ahmad, I., Ghagoor, A., and Mehrotra, K., *A Decentralized Task Scheduling Algorithm and its Performance Modeling for Computer Networks*, Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, pp. 314-321, December 1991.

[4] Ahmad, I., Ghagoor, A., Mehrotra, K., Mohan, C. K., and Ranka, S., *Performance Modeling of Load Balancing Models Using Neural Networks*, Concurrency: Practice and Experience, Vol. 6, No. 5, pp. 393-409, August 1994.

[5] Allia, M., Arlat, J., Fabre, J.-C., Kanoun, K., and Powell, D., *Dependability Evaluation of the Saturation Technique*, Proceedings of the 5th Annual European Computer Conference (CompEuro '91), pp. 437-441, May 1991.

[6] Anderson, T. E., Culler, D. E., and Patterson, D. A., *A Case for NOW (Networks of Workstations)*, IEEE Micro, Vol 15, No. 1, pp. 54-64, February 1995.

[7] Armstrong, R., and Macfarlane, J., *The Use of Frameworks for Scientific Computation in a Parallel Distributed Environment*, Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco, California, pp. 15-25, August 1994.

[8] Bach, M. J., **The Design of the Unix Operating System**, Prentice-Hall Inc., 1986.

[9] Barak, A., Guday, S.,and Wheeler, R. G., **The MOSIX Distributed Operating System, Load Balancing for UNIX**, Lecture Notes in Computer Science, Springer-Verlag, Vol. 672, 1993.

[10] Barak, A., Laden, O., and Yarom, Y., *The NOW MOSIX and its Preemptive Process Migration Scheme*, Bulletin of the Technical Committee on Operating Systems and Application Environments, IEEE Computer Society, Vol. 7, No. 2, pp. 5-11, June 1995.

[11] Barak, A., and Wheeler, R. G., *MOSIX: An Integrated Multiprocessor UNIX*, Proceedings of the Winter 1989 USENIX Conference, San Diego, CA, pp. 101-112, February 1989.

[12] Barbacci, M. R., Doubleday, D. L., Weinstock, C. B., and Wing, J. M., *The Durra Language and Runtime Environment: Tools for PMS Level Programming,* **Decentralized Systems**, Proceedings of the IFIP WG 10.3 Working Conference, pp. 25-27, North Holland, 1990.

[13] Barbacci, M. R., Weinstock, C. B., Doubleday, D. L., Gardner, M. J., and Lichota, R. W., *Durra: a structure description language for developing distributed applications,* Software Engineering Journal, pp. 83-94, March 1993.

[14] Barbacci, M. R., and Wing, J. M., *Specifying Functional and Timing Behavior for Real-Time Applications*, Proceedings of Parallel Architectures and Languages Europe (PARLE) Volume 2. Springer-Verlag, Lecture Notes in Computer Science, Vol 259, pp. 124-140, June 1987.

[15] Benmohammed-Mahieddine, K., Dew, P. M., and Kara, M., *A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems*, Proceedings of the 14th International Conference on Distributed Computing Systems, pp. 484-492, June 1994.

[16] Besana, V., De Luca, J., Giard, E., and Reger, J., **Experiences with SOMobjects: Distributed System Object Model (DSOM)**, IBM International Technical Support Centers, Document No. GG24-4165-00, November 1993.

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

[18] Birman, K. P., Schiper, A., and Stephenson, *Lightweight Causal and Atomic Group Multicast*, ACM Transactions on Computer Systems, Vol 9, No 3. pp. 272-299, August 1991.

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

[20] Blumofe, R. D., and Park, D. S., *Scheduling Large-Scale Parallel Computations on Networks of Workstations*, Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco CA., pp. 96-105, August 1994.

[21] Bricker, A., Litzkow, M. and Livny, M., *Condor Technical Summary*, Computer Science Department, University of Wisconsin - Madison, January 1992.

[22] Butler, R., and Lusk, E., *User's Guide to the p4 Programming System*, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439-4801, April 1994.

[23] Carriero, N., and Gelernter, D., *Linda in Context*, Communications of the ACM, Vol. 32, No. 4, pp. 444-459, April 1989.

[24] Carriero, N., Gelernter, D., Kaminsky, D., Westbrook, J., *Adaptive Parallelism with Piranha*, Department of Computer Science, Yale University, New Haven, Connecticut, YALE/DCS/RR-954, February 1993.

[25] Carriero, N., Gelernter, D., Mattson, T. G., and Sherman, A. H., *The Linda Alternative to Message-Passing Systems*, Parallel Computing, Vol. 20, No. 4, April 1994.

[26] Casavant, T. L., and Kuhl, J. G., *A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems*, IEEE Transactions on Software Engineering, Vol. 14, No. 11, pp. 141-154, November 1988.

[27] Chang, J., and Maxemchuck, N. F., *Reliable Broadcast Protocols*, ACM Transactions on Computer Systems, Vol. 2, No. 3, pp. 251-273, August 1984.

[28] Chen, J., Xie, L., and Sun, Z.-X., *An Intelligent Network Manager for Distributed Task Scheduling*, Proceedings of the 11th International Conference on Computer Communication, pp. 493-499, October 1992.

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

[30] Cheriton, D.R., *The V Distributed System,* Communications of the ACM, pp. 314-333, Vol. 31, No. 3, 1988.

[31] Chiu, J-F, and Chiu, G-M, *Process-Replication for Fault-Tolerance and Performance Improvement in Distributed Computing Systems*, Proceedings of the Third International Symposium on High Performance Distributed Computing, San Francisco, California, pp. 236-243, August 1994.

[32] Chu, W. W., Holloway, L. J., Lan, M., and Efe, K., *Task Allocation in Distributed Data Processing*, Computer, Vol 13, No. 11, pp. 57-69, November 1980.

[33] 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.

[34] Dandamudi, S., *Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems*, Proceedings of the 15th International Conference on Distributed Computing Systems, pp. 484-492, June 1995.

[35] Deremer, F., and Kron, H. H., *Programming-in-the-Large Versus Programming-in-the-Small*, IEEE Transactions on Software Engineering, Vol. 2, No. 2, June 1974.

[36] Eager, D. L., Lazowska, E. D., and Zahorjan, J., *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing*, Performance Evaluation, Vol. 6, pp. 53-68, March 1986.

[37] Eager, D. L., Lazowska, E. D., and Zahorjan, J., *Adaptive Load Sharing in Homogeneous Distributed Systems*, IEEE Transactions on Software Engineering, Vol. 12, No. 5, pp. 662-675, May 1986.

[38] Elnozahy, E. N., and Zwaenepoel, W., *Replicated Distributed Processes in Manetho*, Proceedings of the Conference on Fault-Tolerant Computing Systems, pp 222-230, 1991.

[39] Efe, K., and Schaar, M. A., *Performance of Co-Scheduling on a Network of Workstations*, Proceedings of the 13th International Conference on Distributed Computing Systems, pp. 525-531, May 1993.

[40] Fabre, J. C., *Improving Fault Tolerance in Distributed Systems: The Saturation Approach*, Proceedings of the 5th Annual European Computer Conference (CompEuro '91), pp. 634-641, May 1991.

[41] Farrell, W. E., Fox, W. Given, J. W., MacRitchie, B., and Wang, J., *Data Processing on Networks of Workstations*, Bulletin of the Technical Committee

on Operating Systems and Application Environments, IEEE Computer Society, Vol. 7, No. 2, pp. 35-42, June 1995.

[42] Fox, G. C., et. al., **Solving Problems on Concurrent Processors, Volume 1,** Prentice Hall, 1988.

[43] Fox, G. C., **The Architecture of Problems and Portable Parallel Software Systems,** Northeast Parallel Architectures Center, Technical Report SCCS-134, Syracuse University, July 1991.

[44] Fox, G. C., and Furmanski, W., *The Use of National Information Infrastructure and High Performance Computers in Industry*, Proceedings of the 2nd International Workshop on Massively Parallel Processing Using Optical Interconnections, October 1995.

[45] Fox, G. C., Furmanski, W., Chen, M., Rebbi, C., and Cowie, J. H., *WebWorks: Integrated Programming Environment Tools for National and Grand Challenges*, Northeast Parallel Architectures Center, Technical Report sccs-0715, June 1995.

[46] Fox, G. C., Hiranandani, H., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C, and Wu, M., **Fortran D Language Specifications**, Northeast Parallel Architectures Center, Syracuse University, 1990.

[47] Garcia-Molina, H., and Spauster, A., *Ordered and Reliable Multicast Communication*, ACM Transactions on Computer Systems, Vol. 9, No. 3, pp. 242-271, August 1991.

[48] Geist, A., Beguelin, A., Dongarra, J., Weicheng, J., Manchek, R., and Sunderam, V., **PVM 3 User's Guide and Reference Manual**, Oak Ridge National Laboratory, Document No. ORNL/TM-12187, September 1994.

[49] Gelernter, D., and Carriero, C., *Coordination Languages and their Significance*, Communications of the ACM, Vol. 35, No. 2, pp. 97-107, February 1992.

[50] Ghafoor, A., and Ahmad, I., *An Efficient Model of Dynamic Task Scheduling for Distributed Systems*, Proceedings of the 14th Annual International Computer Software and Applications Conference, pp. 442-447, November 1990.

[51] Graham, I. S., **The HTML Sourcebook**, John Wiley & Sons Inc., 1995.

[52] Grimshaw, A. S., *The Mentat Runtime System: Support for Medium-Grain Parallel Computation*, Proceedings of the Fifth Distributed Memory Computing Conference, Los Alamitos, CA., Vol II, pp. 1064-1073, 1990.

[53] Grimshaw, A. S., *Easy-to-Use Object-Oriented Parallel Processing with Mentat*, Computer, Vol. 26, No. 5, May 1993.

[54] Hariri, S., *High Performance Distributed Systems: Concepts and Designs*, Artech House, in preparation.

[55] 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.

[56] Hariri, S., Yadav, R., Thiagarajan, B., Park, S. Y., Subramanyan, M., and Reddy, R., "A Concurrent Multi Target Tracker: Benchmarking and Portability", International Conference on Parallel Processing (ICPP), August 1994.

[57] Heidelberger, P., and Lavenberg, S. S., *Computer Performance Evaluation Methodology*, IEEE Transactions on Computers, Vol. 33, No. 12, pp. 1195-1220, December 1984.

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

[59] Jacob, B., **SOMobjects: A Pratical Introduction to SOM and DSOM**, IBM International Technical Support Centers, Document No. GG24-4357-00, July 1994.

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

[61] Kaashock, M. F., and Tanenbaum, A. S., *Group Communication in the Amoeba Distributed Operating System*, Proceedings of the 11th International Conference on Distributed Computing Systems, pp. 222-230, May 1991.

[62] Kernighan, B. W., and Ritchie, M. R., **The C Programming Language - Second Edition**, Prentice Hall, 1988.

[63] 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.

[64] Kramer, J., Magee, J., and Finkelstein, A., *A Constructive Approach to the Design of Distributed Systems*, Proceedings of the 10th International Conference on Distributed Computing Systems, pp. 580-587, June 1990.

[65] Kramer, J., Magee, J., and Sloman, M., *Configuration Support for System Description, Construction and Evolution*, **Managing Complexity in Software Engineering**, pp. 247-260, edited by R. J. Mitchell, Peter Pereginus Ltd., 1990.

[66] Kremien, O., and Kramer, J., *Methodical Analysis of Adaptive Load Sharing Algorithms*, IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 6, pp. 747-760, November 1992.

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

[68] Krueger, P., and Livny, M., *A Comparison of Preemptive and Non-Preemptive Load Distributing*, Proceedings of the 8th International Conference on Distributed Computing Systems, pp. 123-130, June 1988.

[69] Kunz, T., *The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme*, IEEE Transactions on Software Engineering, Vol. 17, No. 7, pp. 725-730, July 1991.

[70] Leffler, S. J., McKusick, M. K., Karels, M. J., and Quaterman, J. S., **The Design and Implementation of the 4.3BSD Unix Operating System**, Addison-Wesley, 1989.

[71] Litzkow, M. and Solomon, M., *Supporting Checkpointing and Process Migration Outside the Unix Kernel,* Proceedings of the Winter 1992 USENIX Conference, pp. 283-290, 1991.

[72] Liu, R.-C., *Analysis on the Effects of System Parameters on Load Balancing*, Information Sciences, Vol. 81, No. 1-2, pp. 37-54, November 1994.

[73] Lord, H. D., *Improving the Application Development Process with Modular Visualization Environments*, Computer Graphics, Vol. 29, No. 2, May 1995.

[74] Mechoso, C. R., Farrara, J. D., Spahr, J. A., *Running a Climate Model in a Heterogeneous, Distributed Computer Environment,* Proceedings of the Third International Symposium on High Performance Distributed Computing, pp. 79-84, August 1994.

[75] MacDougall, M. H., **Simulating Computer Systems - Techniques and Tools**, MIT Press, 1987.

[76] Mattson, T. G., *Programming Environments for Parallel Computing: A Comparison of CPS, Linda, P4, PVM, POSYBL, and TCGMSG*, Proceedings of the 27th Annual Hawaii International Conference on Systems Science, Vol. II, pp. 586-594, January 1994.

[77] Mirchandaney, R., Towsley, D., and Stankovic, J. A., *Analysis of the Effects of Delays on Load Sharing*, IEEE Transactions on Computers, Vol. 38, No. 11, pp. 1513-1525, November 1989.

[78] Mullender, S.J., and van Rossum, G., *Amoeba: A Distributed Operating System for the 1990s,* Computer, pp. 44-53, Vol. 23, No. 5, 1990.

[79] Munetomo, M., Takai, Y., and Sato, Y., *A Genetic Approach to Dynamic Load Balancing in a Distributed Computing System*, Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Vol. 1, pp. 418-421, June 1994.

[80] Mutka, M. W., *Estimating Capacity for Sharing in a Privately Owned Workstation Environment,* IEEE Transactions on Software Engineering, Vol. 18, No. 4, 1992.

[81] Mutka, M. W., and Livny, M., *The Available Capacity of a Privately Owned Workstation Environment,* Performance Evaluation, pp. 269-284, Vol. 12, No. 4, July 1991.

[82] The Object Management Group, **The Common Object Request Broker: Architecutre and Specification**, OMG Document Number 93.12.43, December 1993.

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

[84] Park, C., and Kuhl, J. G., *A Fuzzy-Based Distributed Load Balancing Algorithm for Large Distributed Systems*, Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems, pp. 366-273, April 1995.

[85] Peterson, L., Hutchinson, N., O'Malley, S., and Rao, H., *The x-Kernel: A Platform for Accessing Internet Resources,* Computer, pp. 23-33, Vol. 23, No. 5, May 1990.

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

[87] Rehmet, G. M., **Remora - Implementing Adaptive Parallelism on a Heterogeneous Cluster of Networked Workstations**, Master's Thesis, Rhodes University, South Africa, September 1994.

[88] Rinaldo, F., and Wolbers, S., *Loosely Coupled Parallel Processing at Fermilab*, Computers in Physics, Vol. 7. No. 2, pp. 184-190, March 1993.

[89] Rosenberry, W., Kenney, D., and Fisher, G., **Understanding DCE**, O'Reily & Associates, 1992.

[90] Rousselle, P., Hariri, S., and Tymann, P, *Redundant Task Scheduling in a Heterogeneous Computing Environment*, Northeast Parallel Architectures Center, Technical Report SCCS-658, October 1994.

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

[92] Rousselle, P., Yadav, R., Hariri, S., Kim, D., Boddapati, S., and Tymann, P., *Application Development and Execution in a Virtual Computing Environment*, Northeast Parallel Architectures Center, Technical Report SCCS-705, April 1995.

[93] Schaar, M., Efe, K., and Delcambre, L. M., *An Analytical Model for Predicting Performance in a Heterogeneous System*, Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Computing Systems, pp. 328-335, May 1991.

[94] Shoja, G.C., *A Distributed Facility for Load Sharing and Parallel Processing Among Workstations,* Journal of Systems Software, pp. 163-172, Vol. 14, No. 3, 1991.

[95] Simpson, R. A., and Harget, A. J., *A Simulation Study to Determine the Importance of Load Balancing Algorithms for Loosely Coupled Distributed Systems*, IFIP Transactions on Computer Science and Technology, IFIP 13th World Computer Congress, Vol. A-51, pp. 69-74, August 1994.

[96] Stevens, W. R., **UNIX Network Programming**, Prentice Hall, 1990.

[97] Stroustrup, B., **The C++ Programming Language - Second Edition**, Addison-Wesley, 1991.

[98] Stroustrup, B., **The Design and Evolution of C++**, Addison-Wesley, 1994.

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

[100] Tannenbaum, A. S., **Modern Operating Systems**, Prentice-Hall, 1992.

[101] Theimer, M., and Lantz, K., *Finding Idle Machines in a Workstation-Based Distributed System,* IEEE Transactions on Software Engineering, pp. 1444-1458, Vol. 15, No. 11, 1989.

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

[103] Thornborrow, C., Wilson, A. J. S., and Faigle, C., *Developing Modular Application Builders to Exploit MIMD Parallel Resources*, Proceedings of Visualization '93, pp. 134-141, San Jose CA, October 1993.

[104] 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.

[105] Walker, B., Popek, G., English, R., Kline, C., and Thiel, G., *The LOCUS Distributed Operating System,* Proceedings of the 1983 SIGOPS Conference, pp. 49-70.

[106] Wang, C. -J., Krueger, P. E., and Liu, M. T., *Intelligent Job Selection for Distributed Scheduling,* Proceedings of the Thirteenth International Conference on Distributed Computing Systems, pp. 517-524, May 1993.

[107] Wood, M. D., *Replicated RPC Using Amoeba Closed Group Communication*, Proceedings of the 13th International Conference on Distributed Computing Systems, pp. 222-230, 1993.

[108] Zhou, S., *A Trace-Driven Study of Dynamic Load Balancing*, IEEE Transactions on Software Engineering, Vol. 14, No. 9, pp. 1327-1341, September 1988.

[109] Zhou, S., Zheng, X., Wang, J. and Delisle, P., *Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems*, Software - Practice and Experience, pp. 1305-1336, Vol. 23, No. 12, December 1993.

[110] Zhu, W., and Goscinski, A., *Low Cost Load Balancing Algorithms*, Proceedings of the 27th Annual Simulation Symposium, pp. 226-235, April 1994.

Biographical Data

Name:	Philip Rousselle
Date and Place of Birth:	September 10, 1962 Buffalo, New York
Elementary & High School:	Buffalo Public Elementary School #84 Graduated 1976
	Kensington High School Buffalo, New York Graduated 1980
College:	SUNY College at Buffalo BS (Info Systems Management), 1983
	Union College Schnectady, New York MS (Computer Science), 1989
Employment	International Business Machines Inc Endicott NY