Chapter 4

The VCE Application Development Model

The previous chapter introduced the two major components of the VCE. They are, the *software development suite*, which provides software engineering tools for designing and developing large applications, and the *execution suite* which manages dispatching, scheduling and communication among application components across a distributed heterogeneous hardware configuration. A central design element of both these components is the *VCE application description framework*. Each VCE application has a corresponding description expressed in the semantics of the application description framework. The description captures all the design information produced by the specification, design and coding tools which make up the software development suite. The execution suite is supplied an application description as input. It uses the description, and the executable and data files the description refers to, to execute the application. The application description framework is the principle interface between the two major components of the VCE. Tools in the software development suite are geared towards producing and refining an application description while the technology in the execution suite interprets and executes it.

The structure of the application description framework dictates the capabilities which are required of tools included in the software development suite. That is,

any tool, or set of tools, which can produce output which can be translated into VCE application description framework syntax can be included in the software development suite.

4.1 The VCE Application Construction Approach

The design of the software development suite, as laid out in Section 3.2, includes tools that assist the user in all phases of application design, development and implementation. Specification tools help the user to express the characteristics of the problem. Design tools are used to specify the major components of the application, their hardware, synchronization and communication requirements. If an application can not be completely implemented using existing components, compilation tools are available to assist the user in implementing new executable tasks. Tools supporting any development methodology (procedural, object oriented, etc.) or multiple methodologies can be included in the development suite. The only requirements is that they produce application descriptions expressed in, or translatable to, the syntax of the VCE application description framework.

Section 2.11 discussed the differentiation between programming-in-the-small and programming-in-the-large [35]. Unlike most of the parallel/distributed environments discussed in Chapter 2, the VCE encourages a strong separation between high performance programs and high performance applications. In building a Grand Challenge application it may be necessary to exploit a variety of hardware architectures, programming languages and programming tools in a single application. Rather than providing more sophisticated programming paradigms for expressing complex cross-platform communication and synchronization operations, the VCE application description framework allows programs written in any language on a variety of architectures to be incorporated into an application. By isolating dispatching, synchronization and communication specifications in a separate description file, the source code for individual computing tasks can be simplified. It is also easier to reuse programs if they are not dependent on the structure of the applications they are used in [86].

The VCE application description framework is a synthesis of, and extension to, two existing technologies: distributed application description languages (Section 2.8) and distributed object management systems (Section 2.9). Both of these approaches provide mechanisms for expressing how a large software system is decomposed into smaller components and how they are to be dispatched on a distributed hardware configuration to form an application. The VCE application description framework builds on the most powerful characteristics of both these technologies. Unlike current distributed application description languages, the VCE application description framework provides facilities to describe the structure of object oriented applications. It provides a syntax that is simple and flexible enough to support a variety of software engineering approaches including procedural, object oriented and data flow. It can easily be extended to accommodate development models that will evolve in the future. It is also intended for more general purpose uses than distributed application description languages that are specifically designed for fault tolerance and dynamic reconfiguration. Unlike distributed object systems, the application description framework allows application structure to be expressed outside of, and therefore independently of, the source code of the programs that make up the application. This eases program reuse and gives the run time system the greatest possible flexibility in dispatching.

The remainder of this chapter will focus on the structure of the VCE application description framework. The framework defines what is required of tools in the software development suite. It illustrates the VCE application model where *applications* are constructed from smaller computational units such as *tasks* or *method implementations* which are performed by *executable files* running on hardware *platforms*. Tools in the VCE software development suite help users to construct applications in these terms. The application description framework is the bridge between the tools in the software development suite and the distributed application scheduling, dispatching, control and communication tools in the VCE execution suite.

4.2 The VCE Application Description Framework

Durra (Section 2.8) and DSOM (Section 2.9) are two very different approaches to distributed application development environments. The differences between them are largely due to their different objectives. Durra is oriented towards creation of fault tolerant embedded systems while DSOM is meant to ease development of object oriented systems which access remote shared resource.

The objectives of the VCE are also unique. These objectives are to support development of Grand Challenge applications which exploit multiple supercomputers and workstation clusters, and better utilize these hardware facilities. As in Durra, The VCE uses task descriptions, which correspond to programs, and application descriptions which specify the communication and synchronization patterns among a group of tasks. However, while there is a one-to-one correspondence between task descriptions and executable files in Durra (Figure 7), a single VCE task description may point to executable files on several different architectures. This allows the VCE run time system to dynamically choose which machine to run the task on, or to run it on several machines redundantly. Another unique feature of the VCE is that application specification is kept as simple as possible. Both Durra and DSOM consider the application implementer to be a programmer. In the VCE, it is expected that applications will be constructed by scientists and engineers. Therefore, VCE application descriptions only describe the communication and synchronization requirements of a group of tasks or objects. Decisions about how to allocate tasks to processors are made by task implementers or the run time system.

The remainder of this section introduces the elements of the VCE application description framework and illustrates its use to construct an example application.

4.2.1 VCE Platforms

A VCE *platform* is any computing system capable of performing computations for a VCE application. Parallel supercomputers, clusters of workstations and individual high performance workstations can all be regarded as potential VCE platforms.

4.2.2 VCE Tasks

Within the VCE, a *task* refers to a dispatchable entity. A task may be a parallel program and may, therefore, execute on several nodes of a single platform. A task may not simultaneously execute on nodes of more than one platform. For example, a task might require four nodes on an IBM SP2 supercomputer or four nodes from a DEC Alpha workstation cluster. However, a computation that required two nodes of an IBM SP2 and two nodes of a DEC Alpha workstation cluster could not be regarded as a single task.

A single task may have implementations on many platforms. That is, there may be executable files corresponding to a task on multiple platforms which perform the same computations and have the same interfaces.

4.2.3 Intertask vs. Intratask Communication

If a task is implemented as a parallel program on some platform, the communication among the participating nodes is managed by the task. The task could use any parallel programming mechanism such as a message passing tool [99] or parallel programming language [46] to perform intratask communication. Intratask communication is distinct from the task's interfaces, which define how it can interact with other tasks and the VCE.

Intertask communications are managed by the VCE. The application description framework includes a syntax for specifying the intertask communication requirements of application components. The task description syntax (Section 4.2.4) allows the task developer to specify the input consumed and output produced by a task. The application description syntax (Section 4.2.6) allows the application developer (or application development tool) to indicate how outputs of one task or group of tasks are to be routed as input to another. Section 5.2 will discuss how these specifications are used by the execution suite to construct *communication processes* which send output and receive input on behalf of application tasks. These communication processes use the information captured in the application description framework to provide location

[**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_object_name.input_object_type
<input_object_name2.input_object_type2>
< ... >>
<**output:** output_object_name.output_object_type
<output_object_name2.output_object_type2>
< ... >>
<**hints:** <replicateable | non-replicateable>
<duration: short | medium | long>
<method-implementation>>

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

Figure 22: VCE Task Description Syntax

transparent intertask communication functions among application components. They also allow redundantly executing tasks to communicate transparently.

4.2.4 VCE Task Description Syntax

Each task used in a VCE application has a corresponding *task description*. The task description documents various characteristics of the task which enable the VCE run time system to dispatch it and manage its communications.

The syntax of a VCE task description is shown in Figure 22. The parameters of the description have the following meanings:

task_name Is a mnemonic name for the task which will be used to refer to the task in application descriptions. In most cases the name of the task description file should match the task_name.

architecture: path_name The *architecture* refers to a platform architecture. For example, in one VCE prototype implementation (to be discussed in Chapter 6) valid architecture specifications were **SP2** to indicate an IBM SP2, **ALPHA** to indicate a DEC Alpha cluster, and **SUN** for a Sun workstation cluster. The *path_name* identifies an executable file which implements the task on the given architecture.

architecture: number_of_nodes For each target platform this parameter indicates how many nodes the task is to be allotted.

command_line_parameter Is a string to be passed to the task as a command line parameter.

input_object_name.input_object_type Defines the name and type of an input object to be supplied to the task. Details of how input and output parameters are supplied to tasks will be discussed in Section 4.2.5.

output_object_name.output_object_type Defines the name and type of an output object produced the task (see Section 4.2.5).

replicateable‖non-replicateable The run time system uses redundant execution to achieve load balancing and improved performance (Section 5.1). A task is *replicateable* if it does not read any file or shared data object which it also modifies. If this is true, and file updates are made by writing rather than appending, then the task will function correctly even if multiple instances are executing concurrently.

duration: short‖medium‖long This information helps the scheduler determine when redundant scheduling will be beneficial.

method-implementation The task is an implementation of a CORBA compliant [82] object method. The first object listed in the input specification is taken to be the object which the method is to be applied to. The type of this object

task: LU_decomposition
executables: SUN: LU_decomposition.sun
SP2: LU_decomposition.sp2
ALPHA: LU_decomposition.alpha
nodes: SUN: 4
SP2: 4
ALPHA: 4
input: rectangular_matrix.file
output: lower_triangle.matrix
upper_trangle.matrix
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
input: input.matrix
output: inverted.matrix
hints: replicateable

task: multiply_matrix_by_vector_file
executables: SUN: matrix_multiply_1.sun
SP2: matrix_multiply_1.sp2
ALPHA: matrix_multiply_1.alpha
input: input.matrix
input_vector.file
output: output.vector
nodes: SUN: 4
SP2: 4
ALPHA: 4
hints: replicateable

task: multiply_matrix_by_vector_giving_file
executables: SUN: matrix_multiply_2.sun
SP2: matrix_multiply_2.sp2
ALPHA: matrix_multiply_2.alpha
input: input.matrix
vector_input.vector
output: output_vector.file
nodes: SUN: 4
SP2: 4
ALPHA: 4
hints: replicateable

Figure 23: VCE Task Descriptions for Linear Algebra Operations

indicates the class which the method is a member of. The *task_name* indicates the name of the method. More details of the use of object implementations in VCE applications are given in Section 4.2.7.

An example of task descriptions for a set of common linear algebra operations is given in Figure 23.

4.2.5 Task Input/Output Semantics

Every distributed system must provide facilities for communication between different components of a distributed application. In Durra this is done with the **port** construct. Durra task programs use funtions provided by the Durra library to write data to, and read data from, **ports**. Application descriptions specify how the **ports** of various tasks are interconnected via **channels**. DSOM programs communicate with remote objects by invoking methods on their proxies. The interfaces and remote locations of the distributed objects are maintained in the interface and implementation repositories.

VCE_send(output_object_name <, output_object_namr2> <...> **)**

VCE_receive(input_object_name <, input_object_name2> <...> **)**

VCE_test(input_object_name <, input_object_name2> <...> **)**

Figure 24: VCE Communication Operations

The VCE provides three functions which are used for intertask communications (Figure 24). The **VCE_send()** function transmits data to other tasks. The `object_name` specified, which identifies a program variable, must correspond with an `object_name` listed as an **output** object in the task description. The type of the output variable must match the `object_type` indicated in the task description. For instance, the `inverse_matrix` task of Figure 23 must perform the call `VCE_send(inverted)`, where `inverted` is an object of type *matrix*, in order to make its results available to other tasks. Similarly, **VCE_receive** is a blocking receive which retrieves data supplied by other tasks. Again, the variable received must correspond to an **input** parameter in the task description. **VCE_test** returns *true* if new input is available for all the variables listed.

Object types are defined using IDL [82] in a VCE interface repository.

The `file` object type has a special connotation. Performing a **VCE_receive()** on an object of type `file` returns a string which is the name of a file in the task implementation's execution context which holds the actual data. Therefore, it is permitted to do a **VCE_receive()** on a `file` object even if it is specified as an **output** parameter. Such a call would return the name of a file to be used by the task to hold its output. Details on the use of `file` objects are included in Section 4.2.6.

4.2.6 VCE Application Description Syntax

A VCE application description contains instructions on how the run time system is to dispatch and interconnect tasks to realize an application. The application description syntax is shown in Figure 25. Applications are divided into *stages*. Each stage

[**application:** application_name]

[**stage:** stage_name

[**task:** task_name]
<**nodes:** architecture: number_of_nodes
<architecture2: number_of_nodes2>
< ... >>
<**command:** comand_line_parameter <command_line_parameter2> < ... >>
<**input:** input_object_name
<input_object_name2>
< ... >>
<**output:** output_object_name
<output_object_name2>
< ... >>
<**hints:** <duration: short | medium | long>>
<**synchronization:** after other_stage_name begins | completes>
<after other_stage_name2 begins | completes>
< ... >>

<**stage:** stage_name2

[**task:** task_name]
<**nodes:** architecture: number_of_nodes
<architecture2: number_of_nodes2>
< ... >>
<**command:** comand_line_parameter <command_line_parameter2> < ... >>
<**input:** input_object_name
<input_object_name2>
< ... >>
<**output:** output_object_name
<output_object_name2>
< ... >>
<**hints:** <duration: short | medium | long>>
<**synchronization:** after other_stage_name begins | completes>
<after other_stage_name2 begins | completes>
< ... >>

< ... >

Figure 25: VCE Application Description Syntax

employs a task to perform some computation. Many of the specifications of a task can be overridden in an application description. The only values which cannot be changed are the **executables**, the *types* of input and output objects, and the *replicateable* and *method-implementation* hints. The values specified for parameters in a task description which can be overridden later constitute the default values for those parameters.

Each application stage can include **synchronization** specifications which indicate the order in which tasks are to be performed or method implementations instantiated. The **synchronization** parameters are a list of conditions which must be satisfied before the task can be dispatched. Obviously, if one stage performs operations on the output of an earlier stage, then the later stage would list completion of the earlier stage as a synchronization constraint. Special synchronization specifications are also needed if one stage sends data to a concurrently executing stage and the waits to receive results back. In this case synchronization directives are used to indicate that the former task is to be dispatched after the latter task is started.

Interconnections between tasks are achieved by manipulating the object names on the **input** and **output** directives. If an output `object_name` of one task and a compatible input `object_name` of another task are overridden with the same value, it will indicate to the VCE execution process that the output of the one task is to be routed as input to the other.

Figure 26 shows an example of how the tasks of Figure 23 can be used to construct a linear equation solver application. The example illustrates several features of how the VCE run time system interprets an application description. The application accepts as input the files `A_matrix_file` and `B_vector_file` and produces `X_vector_file` as output. All the input and output default `object_name`s established in the task descriptions are overridden to achieve the desired data flow. The **synchronization** parameters are used to impose the correct order of operations. The execution process can determine, through analysis of the application description, that no output of any task will be used as input by a concurrently executing task. That is, all intertask communication is performed by buffering the outputs of earlier tasks

Application Description

application: solver

stage: LU_decompose
task: LU_decomposition
input: A_matrix_file
output: lower, upper

stage: inverse_matrix_1
task: inverse_matrix
input: lower
output: lower_inverse
synchronization: after LU_decompse completes

stage: inverse_matrix_2
task: inverse_matrix
input: upper
output: upper_inverse
synchronization: after LU_decompse completes

stage: first_matrix_multiply
task: multiply_matrix_by_vcetor_file
input: lower_inverse, B_vector_file
output: temp_vector
synchronization: after inverse_matrix_1 completes

stage: second_matrix_multiply
task: multiply_matrix_by_vector_giving_file
input: upper_inverse, temp_vector
output: X_vector_file
synchronization: after first_matrix_multiply completes
after inverse_matrix_2 completes

Application Graph

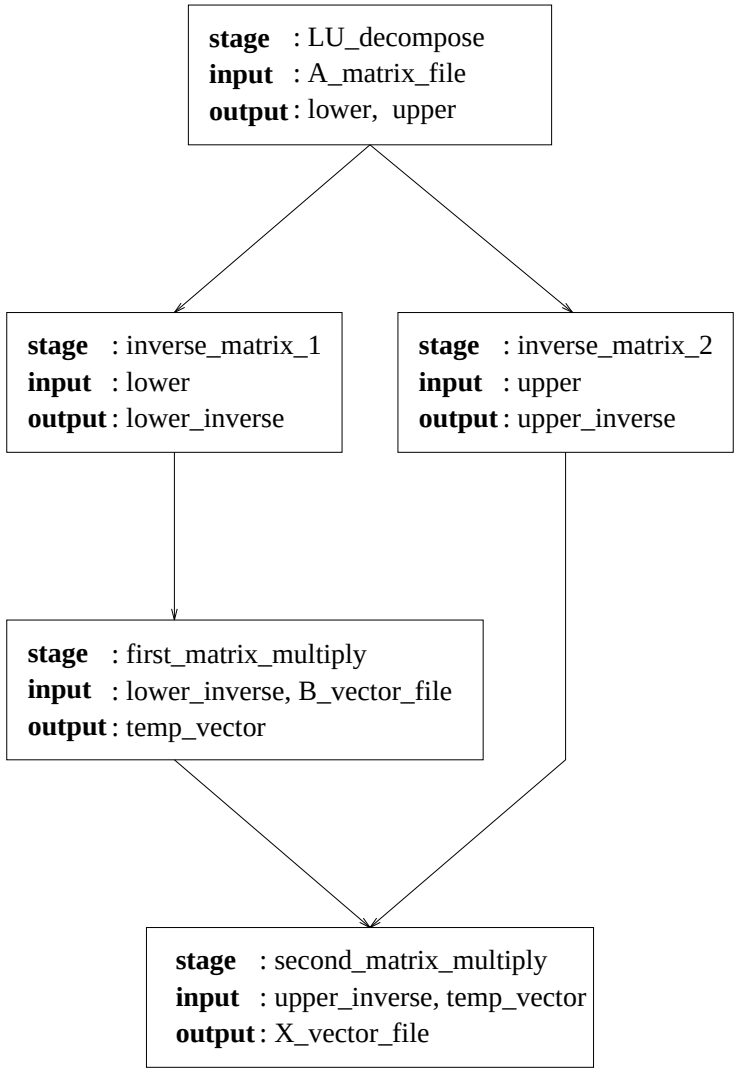

Figure 26: A VCE Application

and providing them as inputs to later tasks.

Figure 26 also illustrates the use of `file` objects in an application description. The task description of `LU_decomposition` (Figure 23) includes the specification "**input:** `rectangular_matrix.file`". The **input** specification of the linear equation solver's `LU_decompose` stage overrides `rectangular_matrix` with `A_matrix_file`. When an implementation of the `LU_decomposition` task issues `VCE_receive(rectangular_matrix)` it will be given the string "`A_matrix_file`" so it can open the file and read its input. The VCE execution process insures that each machine which runs the `LU_decomposition` has access to the `A_matrix_file`.

The three VCE communication operations and the application description framework can be used to construct a variety of application data flow patterns. For instance, if the output of one task is specified as input for several others, then a **VCE_send()** of that output has the effect of a broadcast operation. Similarly, **VCE_receive** can be used to collect output objects from multiple tasks. These communication patterns are established by the way the tasks are used in application description. The same task which feeds input to a single successor task in one application can broadcast its results to many concurrent tasks in another application. This difference in behavior can be accomplished without changing the task implementation or description or recompiling the task.

The application description framework accomplishes several VCE objectives. The syntax is simple, so with just a few pages of documentation it should be easy for a non-programmer to construct an application from a task library. One reason the description syntax is simple is that decisions about how work is assigned to particular machines are made by the run time system rather than the application designer. Facilities for intertask communication are also constructed dynamically at run time rather than being specified in the application description. This simplicity and emphasis on allowing the run time system to deal with as many application execution issues as possible will make it easy to construct intuitive design tools in the Software Development Module.

4.2.7 Object Oriented Input/Output Semantics

In addition to the message oriented communication semantics described in Section 4.2.5, the VCE also uses the syntax of Figure 22 to support object oriented communication semantics. If the hint `method-implementation` is included in a task description, it signals the VCE to treat the executable as a CORBA compliant [82] method implementation. An object method packaged as a VCE task is called a *VCE remote method.* A task can only contain a single method definition. In object oriented environments, CORBA style interface and implementation repositories would be produced by the software development suite as well as the task and application description libraries. The first object listed on the **input** specification of an `object-implementation` task is considered to be the object the method is applied to and identifies the class the method is a member of. The *task_name* must match the name of the remote method. At run time the VCE execution suite checks that the input and output specifications of the task description are a superset of the method signature in the interface repository. Additional input and output objects may be included in the task description to allow the method to invoke other VCE tasks or methods. That is, the **input** objects include parameters received on invocation as well as objects returned by called tasks. Similarly, **output** objects may be passed to called tasks or returned to the method's invoker. Both the task description and the interface repository entries are produced by the tools in the software development suite and the tools insure that the specifications are compatible.

There are two ways VCE remote methods can be invoked, *implicitly* and *explicitly.* Implicit invocations occur when the application description shows that all a method's inputs are supplied by tasks that are predecessors and all the method's return values are used by successor tasks. For example, if all the sample task descriptions shown in Figure 23 included the `method-implementation` hint, the application description of Figure 26 would still be valid. For instance, the `inverse_matrix_1` stage would invoke the method defined by the `inverse_matrix` task on the `matrix` object `upper` and the returned object, `upper_inverse`, would be supplied as one of the inputs to

the subsequent `first_matrix_multiply` stage.

Explicit VCE remote method invocations would be indicated by placing an asterisk after the name of a method being invoked in a source program. This would cause the compilation tools to produce code to alert the run time system to the remote method invocation. The execution suite would use the input and output specifications of simultaneously executing tasks, and its knowledge of the methods they implement, to execute the correct remote method. Again, it is the responsibility of tools in the software development suite to insure that ambiguities do not arise.

4.2.8 Other Communication Semantics

Sections 4.2.5 and 4.2.7 showed how the syntax of the VCE application description framework could be used to support two different communication semantics (massage passing and object oriented). Further extensions to support other paradigms such as distributed shared memory or data flow are also possible. The task description syntax is a language independent and paradigm independent method of indicating the input and output characteristics of a course grain application component, the executable files which implement it, and the hardware requirements of those executables.

4.2.9 Designing VCE Tasks and Applications

The VCE application and task concepts reinforce the notion that task design (program design) and application design are separate activities. This will make the development of grand challenge applications easier by allowing software engineering activities to be more specialized. Different members of a development team can focus on areas where they have particular expertise. For instance, one possible to organize members of a VCE development team would be to divide it into *task designers*, *task developers* and *application designers*.

Task designers can focus on providing general purpose software components with well defined interfaces and high granularity. The goal of the task designer is to anticipate the needs of the application designer.

The *task implementer*, who is presumed to be knowledgeable in parallel programming techniques, can select the best available tool for implementing each task on each platform where it is sensible to do so. Task implementers are the only team members who must be skilled programmers. They can focus on optimizing task implementations on the widest possible variety of high performance architectures.

Application designers should be experts in the application area. They do not need to be concerned with the details of how tasks are implemented or what particular machines are available. Their only concern is to determine how to express an application in terms of the task implementations included by the team's task designers in the VCE repertoire.

The ability of the VCE to enable experts in application areas to construct and execute very large applications that exploit multiple high performance computing systems, without having to master programming languages and other tools, represents a significant advantage over the systems described in Chapter 2.

4.2.10 Extensions to the Application Description Framework

A variety of possible extensions to the application description framework which will make it more flexible, robust and expressive will be discussed in the "Conclusions and Future Work" chapter of this dissertation (Section 7.2.1). Extensions to the syntax can be implemented easily because of the flexibility of the underlying design. One design philosophy that permeates VCE research is that programs (task or object implementations) should be developed independently to encapsulate a general purpose coarse grain computational function. Applications can be developed by expressing how a group of tasks are to be dispatched and interconnected. The application description framework is the VCE approach to communicating the charastics of tasks and applications to the run time execution tools. While the framework may be made more useful by the addition of syntactical elements to express conditional or iterative execution, or other computational characteristics, these can be built upon the current framework of a straight forward task and application description syntax specifically

oriented towards high performance distributed heterogeneous computing.

4.3 Application Execution

The application description framework provides a mechanism for capturing information about the structure of a large scale application. The major phases (tasks) of an application can be represented along with their hardware, file, synchronization and communication requirements. The syntax is easy to parse. In a single pass it is possible for the VCE execution process to build data structures that represent each phase in an application. Tasks that have no synchronization constraints are dispatched initially. Each time a task completes other tasks are checked to determine if they are now ready to begin.

The execution suite judges the run time conditions and maps application tasks to available hardware resources. It determines if redundant task scheduling is appropriate and dynamically allocates intertask communication facilities. The scheduling, dispatching and communication facilities provided by the execution suite are the subject of Chapter 5.

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