RL-TR-95-116
Final Technical Report
July 1995

SYSTEM RESOURCE MANAGEMENT FOR DISTRIBUTED REAL-TIME SYSTEMS

SRI International

Michael Davis, Elin L. Klaseen, Louis C. Schreier,
Alan R. Downing, Jon Peha, Raymond Clark, Douglas Jensen,
Ira B. Greenberg, and Nagendra Siravara

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

RL-TR-95-116 has been reviewed and is approved for publication.

APPROVED: [illegible]

THOMAS F. LAWRENCE
Project Engineer

FOR THE COMMANDER: [illegible]

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE July 1995	3. REPORT TYPE AND DATES COVERED Final Aug 91 - Nov 93

4. TITLE AND SUBTITLE SYSTEM REOURCE MANAGEMENT FOR DISTRIBUTED REAL-TIME SYSTEMS	5. FUNDING NUMBERS C - F30602-91-C-0099 PE - 62702F PR - 5581 TA - 21 WU - 97
6. AUTHOR(S) Michael Davis, Elin L. Klaseen, Louis C. Schreier, Alan R. Downing, Jon Peha, Raymond Clark, Douglas Jensen, Ira B. Greenberg, and Nagendra Siravara	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) SRI International 333 Ravenswood Avenue Menlo Park CA 94025-3493	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3AB) 525 Brooks Rd Griffiss AFB NY 13441-4505	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-95-116

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Thomas F. Lawrence /C3AB/(315) 330-2925

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The overall goal of the SRM effort is to develop techniques for coordinated but decentralized control of system resources in distributed systems. These techniques will provide integrated management of: (1) multiple system resources, including the processing, communication, and data storage components of the system; (2) multiple system objectives (i.e., level-of-service preferences), such as timeliness, precision, and correctness; (3) multiple activities (e.g., applications or tasks) competing for resources; (4) multiple nodes in a distributed system; and (5) multiple techniques and mechanisms for meeting objectives. The resource management system will allocate distributed system resources to multiple competing activities in such a way that objectives as a whole can be satisfied to the highest degree possible. The goals of the research for this effort were to: (1) develop abstractions that model common system objectives such as performance, functionality, availability, precision, and security; (2) develop abstractions that model system components (e.g., processing, communication, and storage subsystems) and higher level services that are based on these components (e.g., distributed database systems or distributed file systems); (3) develop abstractions and mechanisms that provide integrated control across the objectives and components of a distributed system; and (4) show the validity of the abstractions by applying them to one or more sample scenarios (see reverse)

14. SUBJECT TERMS System resource management, Distributed systems, Adaptive control, Quality of service, Multimedia control, control integration	15. NUMBER OF PAGES 106
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

13. (Cont'd)

and implementing a proof-of-concept demonstration application.

ACKNOWLEDGEMENTS

The author wishes to thank the following individuals and organizations, without whose contributions this report could not have been written:

Elin L. Klaseen and Louis C. Schreier of the Information, Telecommunications, and Automation Division, *SRI International*

Alan R. Downing, *Oracle Corporation*

Jon Peha, *Carnegie Mellon University*

Raymond Clark, *Open Software Foundation*

Douglas Jensen, *Digital Equipment Corporation*

Ira B. Greenberg, *Informix Corporation*

Nagendra Siravara, *Siemens Rolm Communications, Inc.*

CONTENTS

FIGURES

1 INTRODUCTION

This final technical report summarizes the work performed for the project entitled "System Resource Management for Distributed Real-Time Systems," in fulfillment of contract data item CLIN 008. SRI International (SRI) is investigating mechanisms and techniques for (1) a distributed real-time operating system characterized by the global, decentralized control of processing, storage, and communications components that are coordinated toward a common goal; and (2) global performance optimization, within consistency and security constraints. We have implemented a simple distributed application that demonstrates the feasibility of these techniques.

1.1 BACKGROUND

Military computer systems are becoming larger, more complex, and more distributed. In addition, they are increasingly required to operate in a highly dynamic environment characterized by

- Limited processing, communication, and data storage resources, and unpredictable loss of these resources
- Dynamic topology and configuration
- Changing threats and modes of operation
- Time-constrained operations.

Despite this hostile and complex environment, the systems are required to be high performance, secure, reliable, and survivable.

Unfortunately, few operating systems used in military systems (or in commercial applications, for that matter) have satisfactory support for many of these requirements. For example, few operating systems consider time constraints, and those that do typically support hard real-time constraints that are too restrictive for many applications in this environment. Instead, systems that support soft real time (for example, the Alpha operating system [Northcutt and Clark 1988]), and consider both the importance and the time constraints of competing activities when determining an execution plan, are more appropriate. Soft real-time systems support the graceful degradation of performance by dropping low-priority tasks during overloads. However, if resource availability is too limited or highly variable, additional support that considers resource limitations is required.

In addition, current operating systems manage individual system resources such as processing, communication, and data storage in an *ad hoc* manner. Different policies are used to manage different resources, and the management of the different resources is not coordinated, particularly when the resources are distributed. The relative importance of different activities is seldom considered in a uniform way across all the system resources. Such uncoordinated management results in suboptimal use of resources, especially when the availability of resources changes. For example, if radar data about incoming missiles are processed at a higher priority than sensor data about the weather, but weather data are transmitted on the communication channels at the same priority as missile data, then processing and communication resources will be wasted and military objectives may not be met.

Another problem is that current systems are too static for a dynamic military environment. A nonadaptive system can fail if it incorrectly assumes that its environment is known and predictable. Distributed systems in the military environment are subject to long-term, permanent changes due to failures or configuration modifications. Such changes mean that the underlying system will fail or will perform inefficiently unless fault tolerance and/or adaptability is incorporated into the system. Abstractions currently used by operating systems are not expressive enough to support adaptability. The ability to express and utilize tradeoffs in terms of resources used and benefit gained is missing. For example, the ability to select among alternate tasks with varying degrees of precision is not supported.

Under the sponsorship of Rome Laboratory (RL), the problem of allocating limited resources among competing activities, according to command preferences for the activities and the usage restrictions imposed by the resources, has begun to be addressed in such projects as the Alpha Soft Real-Time Operating System (Alpha) project, the Adaptive Fault Resistant Systems (AFRS) project, and the Cooperative Gateway project. Many important issues in integrated resource management have not been addressed by these projects; the System Resource Management (SRM) project is intended to address these issues. Alpha considers both the importance and the urgency (time constraints) of competing activities when determining an execution schedule, but it currently does not handle attributes other than time constraints (for example, precision requirements), and it does not coordinate the management of resources other than processing time; nevertheless, it provides a good starting point for further research. The AFRS project focuses on fault tolerance, but its adaptability techniques may be generalized to include other application objectives. The Cooperative Gateway project deals mainly with communication resources and is not explicitly concerned with real-time issues; however, its use of policies for making routing decisions may be applicable to some aspects of integrated resource management.

1.2 OVERVIEW OF PROJECT GOALS

The overall goal of the SRM program is to develop techniques for coordinated but decentralized control of system resources in distributed systems. These techniques will provide integrated management of (1) multiple system resources, including the processing, communication, and data storage components of the system; (2) multiple system objectives (i.e., level-of-service preferences) such as timeliness, precision, and correctness; (3) multiple activities (e.g., applications or tasks) competing for resources; (4) multiple nodes in a distributed system; and (5) multiple techniques and mechanisms for meeting objectives. The resource management system will allocate distributed system resources to multiple competing activities in such a way that the objectives as a whole can be satisfied to the highest degree possible.

This project is the first phase of a multiphase program. During this phase we have developed and demonstrated the major concepts to be refined in later phases. The goals of the research for this phase were to:

1. Develop abstractions that model common system objectives such as performance, functionality, availability, precision, and security
2. Develop abstractions that model system components (e.g., processing, communication, and storage subsystems) and higher level services that are based on these components (e.g., distributed database systems or distributed file systems)

3. Develop abstractions and mechanisms that provide integrated control across the objectives and components of a distributed system
4. Show the validity of the abstractions by applying them to one or more sample scenarios and implementing a proof-of-concept demonstration application.

Our goal was not to develop techniques, mechanisms, or strategies that satisfied specific objectives, but instead to develop a model that integrated existing and future techniques so as to enable them to work together toward a common set of objectives.

1.3 OVERVIEW OF PROJECT APPROACH

Our approach to developing a model of system resource management for distributed real-time systems was (1) to extend and generalize the abstractions used in the Alpha soft real-time operating system, namely time-value functions, objects, and threads [Northcutt and Clark 1988], and (2) to validate these abstractions by applying them to one or more application domains. We selected two application domains, distributed multimedia and distributed command and control (C^2), with an emphasis on the former. We then refined the model, using an iterative approach in which we developed abstractions, applied the abstractions to the application scenarios, identified problems with the abstractions, and revised the abstractions.

When the abstractions were sufficiently complete, we implemented a simple proof-of-concept application that demonstrated the key concepts.

1.4 OUTLINE OF THE REPORT

In our first interim report [Downing and Davis 1992a], we proposed an initial set of abstractions with which user objectives, system resources and constraints, and characteristics of execution techniques (e.g., resources used and objectives satisfied) can be described. In the second interim report [Downing and Davis 1992b], we notionally applied the abstractions to a simple multimedia conferencing application. In the third interim report [Downing and Davis 1993], we revised the general abstractions for integrated control and described a sample architecture for integrated resource management. In this report, we summarize the results of the project, including the implementation of a simple demonstration program, and we make recommendations for future work. We do not attempt to repeat all the material covered in previous reports.

In Section 2, we present an overview of a model for integrated system resource management that serves as a framework for the discussions in the rest of the report, and we briefly describe sample application scenarios that are used as examples throughout the report. We next discuss the key abstractions and components of the model, including objectives and objective functions (Section 3), system resources and their metrics (Section 4), and characteristics of techniques for meeting objectives (Section 5). Next, we discuss our architecture for making resource management decisions, including decisions that involve issues of integrated system control (Section 6). In Section 7, we discuss an information model for the resource management system. In Section 8, we discuss the implementation of a simple video playback application that demonstrates some of the concepts developed during the project. In Section 9, we discuss some related work in resource scheduling and briefly describe other relevant work sponsored by Rome Laboratory and others.

Next, we summarize our conclusions and identify the areas of future research required to support SRM goals (Section 10). Finally, in Section 11 we provide a list of references and an annotated bibliography.

Appendix A discusses the relationship of system resource management to network management standards. Appendix B is a reprint of a paper on SRM that was presented 21–23 June 1994 at the Symposium on Command and Control Research and Development and Decision Aids in Monterey, California [Davis, Downing, and Lawrence 1994]. Appendix C contains a survey of work on low-level integrated control; the survey was part of a previous SRM report [Downing and Davis 1992b].

2 SRM MODEL AND SAMPLE SCENARIO

The goal of the SRM project is to develop an approach to allocating distributed system resources (such as CPU, disk, memory, and communication channels) to multiple competing activities so that the objectives (e.g., level of service preferences) of each of these activities may be satisfied to the highest degree possible. We have developed a set of abstractions with which user objectives, system resources and constraints, and characteristics of execution techniques (e.g., resources used and objectives satisfied) can be described, and have applied the abstractions to a simple multimedia conferencing application. (We also applied the abstractions to a distributed command and control application, as discussed in previous interim reports.) In the following subsections, we present an overview of our model for integrated system resource management, and we briefly describe the multimedia scenario.

2.1 OVERVIEW OF THE MODEL

Figure 1 is a high-level representation of the control loop for system resource management. Application objectives, such as desired levels of timeliness, precision, and correctness, are communicated to the decision-making components of the resource management system. The management system compares the requests of competing processes with the ability of system resources to satisfy those requests. Sample system resources include the local and remote processing, storage, and communication assets in a distributed system. In addition, the resource

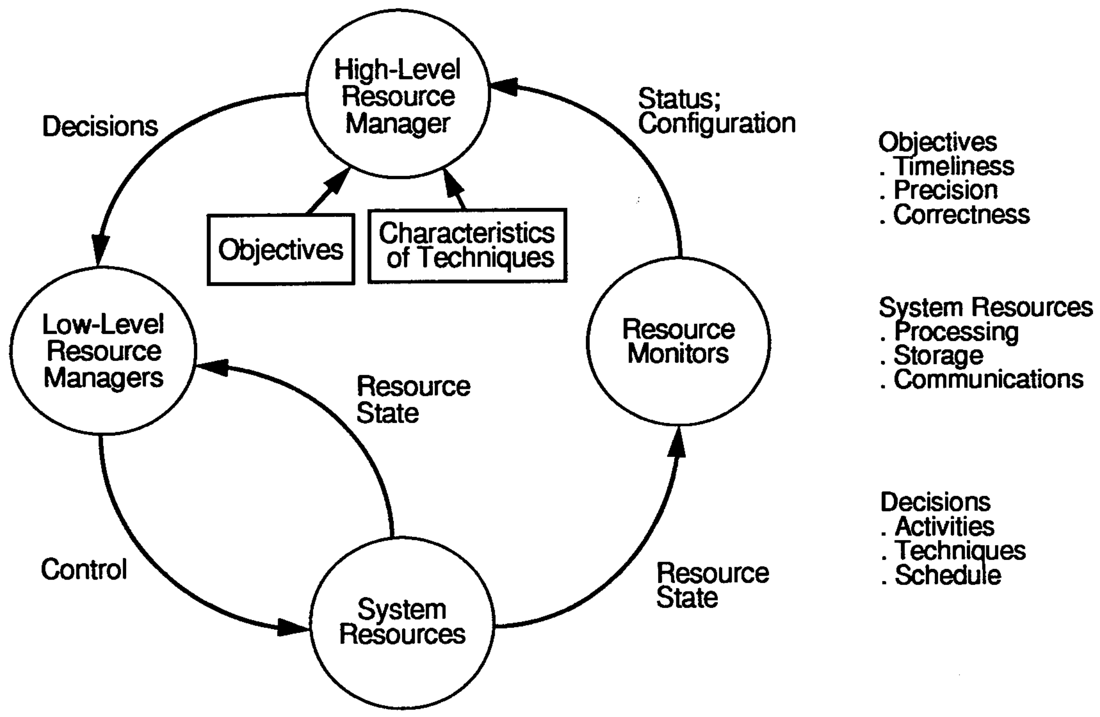


Figure 1. SRM Control Loop

management system considers the software configuration and other current status information to make a scheduling decision to queue, execute, or drop a computing activity. There may be multiple ways or techniques to satisfy a given objective. The resource management system weighs the characteristics of these techniques and selects the one that will produce the most desirable result.

To support adaptation, objectives such as precision and timeliness are specified using objective functions (also called benefit functions). Objective functions express the benefit to the user, or to the system as a whole, of achieving different levels of satisfaction of objectives, and can be used to determine appropriate tradeoffs among objectives. The objectives can be application specific (e.g., the benefit of different frame rates in a multimedia system) or system specific (e.g., the benefit of not exceeding various covert-channel bandwidths). The model assumes that execution techniques (e.g., various data-compression techniques) exist to produce results that fully or partially satisfy the objectives. There is a mapping between the techniques (and their parameters) and the degree to which they satisfy objectives. In addition, there is a mapping between the techniques and the amount of communication, storage, and processing resources they require.

Resource management decisions are made at two levels, based upon information about objectives and resources. Decisions involving medium- to long-range tradeoffs among activities and their objectives are made by a set of cooperating high-level decision makers. High-level decision makers base their decisions upon long-term resource usage statistics, external events, and domain-specific knowledge such as military doctrine or the observable effects of differences in multimedia quality. High-level decisions include the starting and stopping of periodic tasks; the setting of modes in commonly invoked objects (e.g., the correlation or weapon assignment objects of a C^2 application); and the admission control and resource allocation decisions for connection-oriented applications (e.g., a multimedia conferencing system).

Real-time scheduling decisions among competing tasks and threads (subtasks) within the selected activities are made by low-level schedulers for each resource. Each low-level scheduler uses status information about its resource, as well as process control abstractions passed down from the high-level decision makers, to make local decisions. While low-level resource scheduling is decentralized, integrated control (through shared resource status information and consistent interpretation of process control abstractions) is used to ensure that the schedulers complement each other's decisions.

2.2 OVERVIEW OF CONCEPTS AND ABSTRACTIONS IN MODEL

The following concepts and abstractions are used in our model. Many of them are based on abstractions used in the Alpha Operating System. Where our definitions differ significantly from those used in Alpha, we point out the differences.

Thread. A thread in our model is similar to an Alpha thread. It is defined as a locus of control that specifies the state of an execution; a thread can span physical nodes. Associated with a thread are objectives and other attributes that are used for system resource management.

Object. An object in our model is similar to an Alpha object or a server process. It is a passive entity that is similar to an abstract data type (a set of data plus the code used to manipulate the data). Any number of threads can execute concurrently within an object. The data within an object can be accessed only by invoking an *operation* on the object, using a mechanism similar to a remote procedure call. An object can provide multiple operations.

Objective. An objective specifies the desired degree of attainment of a given attribute such as timeliness or precision. Objectives are specified via extensions and generalizations of the Alpha time-value function, which we call *objective functions* or *benefit functions*. (The Alpha time-value function specifies both the *importance* and the *urgency* of producing a result by expressing the value to the system of producing a result as a function of when the result is produced.)

System Resource. A system resource is a hardware or software facility that enables a computation to occur. Examples of low-level system resources include processing, communication, and storage hardware. We also refer to higher-level system resources such as locks and semaphores. Access to system resources is controlled by resource manager objects.

Mode. A mode is a semistatic set of operating conditions for an object or set of objects. Modes are usually set during system initialization or reconfiguration: for example, the security level for an object may be determined by its mode.

Node. A node (or processing node) is a set of processors and memory storage units that are tightly enough connected that they can be considered a single entity. A single-processor or multiprocessor computer is considered a node. A local-area network consists of multiple nodes.

Activity. An activity is a unit of computation that produces a result. An activity may be the execution of an application, process, or thread.

We look upon a distributed system as a layered set of objects (see Figure 2), in which the objects in each layer make use of the services provided by objects in lower layers In the top layer are the application-level objects. In the middle layer are the system-level objects, which provide operating system services and interfaces needed by application-level objects. The bottom layer of objects manage the system resources (e.g., they provide access to resources and maintain information about the status of resources). Resource management objects have operations that are explicitly invoked by other objects, as well as operations that are invoked in response to events such as interrupts. Each layer shown in the figure can actually be several layers, and the functions of the layers are not strongly differentiated. In some cases, application and system objects can act as managers of high-level resources such as databases, files, and locks. For the purposes of our model, we do not differentiate between objects that execute inside the operating system kernel and those that execute outside.

The management of the system is done in several different time scales and at several different degrees of localization. During system initialization and during reconfiguration (as might occur in response to a significant loss of resources), gross decisions about resource allocation are made; these decisions can be made slowly and in advance of need, based on average measured or projected resource usages. Such high-level management decisions are global or are made on behalf of a large part of a distributed system. During steady-state execution of the system, scheduling decisions are made in real time, based on information about instantaneous availability of resources. Such low-level management decisions are localized to a single node and perhaps to a single

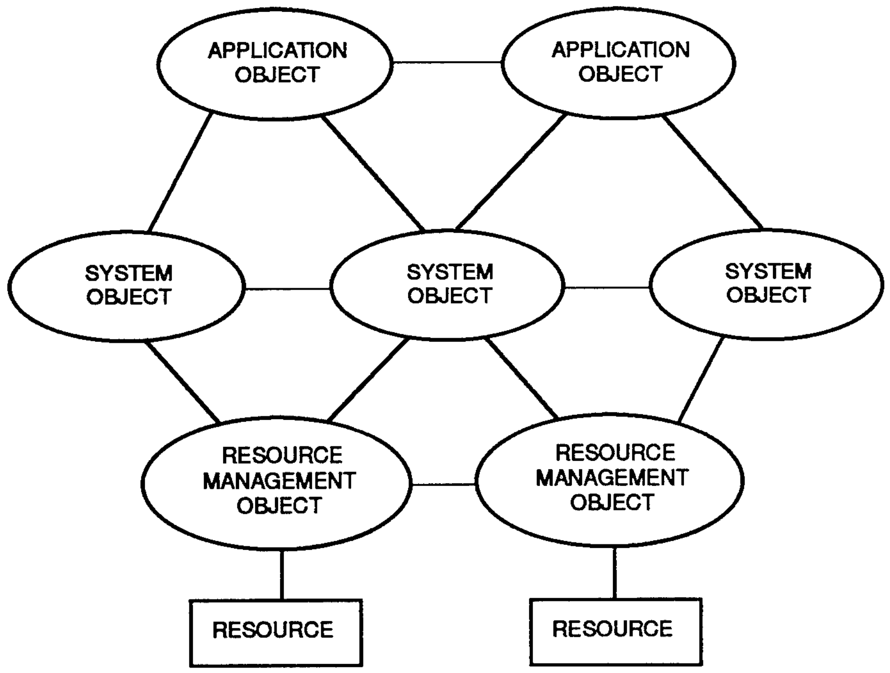


Figure 2. Layered Model of System

resource. The decisions required for adaptation occur at a rate somewhere between these extremes, use information about short-term-average availability of resources, and affect a subset of the distributed system.

The overall management of the system is done conceptually and in a layered manner. Managers at each layer use information about the objectives of the layers above and the resources provided in the layers below to make decisions about resource usage for their own layers.

2.3 INTEGRATED CONTROL

Integrated control is the process of using common abstractions and mechanisms in a consistent manner to manage different resources. Integrated control confers the following advantages:

- System design and management are simplified, because a small set of abstractions and mechanisms can be shared across the system.
- Resources are used efficiently, because a system with integrated control will use an available resource when the originally intended resource is busy.
- By using appropriate mechanisms that integrate control across a distributed system, it is possible to approach global optimization (maximizing the total utility of the work accomplished across a distributed system).

To achieve integrated control, the activities and the managers of system resources must cooperate in achieving common goals. A single manager may control which activities use its resource, but multiple managers must cooperate for activities to complete satisfactorily. Similarly, an activity can control which method to use to perform a function, but the activity needs to cooperate with the resource managers for the system as a whole to perform optimally.

We separate the problem of integrated control into two categories:

- Coordination of multiple activities among resource managers
- Achievement of objectives despite resource constraints within a single activity or closely related sets of activities.

The first category emphasizes integration across resources such as CPU, storage, and communication; integrated control across nodes in a distributed system can be considered a variation of this category. The second category emphasizes integration across objectives such as timeliness and precision. Together, these two categories represent an effort to achieve a system that best meets its objectives, given the available resources. We briefly discuss each of these categories in the next two subsections.

2.3.1 Integration of Multiple Activities

Multiple activities share the same set of resources. Figure 3 illustrates some of the resources used in common in any distributed system: CPU, memory, disk, and communication channels. These hardware resources operate asynchronously with respect to each other. Associated with these physical resources are resource managers that determine when and how activities use the resources. Some of the resources, such as the CPU and memory, are used implicitly by activities

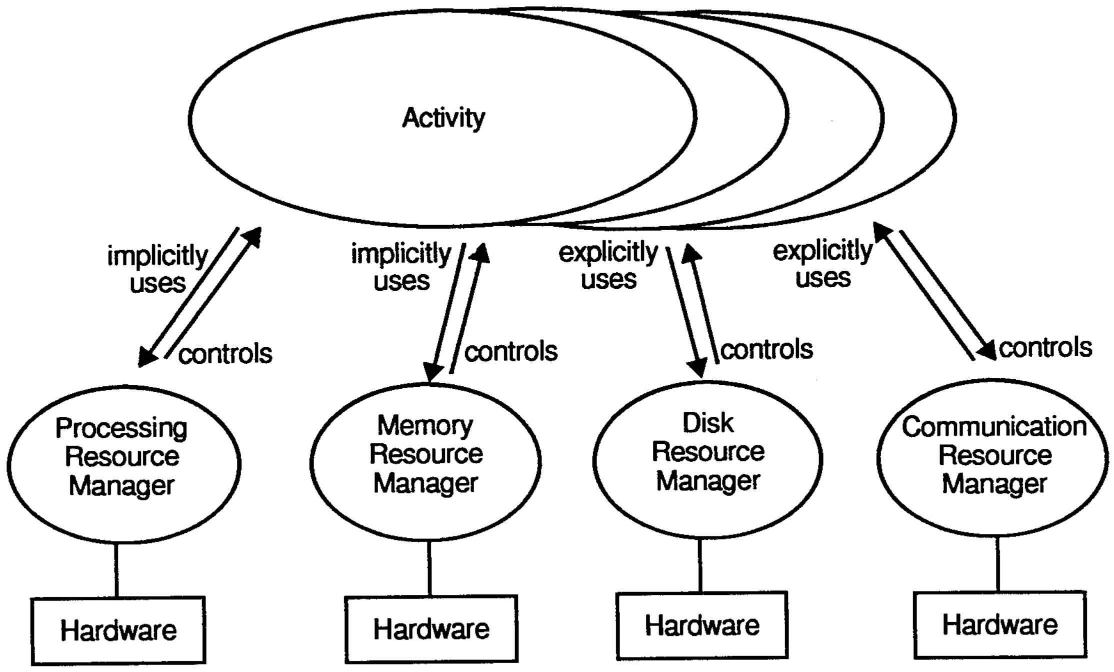


Figure 3. Activities Sharing the Same Set of Resources

that do not formally invoke the resource management objects. However, the resource managers exert direct control by deciding which activities get to use the CPU and memory. Other resource managers, such as the disk and communication managers, are explicitly requested by the activities to perform operations like reading or writing data. Explicit requests can be performed synchronously or asynchronously. Only in the case of synchronous requests, where an activity is blocked until the manager satisfies the request, does the disk or communication resource manager have direct control over an activity.

Therefore, the resource managers have partial control over the same set of activities and should work together to achieve their common objective (such as maximizing the performance of the highest priority activities, or making all hard deadlines, or maximizing the benefit for completing activities). To achieve the common objective, the resource managers must have a consistent view of what the system objective is, and have cooperating policies that manage the activities to best meet the objective. Furthermore, the managers must consistently interpret information about the activities that use them. What this information is depends on the system, but it can include priorities or time-benefit functions, expected computation time, and security levels. Since the physical resource managers are at the lowest layer of a distributed system, the amount of information they are required to understand should be minimized.

The physical resources are the lowest common denominator of all activities. However, there are higher-level system objects that exert indirect control over multiple activities. For example, locks and semaphores are shared logical resources that can block the progress of multiple activities. Physical resource managers must know the effects of using these logical resources in order to make effective decisions about how to allocate their resources. (For example, the processing resource manager uses information to avoid repeatedly scheduling an activity that is waiting for a lock instead of the activity that currently holds the lock).

In addition to providing control over a resource, the logical and physical resource management objects should also provide information about the resource (e.g., its load). Such information is required to provide integrated control within an activity.

2.3.2 Integrated Control Within an Activity

We now present a general description of a system model to provide integrated control within an activity. More detailed examples can be found in the first SRM interim report [Downing and Davis 1992a]. Figure 4 depicts a simple example showing the layered relationships among some higher-layer objects and some resource management objects that coordinate and manage access to disk and communications hardware. Object B explicitly uses the communication and disk resources; Object A uses Object B and explicitly uses the disk resource.

Threads entering each object carry (or have associated with them) objectives, specified in terms of objective functions. Each objective function expresses the benefit to the application of obtaining a result that achieves a quantifiable amount of an attribute. For example, a thread might specify a time-benefit function (the benefit as a function of the time the result was produced) and a precision-benefit function (the benefit as a function of the precision of the results), or might combine the two into a composite time-precision-benefit function.

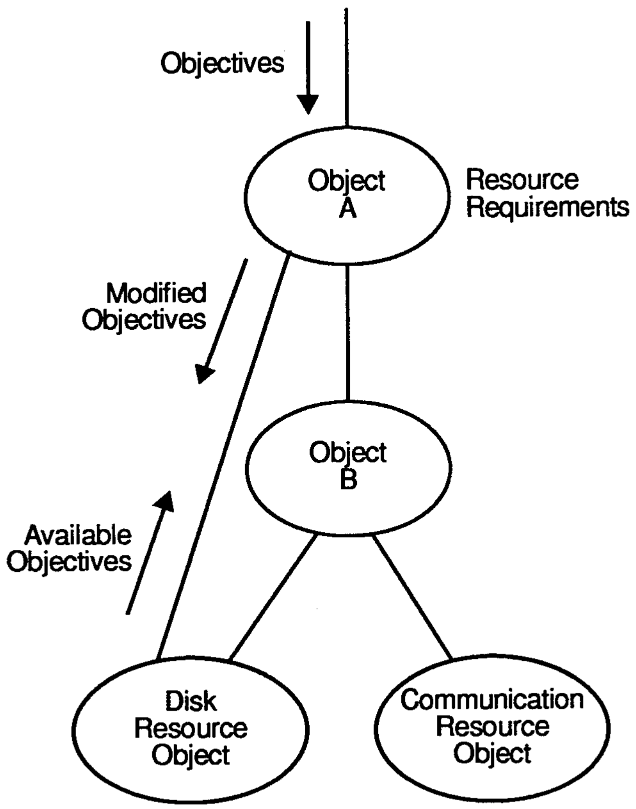


Figure 4. Example of Layered Relationships

Each object or resource manager also provides information about its currently available resources (e.g., the amount of its capacity that is not already reserved or in use, the length of queues, and so forth) to other objects that use it.

The objects (or parts of the resource management system acting on behalf of the objects) interpret the objective functions carried by the threads and estimate the resource requirements to meet the objectives. For each objective that an object understands how to interpret, the object contains (or can access) information that allows it to determine the amount of the resources required as a function of the value of the attribute associated with the objective. For example, an object that computes a mathematical function would have access to information regarding the processing time required to provide a given level of precision; an object that retrieves an image from a remote database would have access to information regarding the disk and communication time required to retrieve an image of a given resolution. Such information about resource usage, which is necessary if tradeoffs among objectives are to be made, might be determined analytically or measured experimentally. The information could be encoded within the object or be managed by a system resource manager, as appropriate.

If an object does not deal with a particular objective function, the object ignores the function and passes it through to other objects that the object invokes.

For each object, decisions must be made regarding tradeoffs among the objectives that the object deals with, and the values of some attributes must be fixed (i.e., some objectives must be "actualized"). For example, if an object can produce results of varying degrees of precision, a decision must be made regarding which degree of precision to produce. This decision is logically made within each object, but may actually be made by a system resource manager on behalf of the object. The information that is required for making the decision is (1) the objectives for the thread, (2) the resource requirements to meet the objectives, and (3) the resources available.

There are tradeoffs regarding the best layer in which to make management decisions. Decisions made at the lower layers, close to the resources that are being managed, are based on resource utilization information that is more complete and current. However, decisions made in higher-layer objects can take advantage of knowledge of the semantics of the objectives, can provide coordination among multiple lower-layer objects, and can provide more flexibility in carrying out decisions. In addition, making decisions at higher layers simplifies the algorithms in the lower layers.

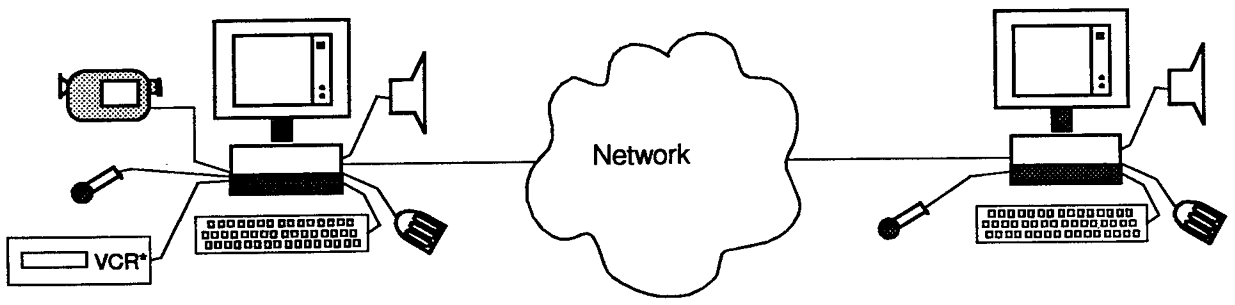
We have generally considered resource usage in terms of time (e.g., processing time, disk time, and communication time). It is possible to generalize resources to include, for example, considerations of space (e.g., memory, disk blocks, and communications buffers). In this case, the objective function passed to the resource would be a space-benefit function rather than a time-benefit function. The object would have to know the space requirements to produce a given objective, rather than only the time requirements.

2.4 MULTIMEDIA CONFERENCING APPLICATION

Multimedia applications can consume large amounts of system resources; any or all of the processing, communication, and storage resources may become bottlenecks. Many multimedia applications, such as conferencing systems, have timing preferences that are not necessarily in the form of hard deadlines. In addition, multimedia applications have quality-related objectives such as video resolution and audio/video synchronization that have large impacts on resource usage.

For these reasons, we have defined a simple multimedia conferencing application to provide a framework for discussion of SRM issues and to form the basis for the proof-of-concept implementation. The application enables a user at one node to present a briefing to a user at another node (see Figure 5). The media include real-time video of the presenter, real-time audio of the presenter's voice, images of maps and slides (possibly overlaid by a movable pointer), and prerecorded audio/video clips. Most of the information flow is unidirectional, but there is a limited amount of feedback from the recipient to the presenter (e.g., audio questions). We assume that the resources (CPU, communication, storage) are not necessarily sufficient to fully satisfy the recipient's performance and quality objectives. Much of the discussion in this paper will be presented in terms of this sample application.

Although the example application uses point-to-point communication, there is nothing in the model that precludes more complex applications that involve one-to-many or many-to-many communication.



*VCR: Videocassette Recorder

Figure 5. Simplified Multimedia Scenario

3 OBJECTIVES AND OBJECTIVE FUNCTIONS

Users of a distributed system have requirements and preferences regarding the resources that should be made available for accomplishing various tasks. For example, in a multimedia conferencing application, the recipient of a briefing has preferences regarding how the briefing information will be communicated and presented (e.g., the audio quality, the frame rate, and the image quality). These quality-of-service preferences, which we call *objectives*, must be represented in such a way that they can be used by the system resource manager for making tradeoffs among different tasks and execution strategies.

In the following subsections, we discuss the various kinds of objectives that are relevant in real-time distributed systems, issues related to categorizing objectives, how to specify objectives using objective functions, and examples of objective functions for the multimedia scenario.

3.1 OBJECTIVES

Objectives are application specific, so there is necessarily a wide range of objectives that might be appropriate in various situations. We believe that it is useful to divide objectives into a small number of classes that share common attributes. The following are definitions of some classes of objectives that are applicable for distributed real-time systems:

- Timeliness—a measure of how well the execution of a system meets time constraints
- Precision—a measure of the quantity and quality of work performed by a system (e.g., the number of decimal places of results in a computation, the resolution of an image, or the amount of data transferred)
- Correctness—a measure of the accuracy and consistency of the state of a system (e.g., the degree to which events are correctly ordered, the degree to which a value matches other values or external truth, or the absence of errors)
- Fault tolerance—a measure of how many faults a system can tolerate before it fails
- Security—a measure of how well protected a function is from disclosure or tampering.

The above list is not the only possible way to classify the objectives of interest to us; for example, we could replace correctness by two or three more narrowly defined classes, or could consider fault tolerance and security to be subcategories of correctness. Furthermore, the above classes are not completely orthogonal. For example, suppose an application processing a stream of sensor data needs to process four out of every five data items for the results to be meaningful, but the best results are achieved if all sensor data are processed. Also, suppose that each sensor data item must be processed within a certain time period. If a data item is processed late, there is clearly a failure of the timeliness objective; however, the precision and correctness objectives might fail as well.

Objectives may be defined at many levels of abstraction, including mission objectives, application objectives, and result objectives (see Figure 6). In the SRM model, objectives are associated with an application's results. All objectives are not relevant to all types of applications

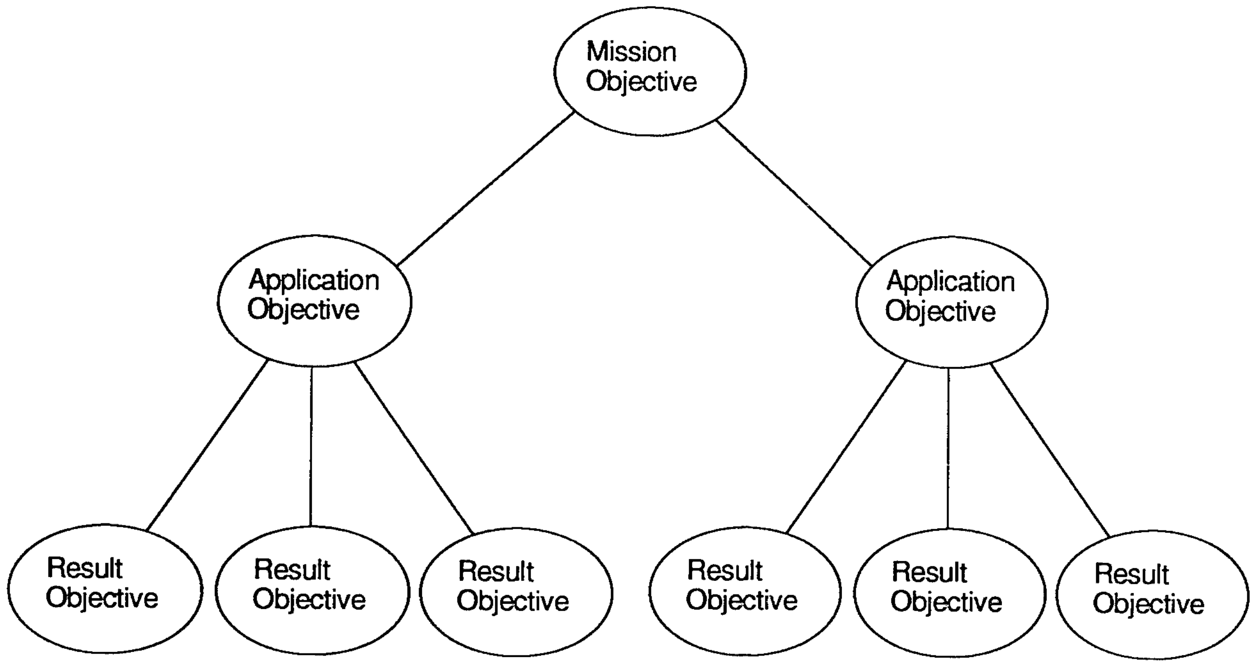


Figure 6. Objectives Defined at Different Levels of Abstraction

and results. For example, some applications do not have unique security objectives; they are willing to accept whatever security policy the system enforces. The security objectives may be applicable only to special management objects that make security-related decisions. An application can have many results and many ways of representing the results, and objectives can be associated with each one of these results.

As a hypothetical example, let us examine ways of assigning objectives to simulation results. The manner of displaying the data can be considered a result. The same numerical results can be presented to a user as a listing of values, as a simple bar graph, or as a series of 3-D images. These are different examples of a "display quality objective" of the numerical data. In addition, all of these displays can be presented in color or black and white, which could be considered an example of meeting a "display precision objective." Depending on the application, a decision to use black and white or color can indirectly affect performance, storage space, and communication bandwidth. In addition, the numerical data could be broken into pieces, each with its own time-benefit objective function. If performance becomes an issue, only parts of the data might be simulated in order to save time. The simulation results as a whole will have a precision objective, which specifies the number of runs versus the benefit (e.g., in terms of being statistically meaningful). This specification allows partial data to be returned to the user of the application instead of all or nothing. Finally, there must be a way to express the relationships among the different objectives, such as their relative benefits. This expression scheme allows the resource manager to determine how to allocate its limited resources and to avoid wasting resources, for example, by doing a high-quality presentation of low-quality results.

3.2 PROBLEMS WITH SPECIFYING ORTHOGONAL OBJECTIVES

Our informal definition of an objective is the degree of satisfaction of some criterion that an entity (e.g., an application, a user, or the system as a whole) finds desirable. In a complex, distributed, real-time application, there are many possible criteria upon which objectives can be based, some of which are closely related to each other. Examples of such criteria include performance, throughput, delay (timeliness), synchronization, ordering, precision, accuracy, correctness, consistency, functionality, availability, reliability, fault tolerance, survivability, security, and safety. We will not attempt to formally define these terms, but will use them in their generally accepted meanings, which will be sufficiently precise for this discussion.

In addition to these generic criteria, there are application-specific criteria that can be considered specializations of the generic criteria. For example, in an application involving the display of video information, image resolution would be a specialization of precision, and frame rate would be a specialization of throughput. Depending on one's point of view, frame rate could also be considered a specialization of precision, because a higher frame rate yields more information about the video sequence being displayed.

There are many possible ways in which the criteria and their related objectives might be categorized. For example, performance, throughput, delay, synchronization, and ordering are all related to time and therefore might be grouped together. Similarly, precision, accuracy, correctness, and consistency are all related to quality and therefore might be grouped together. Another possible grouping is functionality, availability, reliability, fault tolerance, and survivability, which are concerned with whether services can be accessed. Yet another possible grouping is security and safety, which are concerned with limiting interaction between a computing system and the outside world.

Unfortunately, many criteria can fit into more than one category. For example, it might make sense to group synchronization, ordering, and consistency, since they all deal with dependencies among two or more entities. As another example, throughput can sometimes be considered a quality criterion as well as a time-based criterion.

Other possible categorizations divide the objectives into groups such as time-based versus non-time-based, or deterministic versus probabilistic criteria. The group of probabilistic criteria might include availability, reliability, and fault tolerance; if security and consistency were defined in such a way that violations were occasionally allowed (e.g., a certain convert channel bandwidth), then these criteria might also fit into the probabilistic category.

There are many interrelationships among the criteria and their objectives. Often the attainment of one objective will implicitly assume the attainment of another. For example, availability implicitly assumes adequate timeliness and quality (e.g., correctness). Depending on the exact definitions, timeliness sometimes implicitly assumes quality. It is very difficult to devise definitions of criteria that are completely independent of other criteria.

Some criteria are intrinsically interdependent. For example, in the area of communications, the criteria of packet delay and loss at first appear to be independent; however, further thought reveals that loss can be considered infinite delay, and the two criteria are therefore interdependent. Similarly, a synchronization objective that requires in-order packet delivery will affect how well the delay and reliability objectives are met. In the case of video display, frame rate appears at first to be independent of resolution; however, the properties of human perception cause high resolution for moving objects to become less important as the frame rate increases. Thus, objectives based on these two criteria can be interdependent.

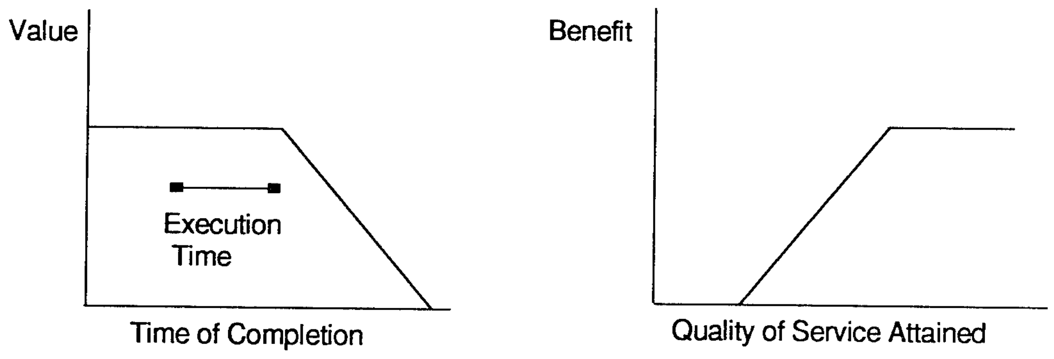
In other cases, criteria that are intrinsically independent become interdependent because of competition for the same resources; for example, increasing the precision may decrease the performance in a CPU-limited system. This is a situation in which the specifications are independent but the implementations are interdependent. As another example, the color of a table and the size of the table may seem to be independent, but if the amount of paint is limited, the color that can be implemented may depend on the size.

3.3 OBJECTIVE FUNCTIONS

An objective describes the importance to the system of producing a given result according to a given service specification. An objective has two aspects: (1) the degree or level to which some service is provided (e.g., the precision of a result), and (2) the relative benefit to the system of obtaining a given level of the service. The objective can be quantified by expressing it as a function relating the benefit to the level of service; for example, we could express a precision objective in terms of a precision-benefit function, where benefit = *f*(degree of precision).

This abstraction is similar to the time-value function used in the Alpha operating system, in which the value of a computation is related to the time at which it is completed. We extend and generalize the Alpha abstraction to allow the specification of arbitrary objectives, not just performance, and to include resources other than time—for example, memory or disk space. We generally use the term *benefit* rather than *value* when discussing objectives, since "value" is already used in many other contexts.

Figure 7(a) shows a graphical representation of a typical Alpha time-value function. The time-value (or time-benefit) function specifies the benefit to the system of completing a computation at various times. The time-value function expresses both the *importance* of completing a computation



(a) Alpha Time-Value Function **(b) Generic QOS-Benefit Function**

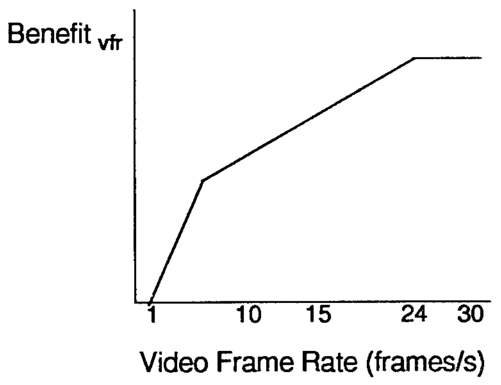
Figure 7. Time-Value Function and Generic Benefit Function

(as indicated by the value) and the *urgency* (as indicated by the time constraint). For reference, an indication of the expected execution time is also included, to allow a determination to be made of the time that a computation must be started in order to finish by a specified time.

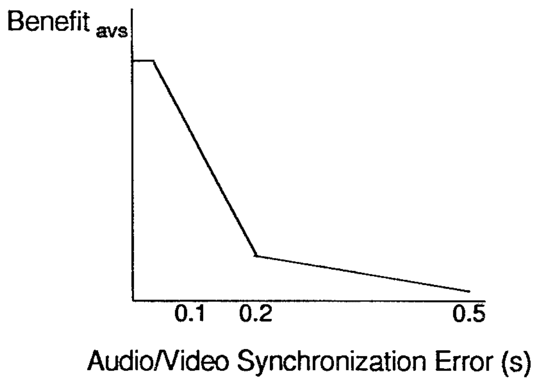
Figure 7(b) shows a generic quality-of-service (QOS)-benefit function (which we also refer to as a benefit function or an objective function). The generic benefit function specifies the benefit to a user or to the whole system as a function of the quality of service (for example, precision or correctness) attained during a computation. In order to fully specify an objective function, it is necessary to define a metric that quantifies the level at which a service is provided; it is useful if the metric is related to a quantity that can readily be measured or observed. For example, a metric for precision relating to a numerical calculation might be the number of decimal places in the result. It is also necessary to quantify the relative benefit of achieving different values of the level-of-service metric, e.g., the relative benefit of three decimal places of precision versus four decimal places. The assignment of relative benefits is subjective and highly application and mission dependent, and may require experimentation. For some applications, the benefit increases as the precision is increased; other applications might not be able to take advantage of more precise data (for example, may have a low-resolution display), and the benefit would remain constant no matter how much the precision increased.

In a multimedia conferencing application, the recipient of a briefing has objectives regarding how the briefing information will be communicated and presented. For example, the benefit to a user of a stream of audio/video information is a function of the video resolution, the image size, the video frame rate, the time delay between the generation and display of the information, the audio fidelity, the degree of audio/video synchronization, and so forth. The recipient's objectives include human factors considerations relating to assimilation of video and audio information. Ideally, the recipient would prefer the highest-quality audio and video for all aspects of the briefing. However, the ideal resources may be unavailable or unnecessary, due to limited resources, hardware capabilities, or different application requirements. For example, images of maps and slides can have a range of acceptable luminance and chrominance settings, with different resource requirements. For display of the presenter's face, the recipient may accept low frame rates, low resolution, and gray scale. However, a high frame rate, high resolution, and wide color range are desirable for prerecorded video clips of reconnaissance flights. The recipient may prefer as high a level of audio fidelity as possible, but may also be willing to accept telephone quality; may be able to tolerate several seconds of delay in the overall presentation, but may prefer that audio and video be synchronized to within a tenth of a second. Figure 8 shows two sample objective functions for multimedia streams. In Figure 8(a), the benefit increases as the frame rate increases, up to a plateau after which the user perceives no improvement. In Figure 8(b), the benefit decreases as the time difference between the corresponding audio and video streams increases.

The benefits associated with given objectives can vary with the mission of the system, and in response to conditions. For example, some objectives are different during combat and noncombat periods; the changes in these objectives can be triggered by a change in the operational mode of the system. Objectives can also change dynamically based on the state of the system; for example, as a missile gets closer, the importance of dealing with the missile increases relative to the importance of other activities.



(a) Objective Function for Video Frame



(b) Objective Function for Audio/Visual Synchronization

Figure 8. Sample Objective Functions for Multimedia

By quantifying the level of service for a given objective and then assigning relative benefits to various levels, we can construct an objective function for any objective we wish to define. The major complication is in relating the benefit values for one objective to the benefit values for another objective. Assuming that it is possible to develop methods to compare benefits across objectives, a resource manager can use generic objective functions to make trade-offs among objectives, without needing to understand the semantics of particular objectives. The use of generic objectives means that we do not have to define a fixed set of objectives that apply to all applications, but instead have the flexibility to define new objectives as appropriate.

When making control decisions, the various system management objectives must be considered together. One approach is to try to combine the various objectives for each activity into a single composite objective that can be plugged into the resource management algorithm. Such an approach is likely to be computationally intractable when several objectives are involved, except possibly when it is used to configure the system. Another approach is to fix certain of the objectives on the basis of requirements and the environment, and use the remaining objectives for scheduling decisions. For example, all the objectives other than timeliness might be fixed (by the selection of operating modes that provide appropriate levels of each) and the scheduling would then be done in such a way as to maximize the timeliness objective. The order in which the objective functions are evaluated and the objectives are fixed must be carefully chosen, since the order can affect the outcome.

3.4 LIMITATIONS OF OBJECTIVE FUNCTIONS

The purpose of an objective function is to be a general, expressive abstraction that can be used by a scheduler or decision maker to determine whether to schedule an application, when to schedule it, and how to execute it (e.g., how much execution time to allocate for variable-length applications). The objective functions are generalizations of time-value functions and cost functions that have proven to be well understood, intuitive to users, and usable by scheduling algorithms such as Best Effort Scheduling [Locke, Jensen, and Tokuda et al. 1985] and Cost-Based Scheduling [Peha and Tobagi 1991]. Unfortunately, objective functions also have limitations that may have to be addressed with additional abstractions.

Objectives that depend on the history of the system rather than just on the current state are difficult to attain by using objective functions as we have described them. Examples of realistic objectives in this category are (1) to have a covert channel below a certain bandwidth, or (2) to have a certain probability that a class of applications will be executed. Attempts to dynamically adjust the time-benefit functions on the basis of the current state of the system with respect to the objectives are likely to lead to instability; periods of time in which the objectives are met will alternate with periods in which the objectives are not met. For example, suppose that there is an objective to process at least 78% of all sensor input. Adjusting time-benefit functions (for example, by increasing the importance of processing a given sensor input when the recent average success rate is less than 78% and decreasing the importance when the recent rate is greater than 78%) may cause a pattern such as 78 successes followed by 22 failures. What is really desired is that each sensor input has a 78% chance of being processed regardless of the fate of the previous sensor inputs. This underlying aim implies the existence of some sort of "distribution objective" that has not been explicitly expressed.

Another problem with objective-benefit functions is the difficulty in quantifying the benefit of satisfying an objective. Not everything can be quantified as easily as a manufacturing environment where benefits can be expressed in dollars. For example, the long-term benefit of a military objective like fault tolerance is hard to quantify. Also, how well a technique satisfies objectives is difficult to understand. For example, a replicated file can either improve performance or hurt performance, depending on how it is used in the long run. In addition, a replicated file can improve read-availability; but it can also hurt or help write-availability, depending on the technique used to maintain consistency.

4 SYSTEM RESOURCES

All complex distributed systems have components for processing, communication, and storage. Applications compete for the use of these components through the use of management software such as device drivers, schedulers, databases, file systems, and communication protocols. This management software is a logical extension to hardware components such as CPU, disks, and communication transmitters and channels. It is worthwhile to distinguish between the management software and the hardware components, since it is only at the hardware/firmware level that orthogonality and independence of the functions truly exist. For example, although a distributed file system can be considered an extension of the storage component, it competes with other software for the use of both the processing and the communication components.

In the following subsections, we take a bottom-up approach to defining system resources: we first describe the primitive hardware components, including the processing, communications, and storage components common to all systems. Next, we discuss the management software that extends the concepts of processing, communication, and storage. We also suggest an approach to management software called "stackable layers" that seems especially appropriate to the SRM goals. Finally, we discuss the software to monitor and distribute the status of system resources—a function that is critical for control integration.

4.1 PRIMITIVE HARDWARE COMPONENTS

In this report, when we refer to the processing, communications, and storage components, we are typically referring to the primitive hardware components of the system. Furthermore, we are discussing generic hardware components, and not specific implementations. For example, our discussions on the hardware components should not be confused with the subsystems in Alpha Release 1* [Northcutt et al. 1988]. Alpha Release 1 had a separate processor dedicated to the functions of processing, communication, storage, and scheduling; this processor performed both high- and low-level management functions; for example, the communication subsystem executed protocols for communication, thread maintenance, transactions, and replication. While such an approach maintains the orthogonality of the management software for each hardware component, it is not commonly used.

The processing hardware component consists of a CPU and its associated firmware. A multiprocessor can have multiple hardware components. Associated with the processing components are attributes such as MIPS (millions of instructions per second) and MFLOPS (millions of floating point operations per second). The major resource that can be consumed by users of the hardware component is time. The processing component is of special interest in that it is often the main cause of contention; all management software and application software requires processing time. In the past, the processing component has typically been the system bottleneck, although the speed of the processing components has been improving faster than that of the storage or communication components.

*All product names mentioned in this document are the trademarks of their respective holders.

The communication hardware component consists of the physical transmitters and receivers and the communication channel. A single computer can have multiple communication components. Attributes associated with the communication component include bandwidth and latency. The major resources consumed by users of the communication component are time and buffer space. The communication component differs from the other components in that a computer's communication management software (e.g., a device driver) may not have complete control over the resource. For example, contention for an Ethernet can result from simultaneous transmissions from multiple sources.

The storage hardware component consists of disks and disk controllers as well as read-only memory (both shared and distributed in the case of a multiprocessor) and memory caches. The storage component's attributes include disk and memory capacity (percentages used, reserved, and available) and the speed of performing seeks, reads, and writes. Users of the storage component consume the time resource, but they can also consume disk space. This project emphasized the time resource, since time is consumed by all of the processor, communication, and storage components.

The processing, communication, and storage components, as defined above, are the main hardware resources that SRM is concerned about. However, these are not the only primitive hardware components. For example, the Silicon Graphics, Inc. (SGI) VGX hardware supports 65,000 test-and-set variables that can be used by management software for locks and synchronization.

4.2 MANAGEMENT OBJECTS AND STACKABLE LAYERS

In this section, we define and discuss the role of management objects. We also discuss the concept of stackable layers, which can exemplify a set of management objects as well as a model to help support the meeting of objectives. Finally, we discuss the resources consumed by these objects.

The primitive hardware components discussed above cannot function without management software, which can be considered as logical extensions of the hardware components. For example, although a distributed database system uses the processing, communication, and storage components, it can also be considered a high-level system function that can be dealt with as a single entity by an application. Such high-level functions would be implemented as management objects in the system.

Management objects are typically hierarchical or layered, where the higher layer extends the functionality of the lower layer. For example, in BSD UNIX systems, interprocess communication is layered: a device driver is the lowest layer, the protocol layer (which may itself have multiple layers as with TCP/IP) comes next, and the socket interface is the top layer, used by all applications to communicate. A programmer deals with the socket interface as a single entity, even though it maps to multiple lower-level protocols. In our model, the device driver, each protocol, and the socket interface would be separate management objects that can be hierarchically layered.

More recently, the concept of "stackable layers" has emerged from the communication and distributed system research communities as a way to define system architectures. Stackable layers allow the same operations or methods to be used for lower-level communication and storage management objects as well as for higher-level management objects. Such a design enables functionality to be added transparently by inserting new layers. Stackable layers are often implemented via object-oriented techniques, and have been used to describe communication protocols and file systems [Peterson et al. 1990; Popek et al. 1990].

Figure 9 illustrates how a stackable layer architecture has been used for a distributed file system, one of the most important hierarchies of storage management objects. Each of the stackable layers of the Virtual File System (VFS), the Network File System (NFS), Ficus, transactions, and the UNIX File System (UFS) can be considered a storage management object that performs a particular function that remains transparent to the user of the storage resource. VFS defines a general object-oriented-like interface for accessing a UNIX file system, including operations such as *read, write, open*, and *close*. Each lower layer supports the same interface as VFS. NFS is logically below VFS and provides access to files located across the network. Ficus, which actually consists of three layers, provides weak-consistency replication of file system volumes over a wide-area network. The transaction layer is a planned layer that supports atomicity and other features. UFS, or a similar file system, is the lowest layer. If a function is not used or supported by one of the layers, calls to that function are merely passed through to the next layer.

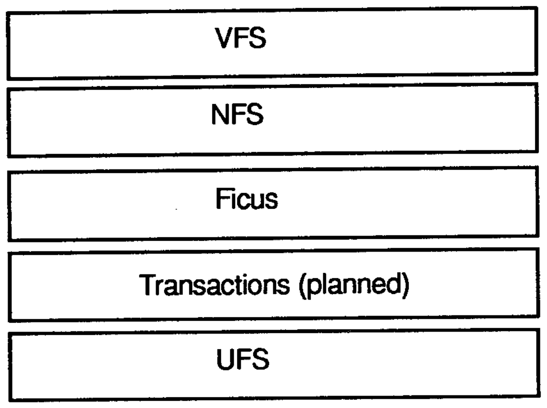


Figure 9. Stackable Layers of a Distributed File System

Stackable layers of management objects can play an important role in the SRM model. Each management layer can be used to satisfy one or more of the application objectives. In addition, multiple layers can support different techniques to support the same objective. For example, an objective that a file be available can be supported by replicating the file via Ficus, if the operations on this file need not be synchronized across the network; if synchronization is required, however, a management layer could be added for that purpose without modification to the other layers. In the SRM model, a thread would carry its availability objective through the management layers until one of the layers could satisfy it.

Management objects that satisfy specific objectives can also be created with stackable layers. For example, suppose that MLS security is not required, but some level of security is desired: an encryption layer could be added to satisfy this objective. Another example is a compression layer that can be used for resource management to reduce the load on the communication and storage components.

4.3 MONITORING OF SYSTEM STATE

System state information is used for (1) making admission control and reservation decisions, (2) scheduling and dropping activities, (3) selecting system and object modes and execution techniques, (4) load-balancing tasks across a network, and (5) tuning and optimizing applications. System state information includes configuration information, resource attributes, and the current status of the resources. The characteristics of the system state information include the following:

- **Time Properties**. The data can refer a single instant, can include time sequences of historical data, or can consist of time averages and other statistical aggregations. In a distributed system in which there are significant time delays in distributing resource status data, instantaneous data is of little use.
- **Level of Detail**. The data can contain details or can be summaries of lower-level data. Typically, in a layered or hierarchical system, resource monitors will collect detailed data from the resources being monitored, but will pass summaries to higher layers.
- **Level of Abstraction**. The data can be at various levels of abstraction, from the status of low-level system resources to information about high-level application abstractions.
- **Dynamic Properties**. The data can vary slowly, as in the case of configuration information, or rapidly, as in the case of scheduling queues.

Configuration information is static or varies slowly. Sample types of configuration information are

- The number and type of machines
- The network topology
- Where objects or other system resources are located in the network
- The number of copies of a particular resource, such as a shared file.

Associated with the configuration are the attributes of system resources; this information is static and includes

- The total size of the disk, memory, or buffer pools
- The speed of the CPU, disk, or communication channel
- The latency of a communications channel.

The current status of system resources is relatively dynamic and includes such information as

- The up/down status of resources
- Loads and queue lengths
- The degree to which objectives are being met
- External conditions (e.g., normal operation vs. alert)
- Hardware and software failures and exceptions.

The status information must be maintained appropriately. Decisions must be made as to the level of detail is appropriate for each type of status information, and how to obtain the information. Information can be distributed by contacting an agent or set of agents (e.g., monitors) or by passively piggy-backing information on threads. If monitors are used, they will have to exchange information so that local information and some remote status information are available at each node. Status information can be obtained by having a monitor contacted by a management object when an event occurs, by having the management object regularly update a monitor, or by having a monitor actively poll management objects. If the monitor is regularly updated, then a pulling or pushing interval must be selected that is appropriate for the resource being monitored and the algorithms using the status information. Much research has already been done on monitoring [Craig 1991; De Boor 1990; Zhao and Ramamritham 1985]. In Appendix B, we discuss the use of open-systems-based management standards to collect status information.

5 CHARACTERISTICS OF EXECUTION TECHNIQUES

A key capability of SRM is the ability to switch among alternative execution techniques in response to changing conditions. In the following subsections, we discuss the kinds of information that must be known about the execution techniques for this adaptation to occur; and we discuss some representative techniques that are useful for the multimedia scenario.

5.1 EXECUTION TECHNIQUES

The resource management system, as embodied in a set of resource management objects, may have one or more ways of satisfying an objective. Therefore, it is necessary to select which technique or set of techniques should be used to best match the objective. The characteristics of the techniques must be known, including (1) the degree to which an objective can be met, (2) the type and level of resources consumed by the technique, and (3) the dependencies among the techniques. If the degree of objective satisfaction and the resource usage characteristics are known, the benefits of the technique can be weighed against the resource availability. If the dependencies are known, the transitions among techniques can be evaluated.

For example, in a distributed multimedia presentation scenario, tradeoffs must be made and appropriate execution techniques must be selected, based on the objectives expressed by the recipient (and, to a lesser extent, by the presenter), and on the availability of system resources. By using appropriate compression techniques, it may be possible to reduce the use of communication resources, although at the expense of using more CPU resources (assuming no special hardware support). Using lower frame rates and lower video resolutions may free CPU and communication resources to provide higher-quality audio. Using additional memory and disk buffers may improve audio/video synchronization while increasing delays. The characteristics of the various execution techniques (in terms of the resources used and their effect on objectives satisfied) must be represented in such a way that the system resource manager can use them when selecting effective execution plans.

Although a resource management object can be considered an extension of the hardware component, it does not necessarily consume resources only on the component it manages; for example, a communication device driver uses processing time as well as memory buffers. For the resource management system to determine how to meet an objective, it needs to know the characteristics of the objects in terms of the resources they consume. For example, the SRM system needs to know the amount of time required by an object to execute an activity, in order to determine whether the deadline specified by the application can be met. Similarly, the SRM system needs to know the precision, fault tolerance, and security characteristics of the objects, as well as the amount of resources (processor, communication, and storage time and possibly space) required to perform various tasks. These characteristics are most reasonably associated with the objects or their operations (methods).

Typically, real-time systems consider only the overall expected execution time of the units of computation being scheduled. However, if the SRM system is to provide the potential for load balancing across the hardware components, it is necessary to decompose the task execution time for each object into a finer level of granularity, possibly including information like

- The time spent within the different components
- Where and when the different components are used during execution of the object
- The shared resources used, such as particular locks.

A scheduler could then use this information to make better scheduling decisions. For example, the scheduler could combine configuration information about where objects are located with resource use information about the expected sequence of objects to be used by a thread (see Figure 10) in order to balance loads across components and nodes. Such an approach could also help avoid the "false overload" problem experienced in Alpha. Unfortunately, the complexity of scheduling to such a low level of detail is likely to be prohibitively expensive; and the necessary computations may be infeasible in real time. Instead, the scheduler could use higher-level abstractions, such as the expected percent of time spent in each component, to make decisions.

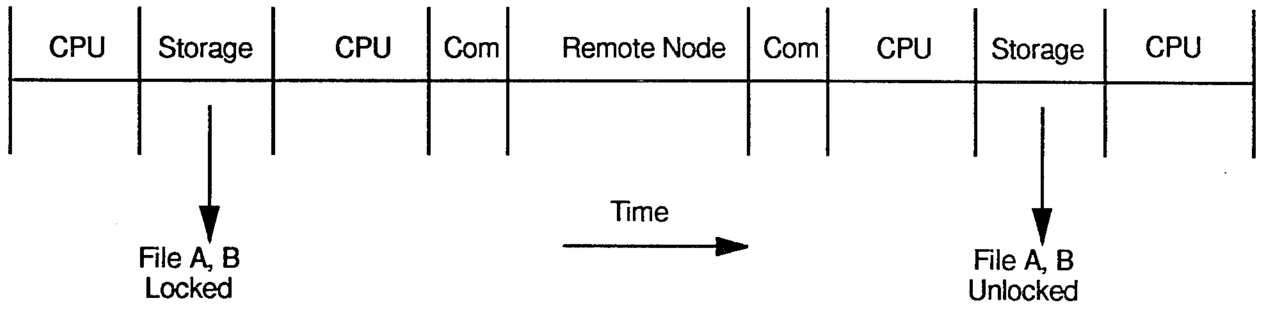


Figure 10. Task Execution Times of Resource Usage

In addition to decomposed task execution times, the ability of the objects to switch techniques in order to better satisfy application objectives must also be known. While decisions involving memoryless techniques, such as different Fast Fourier Transform (FFT) algorithms on a set of input data, can be made on a thread-by-thread basis, decisions involving other techniques, such as concurrency control techniques, cannot be made on a per-thread basis because these decisions involve multiple threads. Switching among the memory-using techniques is modal, and is slower than switching memoryless techniques. Furthermore, the modal techniques may have rules involving how to shift between techniques. For example, Figure 11 shows sample relationships

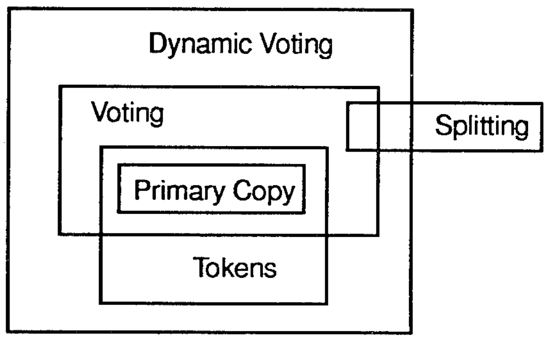


Figure 11. Sample Relationships among Mutual Consistency Methods

among mutual consistency techniques for replicated data. It is possible to transition among these techniques, but it may be desirable for implementation reasons for each of the techniques to switch only to and from the lowest common denominator (the primary copy of the data) [Bhargava and Riedl 1988]. Therefore, it would not be possible to switch directly from a token-based scheme to dynamic voting. Such interdependencies must be known in advance.

5.2 EXAMPLE TECHNIQUES FOR MULTIMEDIA

We have investigated various execution techniques for multimedia. These techniques can produce results that satisfy the objectives related to multimedia conferencing to varying degrees, using varying resources. For example, there are several categories of compression and encoding algorithms, including frequency-, prediction-, and importance-oriented categories. Within each category are several related techniques, such as subband or transform coding in the frequency-oriented category. Which technique to select in order to maximize the benefit depends on the available resources (possibly including special-purpose hardware) and the user objectives (e.g., the loss of information tolerated). We have also identified other techniques for selecting and maintaining frame rates, improving audio/video synchronization, reducing jitter, and supporting varying degrees of error correction.

Several compression and encoding techniques for audio and video cause the data to be conceptually separated into layers, such that the lowest layer provides at least the minimal acceptable quality (e.g., resolution) and each successive layer improves upon the resolution of the lower ones. Typically, the higher layers depend on the lower ones, and are worthless without them. This encoding scheme is known as hierarchical encoding [Shacham 1992]. Hierarchical encoding allows a system to adapt quickly to varying environments; if it is not possible to process all the data, the system can simply drop the data in the highest layers. In a resource-rich environment, all layers are processed and high-resolution images are displayed; whereas in a resource-poor environment, only the lowest layers are processed and low- resolution images are displayed.

We have described these and other techniques in a previous SRM interim technical report [Downing and Davis 1992b].

6 DECISION MAKING

As discussed above in Subsection 2.4, a two-level approach to resource management and scheduling is appropriate for the multimedia conferencing application. A set of cooperating high-level decision makers make decisions involving medium- to long-range trade-offs among activities and their objectives, and select the activities that will be permitted to run. Decentralized low-level schedulers associated with each resource make real-time scheduling decisions among competing threads within the selected activities. A conceptual architecture for decision making is shown in Figure 12 and is discussed in the following subsections.

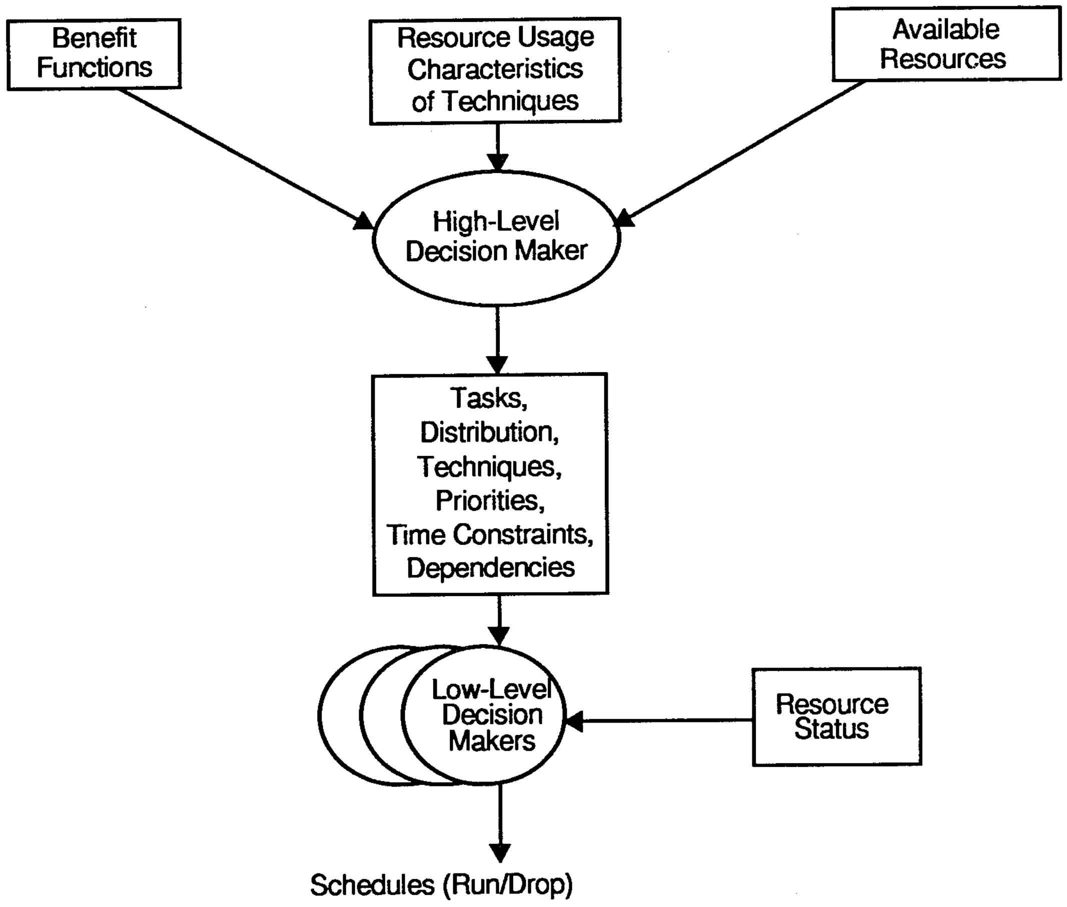

Benefit Functions
Resource Usage Characteristics of Techniques
Available Resources
High-Level Decision Maker
Tasks, Distribution, Techniques, Priorities, Time Constraints, Dependencies
Low-Level Decision Makers
Resource Status
Schedules (Run/Drop)

Figure 12. Conceptual Architecture for Decision-Making

6.1 HIGH-LEVEL DECISION MAKER

The high-level decision maker considers user objective functions, resource constraints, and characteristics (e.g., resource usage and the degree to which they meet objectives) of candidate techniques, including the resources required to transition between techniques, and system status information. On the basis of this information, the decision maker chooses which activities (e.g., which multimedia conferences or periodic tasks) to execute; the nominal amount of resources to devote to each activity; and which techniques and parameters to use when executing these activities, such that the objectives can be met to the optimal degree consistent with satisfaction of the resource constraints. The decision maker then uses these decisions as the basis for generating the information that is required by the low-level real-time schedulers: (1) the expected job lengths (execution times) for the threads within each activity, based on the selected techniques and parameters; and (2) the time-benefit functions for the threads, based partly on a composition of higher-level objective functions.

The high-level decision maker monitors the resource usage and the level of objectives that are being met. If the measured values differ significantly from the goal values previously specified by the decision maker, the decision maker takes compensating actions such as revising resource allocations, techniques, and parameters.

Given the combinatorial growth of the high-level scheduling problem (due to the number of alternative techniques, parameters, resources, activities, and so forth), the use of exhaustive or even statistical techniques is not feasible in real-time and dynamic environments. Various AI techniques have proven their ability to reduce the dimensions of large search spaces such as occur in the SRM problem domain. The SRM decision maker may use different AI techniques for initial scheduling and revised scheduling: namely, knowledge-based scheduling (which uses goals and constraints) and rule-based expert systems, respectively. Specifically, we are investigating the capacity scheduling technique [Sycara et al. 1991], which has been proven successful in resource-constrained scheduling. Note that while the extensively adaptable application described here may use such techniques, other applications with fewer modes of adaptability could use simple decision makers based on tables generated from simulations or experimentation.

6.2 LOW-LEVEL DECISION MAKER

Decentralized low-level resource schedulers are responsible for scheduling the threads within each activity (e.g., a thread that processes a frame within a briefing activity). Low-level scheduling decisions must have relatively low overhead and must be made quickly. The resource schedulers are associated with physical resources such as processors, disks, memory buffers, and communication channels, as well as with logical resources such as databases. Although decentralized, resource schedulers use integrated policies when determining their schedules.

The low-level schedulers use resource status information, control abstractions provided by the high-level decision makers (time-benefit functions and job-length functions), and additional information (such as dependencies) that is maintained by the application. Sample scheduling algorithms that use information similar to this include Locke's Best Effort Scheduler (LBES) [Locke, Jensen, and Tokuda 1985], Cost-Based Scheduling (CBS) [Peha and Tobagi 1991], and Dependent Activity Scheduling (DAS) [Clark 1990]. To adapt to short-term congestion, the low-level schedulers can drop the least important layers of hierarchically encoded multimedia information, enabling the multimedia application to gracefully degrade quality.

Integrated control among low-level schedulers requires that the schedulers consistently interpret abstractions (e.g., time-benefit functions, execution lengths, dependencies, and security levels) associated with the threads. The schedulers do not necessarily have to use identical policies, but they must work toward a common goal (unlike, for example, systems with processor schedulers that minimize missed deadlines, but disk and communication schedulers that maximize throughput.)

7 INFORMATION MODEL

An information structure diagram (based on the entity relationship diagram) models the entities in the resource management system and the relationships among them. Such a tool is useful in organizing a complex system and helping the designer understand it and reason about it.

Figure 13 shows a preliminary version of an information structure diagram for SRM based upon the requirements of the multimedia scenario. Boxes represent entities, and arcs (lines) represent relationships between entities. The arrowheads indicate the multiplicity (or cardinality) of the relationships, with a single arrowhead indicating that one instance of an entity is associated with each instance of the relationship, and a double arrowhead indicating that many instances of an entity are associated with each instance of the relationship. A verb describing the relationship is placed at one end of an arrow, and describes the relationship from the point of view of the entity at the other end of an arrow. Generally, the sense of the verb can be inverted, and the inverted verb can be placed on the opposite arrowhead. For example, the leftmost relationship in the figure can be read as "an application contains many threads" or "many threads are contained in an application."

Another type of relationship is also shown in the figure, namely supertype/subtype or "is a." This relationship is indicated by arcs without arrowheads but with a short perpendicular line on the end of the arc closest to the supertype entity. For example, in the lower right corner of the figure, "processing," "communications," and "storage" are subtypes of "resource"; or we can say that processing, communications, and storage are resources.

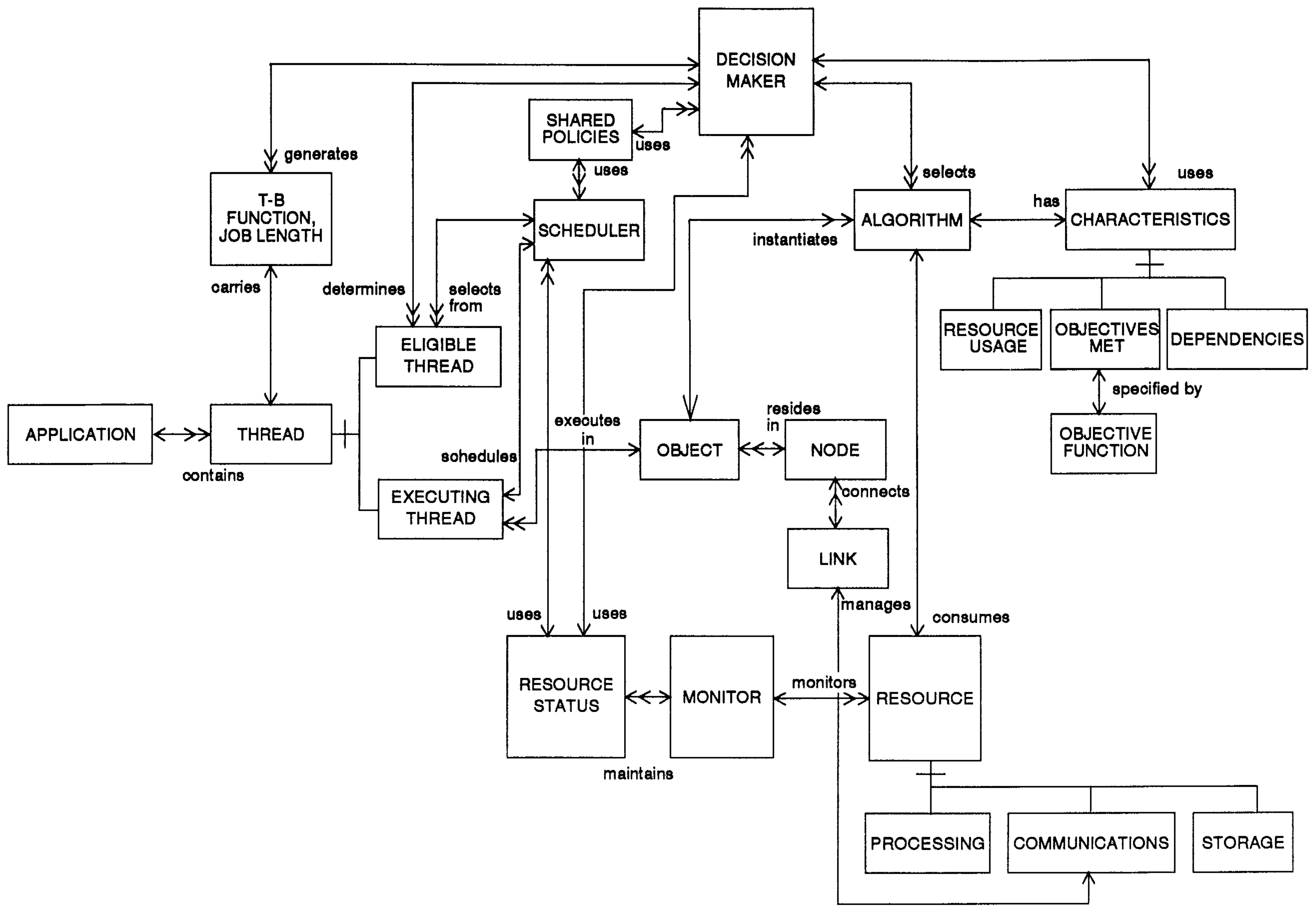


Figure 13. Preliminary Information Structure Diagram for System Resource Management

8 PROOF-OF-CONCEPT IMPLEMENTATION

We implemented a simple remote video presentation application in which video from a video camera or video cassette recorder is captured and digitized at a computer workstation, transferred across a local-area network to a destination workstation, and displayed on the monitor of the destination workstation. The key concept demonstrated is the adaptation of the distributed application to limited communication resources, according to preferences specified by the user.

The application provides the following functionality:

- A computer workstation with appropriate video hardware captures full-motion video from a video camera or video cassette recorder, digitizes the video frames, and compresses them, using the Joint Photographic Experts Group (JPEG) image-compression algorithm.
- The stream of digitized video frames is sent via a local-area network (an Ethernet) to a destination workstation, using standard communication protocols (TCP/IP).
- The destination workstation reads the digital video stream from the local-area network, uncompresses the frames, and displays them in full motion on its color monitor.
- Via a graphical user interface on the destination workstation, the user can connect to the source workstation, start and stop the video playback process, and exit the program. The user can also specify objective (benefit) functions that express his or her preferences regarding frame rate, frame size, and quality (Q) factor (the latter controls the lossiness of the compression and therefore the quality of the images).
- The application monitors the amount of video data transferred; on the basis of the observed data transfer rate, the application determines the combination of objective function parameters (frame rate, frame size, and Q factor) that maximizes the total benefit, and controls the video capture, compression, and transfer process accordingly.
- The user interface continuously displays running averages of the frame rate, bandwidth usage, and Q factor that are attained by the application.

The following hardware and software are required to execute the demonstration application:

- Two Sun Microsystems, Inc. (Sun) SPARC workstations with color monitors, connected via a local-area network
- SunOS (UNIX) operating system, version 4.1.3
- Sun OpenWindows, version 3.0, with extensions to support Parallax hardware
- Motif graphical user interface, version 1.1.4 (required for compilation but not for execution)

- A video camera or VCR with NTSC* video output
- A Parallax Graphics, Inc. (Parallax) XVideo video board and XVideo Toolkit software.

The logical architecture of the SRM multimedia demonstration is shown in Figure 14. The application consists logically of three parts: a sending process (executing on the source workstation), a receiving process (executing on the destination workstation), and a controlling process (executing on an arbitrary workstation, but typically on the destination workstation).

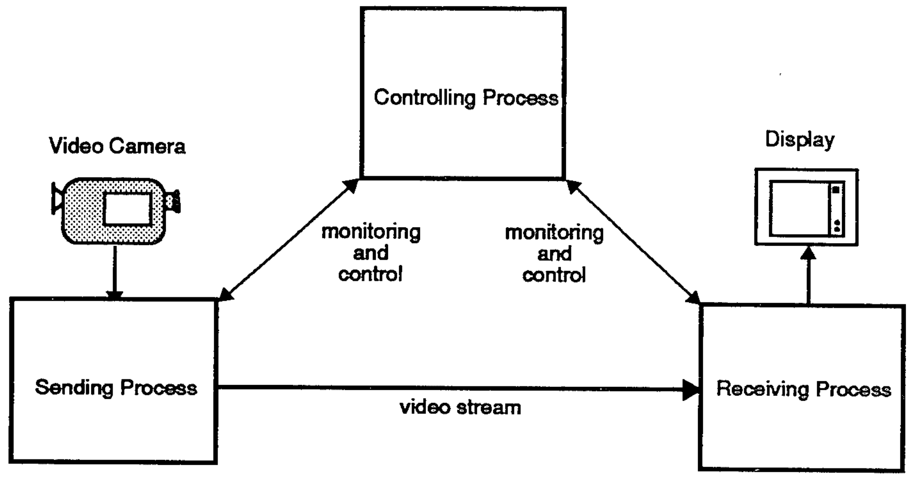


Figure 14. Logical Architecture of SRM Multimedia Demonstration

A more detailed view of the architecture used for the implementation is shown in Figure 15. An OpenWindows server process on the source workstation manages interactions with the video camera (via a Parallax board) and acts as the sending process. The functions of the controlling and receiving processes are combined in a process that executes on the destination workstation. An OpenWindows process on the destination workstation manages interactions with the display (via a Parallax board).

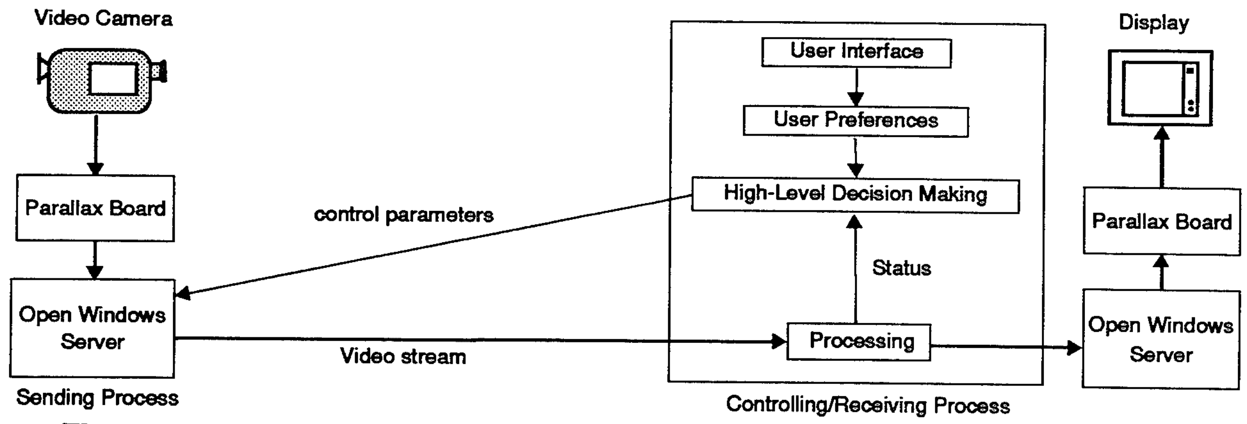


Figure 15. Implementation Architecture of SRM Multimedia Demonstration

*NTSC: National Television Standard Committee.

A video camera or VCR generates an analog video signal that is digitized and compressed by a Parallax board in the source workstation. The sending process and the Parallax board perform various operations on the digital video stream, such as modifying the frame rate, frame size, or Q factor. The resulting digital video stream is transmitted over the network to the destination workstation via TCP/IP. Additional processing is optionally done at the destination, and the video stream is then passed to a Parallax board, which uncompresses each frame and displays it in a window.

The controlling process has a user interface that allows the user to specify frame rate, frame size, and Q factor preferences, and to control operations such as starting and stopping the video capture and display. A high-level decision maker combines user preference information and status information from the receiving process to determine the optimal values for the frame rate, frame size, and Q factor. The selected values are communicated to the sending process, where they affect the operations performed by the Parallax board.

The key algorithm used in the application is the one used by the high-level decision maker; this algorithm is described in pseudocode below.

```
LOOP:
    determine which parameter can be decreased with the least loss of benefit;
    if image quality reduction is best then
        reduce the image quality (by changing the Q factor);
        run with the new Q factor, collect statistics, and calculate the resulting bandwidth;
        if satisfactory then
            go to DONE(finished);
        else
            go to LOOP (try again);
    else if frame size reduction is best then
        calculate the resulting bandwidth assuming a reduced frame size;
        if satisfactory then
            reduce the frame size;
            go to DONE (finished);
        else
            calculate the resulting bandwidth assuming a reduced frame rate;
            if satisfactory then
                reduce the frame rate;
                go to DONE (finished);
            else
                reduce the frame size;
                go to LOOP (try again);
    else if frame rate reduction is best then
        calculate the resulting bandwidth assuming a reduced frame rate;
        if satisfactory then
            reduce the frame rate;
            go to DONE (finished);
        else
            calculate the resulting bandwidth assuming a reduced frame size;
            if satisfactory then
                reduce the frame size;
                go to DONE (finished);
            else
                reduce the frame rate;
                go to LOOP (try again);
DONE:
```

The high-level decision maker considers the three objective functions and the measured average throughput when determining which values for the frame rate, frame size, and Q factor are best. Because the bandwidth usage is proportional to the frame rate and the frame size, it is easy to calculate the change in required bandwidth that will result from a given change in frame rate or frame size. However, there is no simple relationship between changes in Q factor and changes in required bandwidth; for a given Q factor, the amount of compression attained (and therefore the required bandwidth) depends on the complexity of the contents of the frame. The decision maker must therefore determine the effect of changes in the Q factor by monitoring the changes in bandwidth as the Q factor is varied.

9 RELATED WORK

In the following subsections, we briefly discuss related work on aspects of the system resource management problem. We discuss algorithms for scheduling individual resources, as well as algorithms that consider more than one objective. We discuss recent or current projects sponsored by Rome Laboratory and other agencies that show potential for symbiosis with the SRM project and follow-on work.

9.1 SCHEDULING

Considerable research has been done to develop algorithms for the real-time scheduling of individual low-level resources such as CPU, disks, buffers, and communication channels. Many of the algorithms use information that can be derived from time-benefit functions and estimated task execution times. In general, when more information is used, better results are obtained, but at higher scheduling overheads.

Some scheduling approaches have considered trade-offs among particular objectives: for example, timeliness and importance (e.g., LBES [Locke, Jensen, and Tokuda 1985], CBS [Peha 1991], and DAS [Clark 1990]); timeliness and precision (e.g., Liu's algorithms for scheduling imprecise computations [Liu et al. 1991] and CHAOS [Gopinath and Schwan 1989]); and timeliness, importance, and precision (e.g., Better Late than Never [Moiin and Smith 1992]). The approaches that trade off precision, importance, and timeliness are the most general, since many high-level objectives can be mapped into these low-level objectives. There are alternative abstractions for supporting precision: precision-value functions, OR dependencies, and precedence constraints. The advantages and disadvantages of these abstractions for precisions should be further investigated, for example, by using simulations to compare (1) CBS with OR dependencies of alternate precisions; (2) LBES adapted to support alternate precisions using an approach like that of Liu et al.; and (3) the approaches proposed by Moiin and Smith.

Related work in low-level integrated scheduling and control was discussed in a previous SRM interim report [Downing and Davis 1993].

9.2 RELATED PROJECTS SPONSORED BY ROME LABORATORY

Under the sponsorship of Rome Laboratory, the problem of managing limited system resources to meet objectives has begun to be addressed in such projects as the Alpha Soft Real-Time Operating System project, the Adaptive Fault-Resistant Systems project, the Cooperative Gateway project, and the Multilevel Secure Real-Time Distributed Operating System project. In the following subsections, we briefly describe the aspects of these projects that are relevant to SRM.

9.2.1 Alpha Soft Real-Time Operating System

The Alpha operating system provides support for mission-critical, real-time computing in large, complex, distributed environments. Alpha considers both the importance and the urgency of competing activities when determining an execution schedule, and expresses these attributes using time-value functions. Scheduling is done using "best-effort" algorithms that provide graceful degradation under overload conditions. The Alpha distributed computing model is based on threads and objects.

Many SRM abstractions are extensions and generalizations of Alpha abstractions; for example, SRM objective functions are generalizations of Alpha time-value functions. Alpha currently does not handle attributes other than time constraints (for example, precision requirements), and it does not coordinate the management of resources other than processing time (e.g., buffers); nevertheless, it provides a good starting point for SRM.

In a project currently sponsored by Rome Laboratory, the Alpha/Mach Integration Study [Burke 1993], the important features of Alpha, such as distributed threads and best-effort scheduling, are being integrated into the Mach kernel. The resulting Alpha/Mach Integrated Operating System, expected to be demonstrated in early 1996, will be a useful research tool for future SRM concepts.

9.2.2 Adaptive Fault-Resistant Systems

The Adaptive Fault-Resistant Systems project [Goldberg et al. 1993] is developing a model for adapting among different techniques for providing fault tolerance. Figure 16 shows the data flow of AFRS, which is similar at a high level to the data flow for SRM. The AFRS project focuses on fault-tolerance issues, while the SRM project is concerned with developing a model for system resource management as a whole. In future SRM work, the model and the adaptivity techniques developed within the AFRS project may be generalized to include other application objectives.

In a related project [Gupta 1993], a prototype implementation is being developed that demonstrates the dynamic adjustment of fault management techniques in response to changes in system requirements.

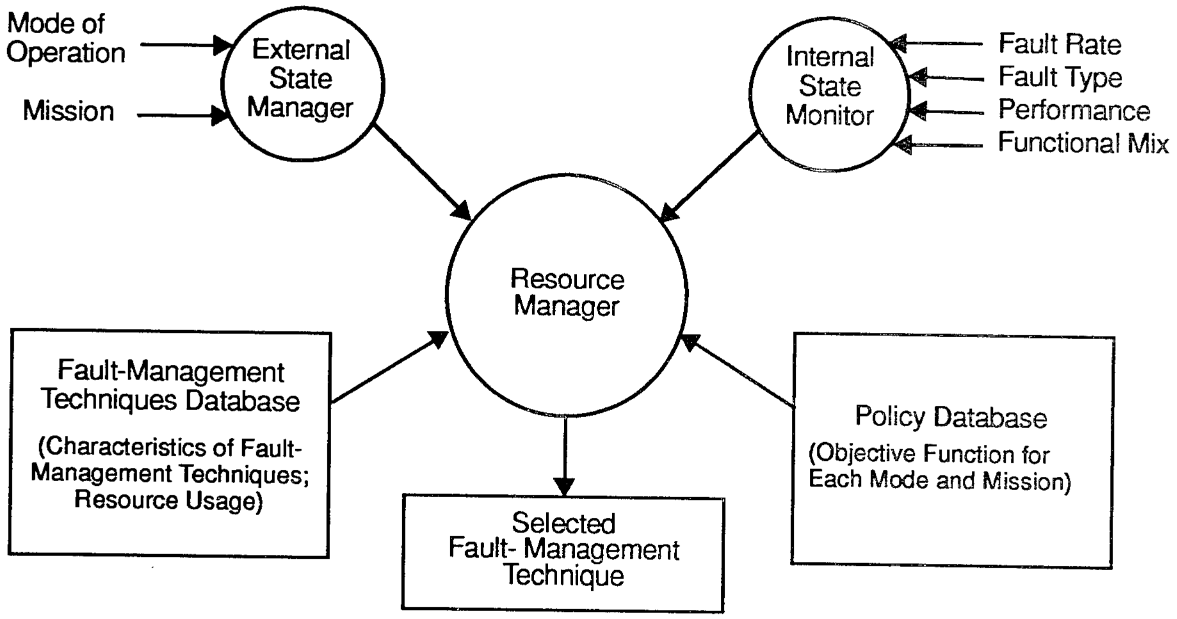

Mode of Operation
Mission
External State Manager
Internal State Monitor
Fault Rate
Fault Type
Performance
Functional Mix
Resource Manager
Fault-Management Techniques Database
(Characteristics of Fault-Management Techniques; Resource Usage)
Policy Database
(Objective Function for Each Mode and Mission)
Selected Fault- Management Technique

Figure 16. High-Level Data Flow in the Adaptive Fault-Resistant System

9.2.3 Cooperative Gateway

The goal of the Cooperative Gateway project [Lee et al. 1993] is to design and demonstrate a prototype cooperative gateway for interdomain, policy-based communications that satisfies the requirements of the International Military Internet. The Cooperative Gateway project allows the specification of objectives and requirements (i.e., policies) for routing messages; some of the abstractions will be relevant for future phases of the SRM project. A brief outline of objectives that are being considered for the Cooperative Gateway, including an indication of each objective's priorities and whether the objective is planned to be demonstrated is provided below.

- Access Restrictions
  - — Path
    - Source user class identification, host, or domain (1)*†
    - Destination host or domain (1) *
    - Entry/exit domain*
    - Domain (to be) traversed (1)
  - — Time of day (5)
  - — Traffic type
    - Protocol (e.g., FTP or mail)
    - Data, voice, or video
    - Traffic sensitivity levels (e.g., security) (3)
    - Data semantics (e.g., sensor data) (3)*
  - — Service
    - Precedence (e.g., priority) (4)
    - Mutual aid
    - Resource allocation (i.e., congestion) (2)
      - Usage limitation*
      - Sharing on a noninterfering basis
      - Sharing with guaranteed percentage of resources
- Quality of Service: Performance (e.g., throughput, delay, reliability)
- Cost:
  - — Bytes transmitted
  - — Packets transmitted.

*Asterisks denote policies planned for prototyping.

†Numbers in parentheses indicate the relative importance of the policies: higher numbers indicate greater importance.

The Cooperative Gateway project deals mainly with communication resources and is not explicitly concerned with real-time issues; however, its use of policies for making routing decisions may be applicable to some aspects of integrated resource management.

9.2.4 Multilevel Secure Real-Time Distributed Operating System

The Multilevel Secure Real-Time Distributed Operating System project (also known as Secure Alpha) [Greenberg et al. 1993] developed abstractions related to trade-offs among security and other requirements such as timeliness, distribution, robustness, and adaptability. Applications are able to specify information to the Secure Alpha operating system that allows Secure Alpha to make informed tradeoffs when all application requests cannot be satisfied. For example, applications can specify time constraints, the relative importances of different activities, guidelines for trading off among requirements, and scheduling policies.

The Secure Alpha project studied the tradeoff between timeliness and covert channel bandwidth. (A covert channel is a communication channel that allows a process to transfer information in violation of a system's security policy. A covert timing channel is one in which a process signals information by using system resources in such a way that the response times of other processes are affected.) The statistical nature of security violations is related to the statistical nature of other kinds of system faults that are relevant to the SRM project.

9.3 OTHER RELATED PROJECTS

The following efforts that are currently underway at SRI are relevant to the SRM project.

9.3.1 Heterogeneous Multicast

In a current project sponsored by ARPA, Heterogeneous Multicast (HMC) [Shacham 1992], SRI is researching techniques for efficiently communicating multimedia information from a single source to multiple destinations in a heterogeneous networked environment. Because of differing capacities on the various links, and differing display capabilities on the various terminals, it is desirable to route different parts of the multimedia information to different destinations. The approach used by HMC is to separate the data into multiple layers of varying quality, in such a way that the data in each layer augments the data in lower layers. The layers are communicated as separate data streams. Each user specifies which layers are needed, and the HMC system routes the layers appropriately to conserve communication resources. A typical set of layers is shown in Figure 17. Layer 0 contains an audio stream, which is considered the most important information. Layers 1 and above contain streams of video frames; each frame is encoded via JPEG compression algorithm. The lowest layers contain low-quality images, as reflected by large Q factors. (The Q factor is an indication of the amount of data loss; a large Q factor indicates more compression and greater data loss, and therefore lower quality.) Higher layers contain successively higher-quality images. In the particular coding method used within HMC, each video layer is independent of the lower layers, resulting in some redundancy (since the information at lower layers is repeated in the upper layers). A different coding algorithm could eliminate this redundancy and therefore use the communication resources more efficiently.

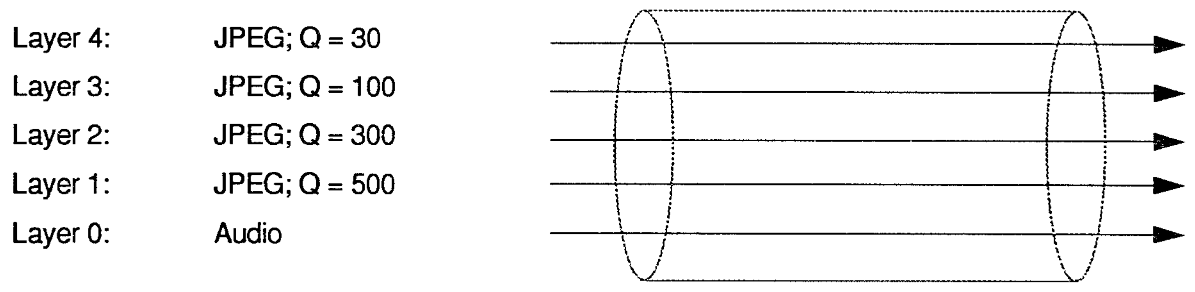


Figure 17. Typical Layers for Hierarchical Multicast

The HMC project deals only with communication resources, but the concepts are transferable to other kinds of resources. For example, applications could be structured to produce sets of partial results with different levels of precision; similar approaches have been used for scheduling imprecise computations [Liu et al. 1991].

9.3.2 Multimedia Multiplexer

In a recent project sponsored by the Electronics and Telecommunications Research Institute (ETRI) of Korea, SRI has developed a multimedia input/output server, the Multimedia Multiplexer (MuX) [Rennison et al. 1993]. The MuX server is coresident with the X Window System server and provides support for the integration and synchronization of time-based media streams. Key aspects of MuX include a client-server model to facilitate distributed computing; a well-developed media integration model; fine-grained synchronization and integration; network-transparent access to multimedia data; and a scripting language to specify time and space relationships among the media. The MuX server has already incorporated some of the concepts of SRM, including using objective functions to specify user preferences and scheduling competing threads using best-effort scheduling. Multimedia applications built on a modified MuX server may be useful for demonstrating SRM concepts.

10 CONCLUSION

In the following subsections, we briefly review the status of the current SRM effort and recommend areas for future research.

10.1 STATUS

We have developed a model for integrated resource management in distributed real-time systems that considers user objectives, resource constraints, and adaptable execution techniques. In developing the model, we have explored a wide range of topics, have defined the scope of the problems that should be addressed under this program, and have identified potential approaches to solving these problems.

Our model used key abstractions from the Alpha soft real-time operating system as a starting point, including such abstractions as distributed threads, objects, and time-value functions. We developed the objective function abstraction as a generalization of the Alpha time-value function.

We notionally applied the model to a multimedia scenario (and, to a lesser extent, a C^2 scenario). We implemented a proof-of-concept demonstration of high-level integrated control, using a simple distributed multimedia application.

10.2 RECOMMENDATIONS FOR FUTURE WORK

In follow-on work to the SRM project, it will be necessary to further develop and apply the work begun in the first phase. We recommend the following efforts:

1. Refine the concepts developed during the first phase.
   - A. Further develop and formalize the models (e.g., the objective functions and the architecture of the control model) developed during the first phase; develop a model for combining and trading off multiple, possibly conflicting, objective functions.
   - B. Further develop algorithms for making distributed control decisions.
     - (1) Determine the required inputs to the algorithms (e.g., information about resource status, application resource usage, and application objectives).
     - (2) Determine what will be controlled and/or managed, and what the required outputs of the algorithms will be.
     - (3) Develop decision algorithms. Such algorithms must be capable of efficient and scalable implementation in distributed tactical computing environments. Examine alternate approaches for limiting the search space, such as heuristic or knowledge-based approaches.

2. Develop the necessary support software.
   - A. Determine the required application support and interfaces to the underlying resource management system.
   - B. Implement the key resource management algorithms and mechanisms within or above an operating system such as UNIX, Alpha, Mach, or integrated Alpha/Mach. When appropriate, use existing products or standards (for example, network management protocols and management information bases).
   - C. Implement a toolkit that allows applications to access the services of the resource management system.
3. Develop a demonstration application.
   - A. Implement a demonstration application (or extend an existing one) that uses the toolkit and the resource management system, and that shows the benefits of the selected approach. Potential applications include an extended multimedia conferencing application based on one developed during the first phase of the SRM project, or a C^2 application based on a demonstration command, control, and battle management (C2BM) system developed for the Alpha operating system at Carnegie Mellon University and Concurrent Computer Corporation.

11 REFERENCES AND ANNOTATED BIBLIOGRAPHY

11.1 REFERENCES

Bhargava, B., and J. Riedl. 1988. "A Model for Adaptable Systems for Transaction Processing," 1988 IEEE Fourth Conference on Data Engineering (February).

Burke, E.J. 1993. "Alpha/Mach Integration Study," presented at Rome Laboratory/C3AB Technical Exchange (October).

Clark, R.K. 1990. "Scheduling Dependent Real-Time Activities," Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Craig, G., 1991. "Distributed Resource Management," presentation at the 1991 Rome Laboratory Technical Exchange Meeting, Rome, New York (November).

Davis, M., A. Downing, and T. Lawrence. 1994. "Adaptable System Resource Management for Soft Real-Time Systems," presented at the *Symposium on Command and Control Research and Decision Aids*, Monterey, California (June).

Davis, M., and N. Siravara. 1993. "System Resource Management Demonstration Software," ITAD-2655-TR-434, SRI International, Menlo Park, California (December).

De Boor, A., 1990. "Customs—A Load Balancing System," Final Project, University of California, Berkeley (June).

Downing, A., and M. Davis. 1992a. "System Resource Management for Distributed Real-Time Systems," ITAD-2655-TR-92-64, SRI International, Menlo Park, California (April).

Downing, A., and M. Davis. 1992b. "System Resource Management for Distributed Real-Time Systems: Sample Scenario," ITAD-2655-TR-92-123, SRI International, Menlo Park, California (August).

Downing, A., and M. Davis. 1993. "System Resource Management for Distributed Real-Time Systems: Sample Architecture," ITAD-2655-TR-93-59, SRI International, Menlo Park, California (March).

Flanagan, J.L., M.R. Schroeder, B.S. Atal, R.E. Crochier, N.S. Jayant, and J.M. Tribolet. 1979. "Speech Coding," *IEEE Trans. Communications*, pp. 710-737 (April).

Goldberg, J., I. Greenberg, R. Clark, K. Kim, D. Jensen, and D. Wells. 1993. "Adaptive Fault Resistance," presented at Rome Laboratory/C3AB Technical Exchange (October).

Gopinath, P., and K. Schwan. 1989. "CHAOS: Why One Cannot Have Only an Operating System for Real-Time Applications," *ACM Operating Systems Review*, Vol. 23, No. 3 (July).

Greenberg, I., P. Boucher, R. Clark, D. Jensen, T. Lunt, P. Neumann, and D. Wells. 1993. "The Secure Alpha Study—Final Summary Report," SRI Project 2385, SRI International, Menlo Park, California (June).

Gupta, B. 1993. "Adaptive Fault Resistant System (AFRS) Project Overview," presented at Rome Laboratory/C3AB Technical Exchange (October).

Lee, D.S., S.T. Merchant, D.Y. Lee, and J.Wong. 1993. "Enhanced Internetwork Policy Routing: Software Design Document—Draft," ITAD-3603-TR-93-051R2, SRI International, Menlo Park, California (August).

Liu, J.W.S., K. Lin, W. Shih, A.C. Yu, J. Chung, and W. Zao. 1991. "Algorithms for Scheduling Imprecise Computations," *IEEE Computer* (May).

Locke, C.D., E.D. Jensen, and H. Tokuda. 1985. "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of IEEE Real-Time Systems Symposium* (December).

Moiin, H., and P.M.M. Smith. 1992. *Better Late Than Never*, submitted for publication, University of California, Santa Barbara, California.

Northcutt, J.D., and R.K. Clark. 1988. *The Alpha Operating System: Programming Model*, Archons Project Technical Report 88021, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (February).

Northcutt, J.D., R.K. Clark, S.E. Shipman, and D.C. Lindsay. 1988. *The Alpha Operating System: System/Subsystem Specification*, Archons Project Technical Report 88051, Department of Computer Science, Carnegie Mellon University (May).

Peha, J. 1991. "A Cost-Based Scheduling Algorithm to Support Integrated Services," *Proceedings IEEE INFOCOM-91*, Miami, Florida (March).

Peha, J.M., and F.A. Tobagi. 1991. "A Cost-Based Scheduling Algorithm to Support Integrated Services," *Proceedings of IEEE INFOCOM '91*, Miami, Florida (March).

Peterson, L., N. Hutchinson, S. O'Malley, and H. Rao. 1990. "The x-kernel: A Platform for Accessing Internet Resources," *IEEE Computer*, Vol. 23, No. 5 (May).

Popek, G.J., R.G. Guy, T.W. Page, and J.S. Heidemann. 1990. "Replication in Ficus Distributed File Systems," *Proceedings of the IEEE Workshop on Management of Replicated Data*, Houston, Texas (November).

Rennison, E., M. Brown, A. Downing, K. Finn, and S. Randolph. 1993. *The MuX Multimedia I/O System*, ITAD-2026-TR-93-178, SRI International, Menlo Park, California (May).

Shacham, N. 1992. "Multipoint Communication by Hierarchically Encoded Data," *Proceedings of IEEE INFOCOM '92*, Florence, Italy, pp. 2107-2114 (May).

Sycara, K.P., S.F. Roth, N. Sadeh, and M.S. Fox. 1991. "Resource Allocation in Distributed Factory Scheduling," *IEEE Expert*, Vol. 6, No. 1, pp. 29-40 (February).

Zhao, W., and K. Ramamritham. 1985. "Distributed Scheduling Using Bidding and Focused Addressing," *Proceedings of the IEEE Real-Time Systems Symposium* (December).

11.2 ANNOTATED BIBLIOGRAPHY

Abbott, R.K., and H. Garcia-Molina. 1988. "Scheduling Real-Time Transactions: A Performance Evaluation," *Proceedings of the 14th VLDB Conference*, Los Angeles, California.

This was one of the earliest papers on in-depth simulations that investigated the integration of resource scheduling techniques such as FCFS, EDF, and LL transaction scheduling with high-priority and conditional restart concurrency control.

Abbott, R.K., and H. Garcia-Molina. 1992. "Scheduling Real-Time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 17, No. 3, pp. 513-560 (September).

This is a combination of two other papers by Abbott and Garcia-Molina.

Abbott, R.K., and H. Garcia-Molina. 1990. "Scheduling I/O Requests with Deadlines: A Performance Evaluation," presented at IEEE Real-Time Systems Symposium.

This paper further develops a 1988 paper (referenced above) by the same authors. The simulations described in this paper integrated transaction scheduling, concurrency control, and I/O scheduling. In addition to the traditional disk scheduling techniques (i.e., FCFS, shortest seek time, elevator scan), real-time disk scheduling (i.e., EDF, EDF Scan, Feasible-Deadline Scan [FDscan]) was simulated. Real-time buffer management was also investigated. A sample result is that FDscan tended to miss fewer deadlines, but had higher average response times and was very unfair to disks with deadlines on the outermost areas.

Baker, R., A. Downing, K. Finn, E. Rennison, D. Kim, and Y. Lim. 1992. "Multimedia Processing Model for a Distributed Multimedia I/O System," presented at the Third International Workshop on Networking and Operating System Support for Digital Audio and Video, San Diego, California (November).

Baumgartner, K.M., R.M. King, and B. Wah. 1989. "Implementation of Gammon: An Efficient Load Balancing Strategy for a Local Computer System," *Proceedings of International Conference on Parallel Processing*.

Bihari, T.E., and P. Gopinath. 1992. "Object-Oriented Real-Time Systems: Concepts and Examples," *IEEE Computer*, Vol. 25, No.12 (December).

Brady, P.T. 1968. "A Statistical Analysis of On-Off Patterns in Sixteen Conversations," *The Bell System Technical Journal*, Vol. 47, No. 1 (January).

Chu, W.W. 1992. "COBASE: Cooperative Distributed Database Systems," presentation at SRI International, Menlo Park, California (November).

This knowledge base supports the relaxation of constraints until queries receive nonnull responses.

Clark, R.K. 1990. "Scheduling Dependent Real-Time Activities," Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

This work (also cited in this report and listed as a reference) combined a scheduling technique similar to LBES with conditional restart transaction management.

Cox, E. 1993. "Adaptive Fuzzy Systems," *IEEE Spectrum* (February).

Daumer, W.R. 1982. "Subjective Evaluation of Several Efficient Speech Coders," *IEEE Trans. Communications*, pp. 655-662 (April).

Fox, E.A. 1991. "Advances in Interactive Digital Media Systems," *IEEE Computer*, Vol. 24, No. 10 (October).

Gatenbein, R.E., T.F. Lawrence, and S.Y. Shin. 1992. "A Taxonomy for Fault Management in Survivable Distributed Systems," submitted to the *Symposium on Reliable Distributed Systems.*

Gopinath, P., and K. Schwan. 1989. "CHAOS: Why One Cannot Have Only an Operating System for Real-Time Applications," *ACM Operating Systems Review*, Vol. 23, No. 3 (July).

In CHAOS, a hard real-time system with EDF scheduling, if loads are increasing (due to external influences) to the point where activities may miss a deadline, the bindings are changed to more imprecise and less time-consuming alternate methods for an object. The CHAOS architecture monitors applications, checks constraints, and then selects and enacts adaptation. This paper is also listed above as a reference cited in this report.

Graham, M.H. 1992. "Issues in Real-Time Data Management," *The Journal of Real- Time Systems*, Vol. 4, pp. 185-202.

This is a good review of current research, including summaries of some experimental and simulation results as well as an extensive bibliography.

Haritsa, J.R., M.J. Carey, and M. Livny. 1990. "Dynamic Real-Time Optimistic Concurrency Control," presented at IEEE Real-Time Systems Symposium.

Although non-real-time work has concluded that locking protocols, which conserve resources, perform better than optimistic techniques when resources are limited, real-time research has concluded the opposite. When EDF is used and late transactions are immediately discarded, optimistic concurrency control outperforms locking over a wide range of system loading and resource availability. Discarded transactions do not cause restarts of other transactions in OCC. Two problems with high-priority two-phase locking (2PL) are (1) restarts for transactions that are ultimately discarded; and (2) priority reversal when there are time-dependent priorities.

This paper proposes a new version of optimistic currency control (OCC) in which low-priority transactions that would cause the restart of higher-priority transactions would wait until the higher transactions abort or commit; if a higher-priority transaction commits, conflicting low-priority transactions are restarted.

Harista, J.R., M. Livny, and M.J. Carey. 1991. "Earliest Deadline Scheduling for Real-Time Database Systems," presented at IEEE Real-Time Systems Symposium.

Previous work makes it clear that EDF works worse than most policies when an overload occurs, because tasks with insufficient time to complete before their deadlines are often highest priority. On the other hand, approaches that use time-value functions cannot be used for real-time databases, since knowledge about transaction resource and data requirements is unavailable in database applications. Therefore, this paper proposes

"Adaptive Earliest Deadline," where the goal is to maximize the value of in-time transactions. This policy allows a tradeoff between transaction values and deadlines in generating priority order.

Huang, J., J. Stankovic, K. Ramamritham, and D. Towsley. 1991. "On Using Priority Inheritance in Real-Time Databases," presented at IEEE Real-Time Systems Symposium.

This paper gives results that integrate CPU scheduling (i.e., EDF and Multilevel feedback queue) with conflict resolution (i.e., Always Wait, Priority Inheritance, Priority Abort, and Conditional Restart). A sample result is that Conditional Restart works well for wide range of loads.

Hui, J.Y. 1988. "Resource Allocation for Broadband Networks," *IEEE JSAC*, Vol. 6, No. 9 (December).

Jayant, N.S., and P. Noll. 1984. *Digital Coding of Waveforms: Principles and Applications to Speech and Video*, Prentice-Hall International, Inc., Englewood Cliffs, New Jersey.

Kao, B., and H. Garcia-Molina. 1992. "An Overview of Real-Time Database Systems," Technical Report, Stanford University, Stanford, California.

This is an excellent overview paper. It summarizes I/O policies (i.e., FIFO, Priority, By Position), policies to resolve data conflicts (i.e., Wait, Highest Priority Wins, Wait-Promote, Conditional Restart) and transaction scheduling algorithms (i.e., FCFS, EDF, Least Laxity).

Results for scheduling/data conflicts that were discussed included the following:

- *EDF/WP worked best when there were low to medium arrival rates.*
- *LL/WP worked best when there were high arrival rates.*
- *When there was a single spike "overload," high priority was the better way of resolving shared data conflicts for locking techniques.*

Kao, B., and H. Garcia-Molina. 1992. "Deadline Assignment in a Distributed Soft Real-Time System," Technical Report STAN-CS-92-1452, Stanford University, Stanford, California (October).

In the system under study, transactions could be broken down into subtasks (e.g., for disk or for CPU) and subtasks could then be scheduled separately. For example, if least laxity is used, slack could be allocated proportionately to the subtasks' expected run times and each subtask could be given its own deadline that is sooner than the "global" deadline (for the full transactions). As a result, those subtasks with much slack time to begin with completed easily, while those without ample slack time to begin with would often miss their deadlines.

Kim, W., and J. Srivastava. 1991. "Enhancing Real-Time DBMS Performance with Multiversion Data and Priority Based Disk Scheduling," presented at IEEE Real-Time Systems Symposium.

Kim's and Srivastava's results show that these techniques decrease miss ratio significantly over non-real-time systems.

Liu, J.W.S., K. Lin, W. Shih, A.C. Yu, J. Chung, and W. Zao. 1991. "Algorithms for Scheduling Imprecise Computations," *IEEE Computer* (May)

These algorithms reduce the overall error in a system where tasks can use their intermediate results. A variation of EDF schedules mandatory subtasks first and then tries to schedule as many optional subtasks as possible without removing any mandatory subtasks from the schedule. (This paper is also cited in the text of this report and appears in the list of references.)

Locke, C.D., E.D. Jensen, and H. Tokuda. 1985. "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings of the IEEE Real-Time Systems Symposium* (December).

This paper describes Locke's Best Effort Scheduler (LBES). By means of a two-pass algorithm, potential jobs with the highest value densities (value accrued per unit of remaining computation time) are added to a schedule in EDF order as long as they can make their deadline.

Maynard, D.P., S.E. Shipman, R.K. Clark, J.D. Northcutt, R.B. Kegley, B.A. Zimmerman, and P.J. Keleher. 1988. "An Example Real-Time Command, Control, and Battle Management Application for Alpha," Archons Project Technical Report 88121, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (December).

Moiin, H. and P.M.M. Smith. 1992. "Better Late Than Never," submitted for publication, University of California, Santa Barbara, California.

Moiin and Smith's algorithm allows a tradeoff between precision and timeliness. Precision-value functions and time-value functions are combined into a single function. Optimal (i.e., using an NP complete algorithm) and heuristic (i.e., solving linear differential equations for linear objective functions) solutions to the subproblems of ordering (e.g., EDF) and optimization (of value) for a given set of tasks are discussed. (This paper is also cited in the text and reference list of this report.)

Montanari, U. 1974. "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Information Sciences*, Vol. 7, pp. 95-132.

National Computer Security Center. 1985. "DoD Trusted Computer System Evaluation Criteria (TCSEC)," *Orange Book* (December).

Peha, J. 1992. "Priority Token Bank: Admission Control and Scheduling for an Integrated-Services Network," work in progress, Carnegie Mellon University, Pittsburgh, Pennsylvania.

The Priority Token Bank allows different performance objectives to be met, while maintaining the rates for high-speed networks.

Pu, C. 1991. "Generalized Transaction Processing," *International Workshop on High Performance Transaction Systems.*

Ripley, G.D. 1989. "DVI-A Digital Multimedia Technology," *Communications of the ACM*, Vol. 32, No. 7 (July).

Shacham, N., and P. McKenney. 1990. "Packet Recovery in High-Speed Networks using Coding and Buffer Management," *Proc. IEEE Infocom '90*, San Francisco, California, pp. 124-131 (June).

Shivaratri, N.G., P. Krueger, and M. Sinhal. 1992. "Load Distributing for Locally Distributed Systems," *IEEE Computer*, Vol. 25, No.12 (December).

Stankovic, J.A., and W. Zhao. 1988. "On Real-Time Transactions," *SIGMOD Record*, Vol. 17, No. 1 (March).

This often-cited paper presents key ideas on real-time transactions.

Tokuda, H., C.W. Mercer, Y. Ishikawa, and T.E. Machok. 1989. "Priority Inversions in Real-Time Communication," presented at IEEE Real-Time Systems Symposium.

Due to priority inversion problems, message communication may not be bounded in a network without real-time communications support. While most communication protocols do not support real-time abstractions, VMTP was an exception that the authors exploited.

Wu, K., P.S. Yu, and C. Pu. 1991. *Divergence Control for Epsilon-Serializability*, Technical Report CUCS-002-91, Columbia University, New York, New York (June).

A bounded degree of divergence can decrease the amount of blocking and roll-back required and therefore improve performance. The bounds are user-provided objectives.

Zhang, L. 1992. "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," talk at the Stanford Distributed Systems Research Seminar (30 April).

Appendix A

OPEN INTEGRATED SYSTEM MANAGEMENT

OPEN INTEGRATED SYSTEM MANAGEMENT

Elin L. Klaseen
Information, Telecommunications, and Automation Division
SRI International

A.1 INTRODUCTION

The System Resource Management (SRM) program goal is to define mechanisms for real-time management and control of distributed resources for specific classes of users and applications. This goal poses formidable problems; yet, by extending SRM to incorporate existing standards-based management work, the SRM program can effectively advance its goals. In this appendix we seek to define a strategy for expanding the SRM work into two areas: integration of communication resource management, and utilization of open systems based management standards.

Several communities have participated in the management standardization efforts representing various interests. Although past work has focused on management of communication resources, the architectural models proposed are applicable to system and communication resource management. Standards not only define an overall architectural model for system management, but also provide a common syntax for defining management information, a common protocol for accessing that information, and the beginnings of functional units for manipulating that information. By basing the SRM management information structure, manager interactions, and management functions on standardized interfaces, SRM can leverage existing standardized products, as well as provide a flexible and extensible framework for future distributed integrated management efforts at Rome Laboratory.

This appendix has five sections. Section 2 presents background information on the scope and status of the standardization efforts and outlines how SRM fits into these system management models. Section 3 presents a more detailed look at the standards and how they apply to communication resources; initial strategies for defining how SRM can use these management standards are discussed. Section 4 presents the preliminary work within the standardization communities in the area of host and system resources. Again, strategies are presented for defining how SRM can benefit from this existing work. Section 5 lays out possible future strategies for the System Resource Management Program.

A.2 SYSTEM MANAGEMENT STANDARDS

For a more comprehensive description of standardization efforts, the reader is referred to the general texts [Westgate 1992; Rose 1990]; the International Standards Organization (ISO) standards; and the Internet Requests for Comments (RFCs).

A.2.1 SCOPE

System management, in its most general sense, is the management of resources. The term resource can mean anything that plays a part in the operation of a system, including hardware, software, a process, or stored data. Although the system management effort was initially focused on communication resource management, the basic models proposed are being extended to apply to the management of distributed applications. The discussion and examples presented here are focused mainly on communication resources. Within any management framework, there are various levels at which a "management" task might be performed:

- Management within the operation of an instance of communication (i.e., negotiation of quality of service parameters). These mechanisms are generally provided by the protocol in use.
- Management within a given layer of the communication stack (i.e., routing tables updates for network layer protocols, or station management for FDDI), which can affect several instances of communication. These mechanisms generally use a special-purpose layer management protocol to perform their function.
- Management of system resources through the use of general-purpose management tools and protocols.

The goal of integrated system management implies that one utilizes the latter (i.e., general purpose management tools and protocols), to monitor, control and coordinate the operation and layer management tasks. To achieve this goal, management standardization efforts have defined the following:

- A structure for the representation of management information
- Common management functions and services
- A common protocol as a means of communicating management information.

Note that the above are defined independently of the particular resource being managed. System management, therefore, is really more the management of the abstraction of a resource (i.e., a managed object).

As with other areas of computing technology, there are multiple methods and proposals for system management. Each method has its advantages and disadvantages when applied to a given system. These efforts, however, are not diametrically opposed, and through intermediaries such as application gateways and proxies, are being brought together in integrated management systems.

A.2.2 SYSTEM MANAGEMENT ARCHITECTURE

The following discussions of system management architecture have a decidedly OSI slant. This is not meant as a wholesale endorsement of that approach, but rather, a recognition and acceptance of the terminology and functional divisions and units they propose. Similar capabilities may or may not be found in other approaches, but the author believes that most commercial management future strategies will necessarily accommodate at a minimum both the Internet and the OSI methods.

OSI divides the functional requirements of system management into five broad categories. The following descriptions have been taken from ISO standard 7498-4:

> Fault management is the set of facilities which enables the detection, isolation and correction of abnormal operation of the OSI Environment (OSIE). Faults cause open systems to fail to meet their operational objectives and they may be persistent or transient. Faults manifest themselves as particular events (e.g., errors) in the operations of an open system. Error event detection provides the mechanism for recognizing faults. Fault management is the set of facilities to:
>
> - maintain and examine error logs;
> - accept and act upon error detection notifications;
> - trace faults;
> - carry out sequences of diagnostic tests;
> - correct faults.
>
> Configuration management identifies, exercises control over, collects data from and provides data to open systems for the purpose of preparing for, initializing, starting, providing for the continuous operation of, and terminating interconnections services. Configuration management includes functions to
>
> - set the parameters that control the routine operation of the open system;
> - associate names with managed objects and sets of managed objects;
> - initialize and close down managed objects;
> - collect information on demand about the current condition of the open system;
> - obtain announcements of significant changes in the condition of the open system; and
> - change the configuration of the open system.
>
> Performance management enables the behavior of resources in the OSIE and the effectiveness of communication activities to be evaluated. Performance management includes functions to
>
> - gather statistical information;
> - maintain and examine logs of system state histories;
> - determine system performance under natural and artificial conditions; and
> - alter system modes of operation for the purpose of conducting performance management activities.

The purpose of security management is to support the application of security policies by means of functions which include

- the creation, deletion and control of security services and mechanisms;
- the distribution of security-relevant informations; and
- the reporting of security-relevant events.

Accounting management enables charges to be established for the use of resources in the OSIE, and for costs to be identified for the use of those resources. Accounting management include functions to

- inform users of costs incurred or resources consumed;
- enable accounting limits to be set and tariff schedules to be associated with the use of resources; and
- enable costs to be combined where multiple resources are invoked to achieve a given communication objective.

Although these areas are useful for general discussion and analysis requirements, they quickly become amorphous when one is talking about actual management applications and the functions they perform (e.g., consider a management application that detects degradation of service between two hosts, but upon closer examination determines a communication link is at fault, and corrects the problem by reconfiguring the line). This recognition that management applications are actually built upon functional units led the OSI community to define common management capabilities and objects as well as support routines and objects for those management functions.

A.2.2.1 Management Information

Fundamental to system management is the concept of the managed object (MO). The MO represents the resource to the management system. The management information is accessed via management operations, the results of those operations, and notifications representing events of significance as passed from the MO to the manager. Hence, in ISO terminology, a MO is defined in terms of

- The properties or characteristics visible at the managed object boundary—these are termed its *attributes*, and each attribute has a *value*
- The *management operations* that may be applied to it
- The *behavior* it exhibits in response to management operations
- The *notifications* it emits.

The ISO standards 10165-2 and 10165-4 represent the ISO definition of management information and guidelines for the definition of managed objects (GDMOs), respectively. The Internet equivalent is defined in RFC 1155.

A.2.2.2 Management Functions

The concept of management of functional areas in OSI grew out of a lengthy process of analyzing the requirements for network management. It was recognized that among the five areas of network management, common functional requirements existed for the logging of information, notification of events, scheduling of tasks, alarm reporting, and so forth. In addition, because of the chosen structure of management information, functional units specifically for the management of management information (i.e., object, state, and relationship management) were required.

The work load monitoring function (ISO 10165-11) and summarization functions (ISO 10165-13) are sample functions utilized in performance management. The work load monitoring function provides parameters from which a work load value can be calculated; it also provides the ability to capture the data (through polling), enhance the data through statistical averaging, and gauge the data according to specified thresholds. The summarization function extends the work load monitoring capabilities by being able to aggregate data from one or more attributes that may be obtained from one or more managed objects. Underlying the concept of summarization is the concept of a scanner. The standard defines various types of scanners that are differentiated by the statistical algorithms they use when enhancing and analyzing the data, and the nature of the objects and attributes to be monitored.

An additional functional unit of interest to SRM is the accounting meter function (ISO 10164-10), currently in draft status. This standard identifies the following objects of concern to accounting management:

- An accounting meter, which accounts for the usage of an resource
- An accounting record, which is a structure representation of accounting data
- An accounting log, which provides a repository for accounting records
- A quota record, which maintains information to authorize the use of a resource by a particular user.

A.2.2.3 Management Communication

The services and protocols supporting OSI management are provided by the Common Management Information Service Element (CMISE) and the Common Management Information Protocol (CMIP). Although originally specified to run over full OSI stacks (i.e., underlying connection-oriented transport of TP4), implementations exist that run over a TCP/IP stack (RFC 1006), and within LAN environments at the data link layer level (IEEE standard 802.1F). The Internet approach utilizes the Simple Network Management Protocol (SNMP).

In general, these approaches provide services for the retrieval and manipulation of information (gets and sets), and event-driven services such as event notifications and information reports. In addition to these, the OSI approach provides services for managed object manipulation such as the creation and deletion of a managed object and the invocation of a management action.

A.2.2.4 Management Model

The OSI model for management is shown in Figure A-1 [Westgate 1992, p. 8].

A managing process exists in a managing system and communicates with a managed system concerning one or more managed objects. The managed system contains an agent process, which is responsible for controlling access to the managed object (i.e., the management operations), and disseminating results of the operation in the form of event reports, and/or confirmations as appropriate. This simple manager/managed model can be hierarchically extended such that a system may act as a managing system in one interaction and as the managed system in another. Figure A-2 is an illustration of such possible multisystem configurations [Westgate 1992, p. 11].

A central problem of management is compressing vast amounts of real-time operational data to accomplish management decisions. Although the concept of layering the management allows increasing levels of abstraction and distillation of information, it can also distance the final management decision from the actual resource to the point where delays and overhead may make that decision potentially useless. The desire for an indication of the instantaneous resource state must be weighed against the overhead incurred in obtaining that status.

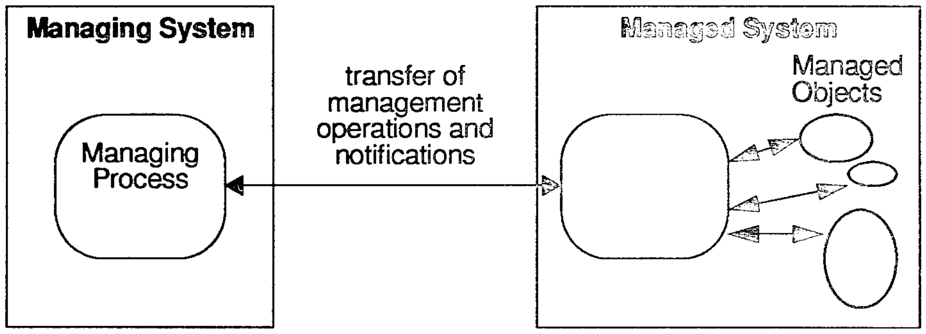


Figure A-1. Basic Model for Communication

A.2.3 EXISTING STANDARDS AND ORGANIZATIONS

The two primary organizations that promote management standards are the Network Management Area within the Internet Engineering Task Force (IETF) and the Network Management Working Groups within the ISO OSI community. The former is by far the most ubiquitous in terms of products, installations, and accessibility. The latter has yet to reach its full potential, but is gaining support from substantial industry partners. A third organization promoting yet another alternative is the Open Software Foundation (OSF) and the proposals for Distributed Computing and Management Environments (DCE and DME). Very generally, it can be said that the IETF proposals come from the data communication perspective, the OSI proposals from the telecommunication perspective, and the OSF proposals come from the relatively newer distributed computing perspective. To further characterize these approaches, it could be said that the Internet approach was bottom up (focused on end devices) and the ISO approach has been top down (defining application requirements). The DME approach is somewhere in the middle, attempting to generalize management services and interfaces to those services. The following sections briefly review these various approaches.

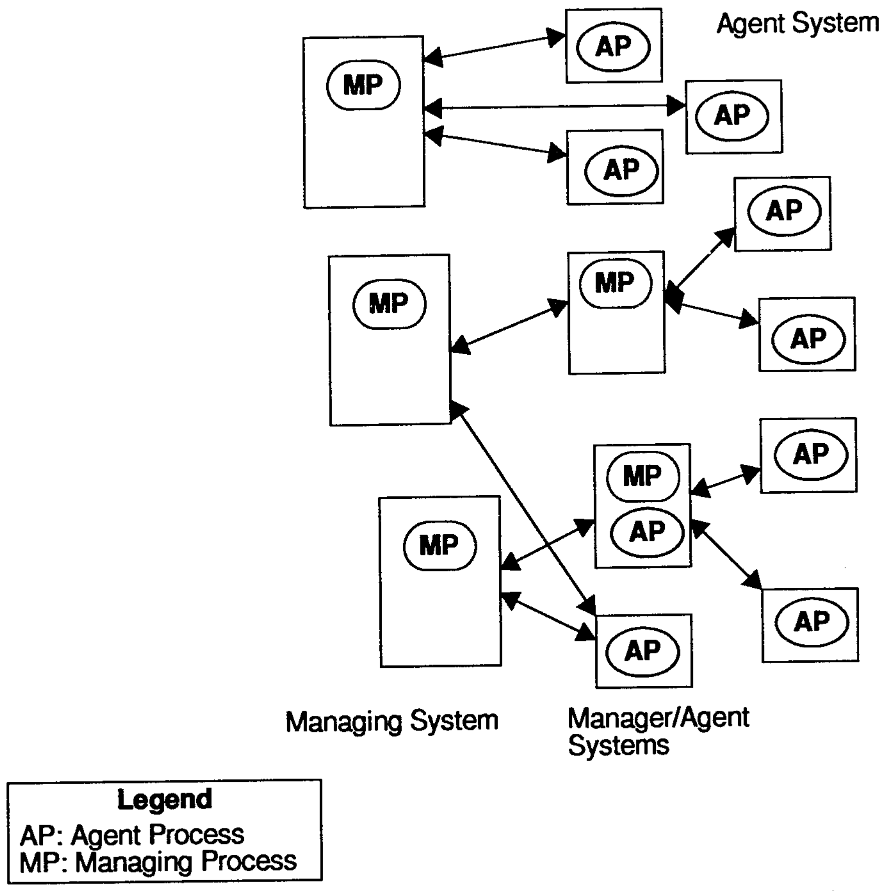


Figure A-2. Multisystem Configuration

A.2.3.1 SNMP

The structure and identification of management information for TCP/IP-based internetworks was first proposed in a collection of Internet RFCs in 1988. The underlying premise behind the approach was simplicity with extensibility; hence the name of the protocol: the Simple Network Management Protocol. At the time SNMP was initially proposed, it was viewed as a short-term solution, and it was assumed that the longer-term approach would be OSI's Common Management Information Protocol running over TCP/IP (CMOT). An evolutionary path was guaranteed by maintaining a similar structure of management information for both protocols. The increased complexity of supporting CMOT led the developers to abandon the CMOT transition path. SNMP, in the meantime, has gained a strong foothold in the marketplace, ensuring in some form its continued longevity. The SNMP protocol operates over a connectionless service (UDP) and assumes a polling paradigm. SNMP offers basic services for the retrieval and manipulation of management information (get, get-next and set) and basic event notification (trap). The next version of SNMP, SNMPv2, expands on basic services by providing mechanisms for large information transfer and a confirmed transfer (*getBulk* and *informPDU*, respectively), and additional security provisions.

In keeping with the concept of SNMP's simplicity, the SNMP developers chose a simple structure for management information. Managed objects are accessed via a virtual information store, termed the Management Information Base (MIB). Although management information is described by the data description language Abstract Syntax Notation One, (ASN.1), only a small subset of primitive types were allowed. The concept of an object type was also kept simple in that it basically is an abstraction of a single variable. Objects are related to each other by being grouped together in conceptual tables. Initial MIB definitions have focused primarily on end devices (e.g., modems, bridges, terminals, hubs, and concentrators). Application working groups are currently in the process of defining MIBs for such applications as electronic mail and directory services.

A.2.3.2 CMIP/CMISE

Subsection A2.2 above has provided some background on the OSI architecture for system management. The OSI management approach provides a complex and extensive structure for management information. In contrast to the SNMP approach in which managed objects must be statically defined, CMIP/CMISE provides services and functions for the dynamic creation, deletion, and manipulation of managed objects.

A.2.3.3 OSF DME

The Distributed Management Environment (DME) was created by the Open Software Foundation (OSF) to address the need for an integrated approach to distributed resource management. DME itself does not include much in the way of traditional management applications. Instead, it provides common platforms for developing those applications. Much of DME relies on previous OSF technology, the Distributed Computing Environment (DCE), and is oriented towards system management. DME consists of several components. The network management option (NMO) provides a platform for traditional network management applications. The NMO includes both SNMP and CMIP as underlying protocol options accessed via a common interface. For system management, DME provides an object management framework (OMF) and an object-oriented platform for distributed peer-to-peer applications. Lastly, DME provides a set of general services for distributed printing, event notification, license management, and software distribution.

A.2.4 SRM ARCHITECTURE AND SYSTEM MANAGEMENT STANDARDS

SRM proposes a generic model of management that is both bottom up and top down. From the bottom, the integrated control occurs at the points of potential conflict, i.e., the shared resources. From the top, objectives are used to resolve or avoid the resource conflicts. The major functions required by SRM are

- Monitoring (capacity analysis)
- Decision making (decision makers)
- Control (resource scheduling).

The components of the SRM architecture are resource managers, resource monitors, decision makers, and user interfaces. Outside of the user interface, the requirements of each component map into structures and services provided by the management model. The monitoring and control of a resource requires an abstraction of the resource in the form of management information, as well as protocols for the access to that information. Most of this information, although voluminous, is fairly generic and applicable to any number of management applications. Examples of existing MIB definitions that are potentially applicable to these needs are presented in the following sections. The decision-making component will require knowledge of user requirements and of how well those requirements can be met by the underlying resources, in terms of how well the objective functions map to the observed performance and capabilities of the resource. The resource managers provide the needed control over the resource in terms of both preventive operations (e.g., routing, scheduling, and bandwidth allocation) and reactive operations (e.g., overload control and window/rate control).

A.2.4.1 SRM Architecture

The SRM architecture defines at its lowest level resource monitors responsible for maintaining and reporting the status of the processing, communication, and storage resources. The resource monitor process represents an abstraction of the actual physical resource. In standards terminology, the resource monitor is the management agent. Each agent process provides basic information about the state of its resource and control over it. This syntactic information is represented by the MIB, is independent of higher-level management functional areas and can be architecture specific (by architecture, we mean the chosen structure of management information and communication, i.e., SNMP vs.CMIP/CMISE vs. others). Above this layer are generalized manager processes that perform some form of correlation and coordination of management information and control. It should be noted that many manager processes may also be significantly removed from the actual management applications, so as to be incapable of providing the needed semantic integration of the information. The SRM resource high-level decision makers represent application-specific manager processes. These applications provide the management and control of a distributed set of resources and hence the mechanisms necessary for performing the integration of SRM. However, by utilizing a uniform view of the resources in terms of standardized manager and agent interfaces and structure, the SRM management processes are reduced in complexity. This next level of abstraction within the management processes has been recognized as the next step in the evolution of management architectures. The recent IFIP conference on Integrated Network Management (IFIP 93) presented many works in progress in this area [Neumair 1993].

A.2.4.2 SRM Layered Approach

To begin the process of deriving our layered model, it is necessary first to select a generic system management model that will be suitable as a basis for refinement of the SRM-specific requirements: i.e., modelling the processor, communication, and storage resources and the SRM component requirements of performance monitoring and control. The natural choice is the use of the object-oriented paradigm. Among other reasons, this choice facilitates the mapping of the SRM requirements to the standardized management information. We have initially identified as our model the OSI object-oriented model, because it is felt to provide the needed abstractions and extensibility to support our project; however, other models should also be investigated for their suitability to SRM program and objectives.

The second step requires the characterization of performance-related properties that can be uniformly applied to all classes of relevant resources and are adequate for integrated performance management. This definition will be based on the information demands (objective functions) of the applications supported by SRM. Essentially, these high-level objective functions must be mapped to low-level capabilities. Some have referred to this mapping as the defining of "health functions" of the resource [Goldszmidt and Yemini 1993], while others use the term "quality of service." Health functions define observable, distilled points of resource behavior. SRM objective functions define what it means to be healthy as far as the user is concerned (i.e., the boundaries or thresholds of good health and bad health). To reduce the complexity, it is important that the identified health functions be small in number, yet accurately capture the essence of the capability of the resource.

The third step is to uniformly apply the health functions to the resources in our generic system management model. This step requires a mapping of our identified SRM health functions to the available standardized management information. Some of these capabilities and quantities are supported by the existing standards (ISO standard 10165-11 and RFC 1271); others can be derived from the provided information or will need to be defined completely by SRM developers. The next sections discuss these issues.

A.3 COMMUNICATION RESOURCES

SRM currently uses a simple model of communication resources, based on the end-to-end transmission time (which may be a function of the message length). This metric may be adequate when a traditional CSMA/CD single LAN environment is under consideration. However, when one expands the network across multiple LANs and into a WAN, the concept of delay does not accurately gauge the type of service available from the myriad communication resources in a heterogenous environment. The challenge is to distill the definition of capacity from the volumes of information coming from the communication resources.

A.3.1 COMMUNICATION SERVICE AND HEALTH METRICS

Very generally, communication resources provide transport services to end users. The types of service can be characterized as connection oriented, connectionless, and transactional. These service class distinctions will become important because they are a means of further refining the mapping of user objective functions (i.e., service requests) to resource health functions (i.e., the quality of a service). From the ISO perspective, it should be noted that the concept of service provider and service user is repeated throughout the layers of the underlying transport stack (i.e., the transport protocol machine uses the services of the network-layer protocol machine); hence, the concept of quality of service will vary with this relationship. Ideally, we would like to focus on the nature of the relationship of the user, as defined at the presentation layer and above, to the underlying service provided by the session layer and below.

The following list provides a simplified definition of communication resources health metrics:

- Connectivity: the current reachability of end-user systems.
- Throughput: the rate of transmitted and/or received user data.
- Utilization: the current load on the resource vs. its maximum capacity.
- Interarrival time: the time between packet arrivals; synchronous and isochronous data will require a near-constant interarrival time.
- Error rate: the ratio of packets lost to packets delivered, or the ratio of connections lost or released to the total number of connections.
- Delay: the length of time between the request for transfer of data by the source to the time of arrival at the destination.

It should be noted that many of the above quality-of-service metrics map directly to the ISO quality-of-service parameters (ISO standard 8072). Unfortunately, although the ISO community had thought to add the hooks for such concepts, up until recently very little work had been done on refining the QOS parameters, integrating them into the upper layers of the ISO stack, and, lastly, supporting them in commercial products. The Internet suite of protocols has not advanced further along in this aspect.

A.3.2 COMMUNICATION CONTROL

The monitoring of the communication-resource quality of service meets part of the SRM goals. The second and more complex aspect is the control of those resources. In the SRM model of operation, the SRM user requests a certain level of service expressed in terms of high-level objective functions. The SRM maps these requests to low-level capabilities and then either denies, negotiates, and/or expedites the request. During the period of service, the nature of the service may also change, requiring further potential termination or negotiation of service. The extent to which the SRM decision maker can control these resources is an area requiring further investigation. As an example, if alternate paths exist between two hosts, it is possible that each path can provide a different level of service (i.e., one path could be over a multihop extended LAN, and the other could utilize a high-speed ATM backbone). Whether the SRM is capable of specifying which route to take is dependent on the underlying protocols (i.e., source- and policy-based routing protocols).

A.3.3 MANAGEMENT DOMAINS

Additional issues that add to the complexity of communication resource monitoring and control are the possibility that the end hosts may reside in different administrative domains, or that they may require traversing publicly administered domains (i.e., the telephone network) for which little or no monitoring and control information is available.

A.4 HOST AND SYSTEM RESOURCES

The initial SRM studies have identified considerable work to date in the area of what can be termed low-level shared host resources and the techniques for controlling access to those resources. The initial target host for SRM, the Alpha operating system, provides the needed source code access to the host resources and scheduling algorithms. If SRM is to be viable on other platforms and environments, however, abstract structures and platform-independent mechanisms are required.

The concept of abstracting host resources within the system management model is new to both the Internet and OSI communities. As with all other efforts, the Internet community has made the quickest advances in identifying and formulating the attributes of a management information base for host resources. This work on a host-resources MIB is found in a draft document by Grillo and Waldbusser [1993]. Currently, this MIB definition is focused mainly on providing configuration information for the host (i.e., system up time, system storage capacity, available system devices, and installed software). A limited amount of performance data is defined and/or can be derived from the defined metrics (e.g., process load as a function of current loaded/running process contexts vs. the maximum number of process contexts supported; storage load as a function of storage size vs. storage used). This MIB definition (as with most other Internet MIBs) provides very few basic characterizations of an object's temporal behavior. Currently the MIB defines, on a per-process basis, the process's percentage of CPU usage and its total amount of allocated real system memory. SRM could participate in the refinement and extension of the host resources MIB as a starting point for modeling host resources.

An additional group providing mechanisms for management of host resources is the Desktop Management Task Force (DMTF). The DMTF is defining a desktop management interface (DMI) that lets host systems provide information to network management systems. The group has been working on the specification for over a year and preliminary products are expected within the next year.

A.5 FUTURE WORK

This appendix has sought to provide a rationale and strategy for future directions of the SRM program. The strategy is twofold. First, we propose to further the definition of communication resources within the SRM model. Second, we propose to base the management structure and communication as much as possible on standardized management interfaces and protocols. This latter strategy allows SRM to leverage considerable work within the network management community.

The tasks involved in these future directions are as follows:

- Identify communication-level objectives and characteristics for all classes of SRM users and applications.
- Identify existing efforts and progress in the implementation of standardized management models and architectures.
- Model SRM resource monitors and managers in terms of standardized models and architectures.

Demonstration milestones for these tasks would include the following:

- Initially, provide limited monitoring capabilities (identify key host and communication-level characteristics), but provide no control or low-level decision making.
- In the middle term, expand the monitoring capabilities to include additional metrics, and provide limited control and decision-making capabilities.
- In the long term, expand the decision making and control for SRM within an integrated system management environment.

A.6 REFERENCES

Goldszmidt, G., and Y. Yemini. 1993. "Evaluating Management Decisions via Delegation," *IFIP Transactions, Integrated Network Management—III*, eds. H. Hegering and Y. Yemini, North-Holland, New York, p. 247.

Grillo, P., and S. Waldbusser. 1993. "Host Resources MIB," draft-ietf-hostmib-resources-02.txt (30 June).

Neumair, B. 1993. "Modelling Resources for Integrated Performance Management," *IFIP Transactions, Integrated Network Management—III*, eds. H. Hegering and Y. Yemini, North-Holland, New York, p. 109.

Rose, M.T. 1990. *The Open Book: A Practical Perspective on OSI*, Prentice-Hall, Englewood Cliffs, New Jersey.

Westgate, J. 1992. *Technical Guide for OSI Management*, NCC Blackwell.

Appendix B

REPRINT OF SYMPOSIUM PAPER: ADAPTABLE SYSTEM RESOURCE MANAGEMENT FOR SOFT REAL-TIME SYSTEMS

Adaptable System Resource Management for Soft Real-Time Systems

Michael Davis and Alan Downing[*]

SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025

Thomas Lawrence

Rome Laboratory, 525 Brooks Road, Griffiss AFB, New York 13441

Abstract

SRI International (SRI), under the sponsorship of Rome Laboratory,[†] has developed a model for resource management in distributed soft real-time systems. The model considers user objectives, resource constraints, and adaptable execution techniques. We have developed the model in the context of a multimedia conferencing application within a command and control environment. We have implemented a prototype distributed multimedia display application that demonstrates key aspects of the model, including adaptation to a changing execution environment.

1. Introduction

The new world order will bring significant changes to the global mission of the United States military forces. As budgets shrink and strategies change, there will be an increased emphasis on joint operations and system integration; thus, commanders will need to do much of their planning in a distributed collaborative environment. Distributed database systems will maintain a common picture of the battlefield, and a common plan. Multimedia information such as imagery and video will become important components of the distributed planning process.

The information used for carrying out the planning process, and the results of the planning process, will be communicated among many users around a large, complex, distributed computing environment. This military computing environment will be a highly dynamic one characterized by

- Limited processing, communication, and data storage resources, and unpredictable loss of these resources
- Dynamic topology and configuration
- Changing threats and modes of operation
- Time-constrained operations.

[*]Mr. Downing's current address is Oracle Corporation, 400 Oracle Parkway, Redwood City, CA 94065.

[†]Contract F30602-91-C-0099, Rome Laboratory.

Even in this hostile and complex environment, high performance, security, reliability, and survivability will be required.

Unfortunately, few operating systems used in military systems (or in commercial systems, for that matter) have satisfactory support for many of these requirements. For example, real-time operating systems typically support hard real-time constraints that are too restrictive for many applications in this environment. Instead, systems that support soft real time (for example, the Alpha operating system [Northcutt and Clark, 1988]), and consider both the importance and the time constraints of competing activities when determining an execution plan, are more appropriate. Soft real-time systems support the graceful degradation of performance by dropping low-priority tasks during overloads. However, if resource availability is too limited or highly variable, additional support that considers resource limitations is required.

In addition, current operating systems manage individual system resources such as processing, communication, and data storage in an *ad hoc* manner. Different policies are used to manage different resources, and the management of the different resources is not coordinated, particularly when the resources are distributed. The relative importance of different activities is seldom considered in a uniform way across all the system resources. Such uncoordinated management results in suboptimal use of resources, especially when the availability of resources changes. For example, if radar data about incoming missiles are processed at a higher priority than sensor data about the weather, but weather data are transmitted on the communication channels at the same priority as missile data, then processing and communication resources will be wasted and military objectives may not be met.

Another problem is that current systems are too static for a dynamic military environment. A nonadaptive system can fail if it incorrectly assumes that its environment is known and predictable. Distributed systems in the military environment are subject to long-term, permanent changes due to failures or configuration modifications. Such changes mean that the underlying system will fail or will perform inefficiently unless fault tolerance and/or adaptability is incorporated into the system. Abstractions currently used by operating systems are not expressive enough to support adaptability. The ability to express and utilize tradeoffs in terms of resources used and benefit gained is missing. For example, the ability to select among alternative tasks with varying degrees of precision is not supported.

Under the sponsorship of Rome Laboratory, SRI International (SRI) has begun to address the problem of allocating limited resources among competing activities, according to command preferences for the activities and the usage restrictions imposed by the resources. In this paper, we discuss the results of a recent project sponsored by Rome Laboratory in which SRI considered this problem. We outline a high-level architecture for an adaptable soft real-time resource management system, and briefly describe a prototype application that can take advantage of such a system.

2. System Resource Management Model

In the System Resource Management (SRM) project [Davis, 1994], SRI developed a model for resource management in distributed real-time systems. This model considers user objectives, resource constraints, and adaptable execution techniques. The overall goal of the SRM project was to develop an approach to allocating distributed system resources (such as CPU, disk, memory, and communication channels) to multiple competing activities so that the objectives (e.g., level of service preferences) of each of these activities can be satisfied to the highest degree possible. We have developed a set of abstractions that make it possible to describe user objectives, system resources and constraints, and characteristics of execution techniques (e.g., resources used and objectives satisfied). We have implemented a prototype distributed video display application that demonstrates adaptation to a changing execution environment, using a subset of the elements of the model.

To support adaptability, objectives such as precision and timeliness are specified by means of *objective functions* (also called *benefit functions* or *value functions*). Objective functions express the benefit to the user or the system as a whole of achieving different levels of satisfaction of objectives, and can be used to determine appropriate tradeoffs among objectives. The objectives can be application specific (e.g., the benefit of different frame rates in a multimedia system) or system specific (e.g., the benefit of not exceeding various covert-channel bandwidths in a secure system). The model assumes that execution techniques (e.g., various data compression techniques) exist to produce results that fully or partially satisfy the objectives. There is a mapping between the techniques (and their parameters) and the degree to which they satisfy objectives. In addition, there is a mapping between the techniques and the amount of communication, storage, and processing resources they require.

Based upon information about objectives and resources, resource management decisions are made at two levels, as shown conceptually in Figure 1. Decisions involving medium- to long-range tradeoffs among activities and their objectives are made by a set of cooperating high-level decision makers. High-level decision makers base their decisions upon long-term resource usage statistics, external events, and domain-specific knowledge such as military doctrine or the observable effects of differences in multimedia quality.

Real-time scheduling decisions among competing tasks and threads (subtasks) within the selected activities are made by low-level schedulers for each resource. Each low-level scheduler uses status information about its resource, as well as process control abstractions passed down from the high-level decision makers, to make local decisions. While low-level resource scheduling is decentralized, integrated control (through shared resource status information and the consistent interpretation of process control abstractions) is used to ensure that the schedulers complement each other's decisions.

3. Multimedia Conferencing Application

Multimedia applications can consume large amounts of system resources; any or all of the processing, communication, and storage resources may become bottlenecks. Many multimedia applications, such as conferencing systems, have timing preferences that are not necessarily hard deadlines. In addition, multimedia applications have quality-related objectives such as video resolution and audio/video synchronization that have large impacts on resource usage.

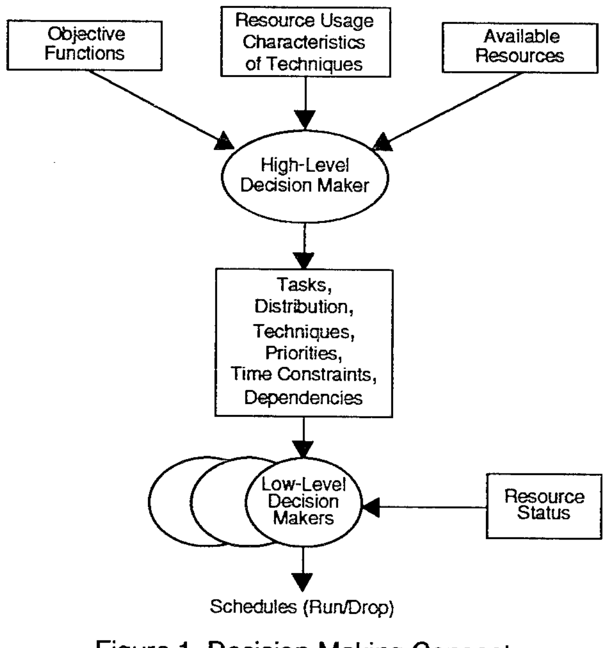

Objective Functions
Resource Usage Characteristics of Techniques
Available Resources
High-Level Decision Maker
Tasks, Distribution, Techniques, Priorities, Time Constraints, Dependencies
Low-Level Decision Makers
Resource Status
Schedules (Run/Drop)

Figure 1. Decision-Making Concept

For these reasons, we have defined a simple multimedia conferencing application to form the basis for a proof-of-concept implementation of adaptable system resource management concepts. The application enables a user at one site in a distributed system to present a briefing to a user at another site. The media include real-time video of the presenter, real-time audio of the presenter's voice, images of maps and slides (possibly overlaid by a movable pointer), and prerecorded audio/video clips. Most of the information flow is unidirectional, but there is a limited amount of feedback from the recipient to the presenter (e.g., audio questions). We assume that the resources (CPU, communication, storage) are not necessarily sufficient to fully satisfy the recipient's performance and quality objectives. Much of the discussion in this paper will be presented in terms of this sample application.

4. Objectives

The users of a distributed system have requirements and preferences regarding the resources that should be made available for accomplishing various tasks. For example, in a multimedia conferencing application, the recipient of a briefing has preferences regarding how the briefing information will be communicated and presented (e.g., the audio quality, the frame rate, and the image quality). These quality-of-service preferences, which we call ***objectives***, must be represented in such a way that they can be used by the system resource manager for making tradeoffs among different tasks and execution strategies.

Objectives may be defined at many levels of abstraction, including mission objectives, application objectives, and result objectives. In our model, objectives are associated with an application's results. All objectives are not relevant to all types of applications and results. For example, some applications do not have unique security objectives; they are willing to accept whatever security policy the system enforces. The security objectives may be applicable only to special management objects that make security-related decisions. An application can have many results and many ways of representing the results, and objectives can be associated with each one of these results.

As a hypothetical example, let us examine ways of assigning objectives to simulation results. The manner of displaying the data can be considered a result. The same numerical results can be presented to a user as a listing of values, as a simple bar graph, or as a series of 3-D images. These are different examples of a "display quality objective" of the numerical data. In addition, all of these displays can be presented in color or black and white, which could be considered an example of meeting a "display precision objective." Depending on the application, a decision to use black and white or color can indirectly affect performance, storage space, and communication bandwidth. In addition, the numerical data can be broken into pieces, each with its own time-benefit objective function. If performance becomes an issue, only parts of the data might be simulated in order to save time. The simulation results as a whole will have a precision objective, which specifies the number of runs versus the benefit (e.g., in terms of being statistically meaningful). This specification allows partial data to be returned to the user of the application, instead of all or nothing. Finally, there must be a way to express the relationships among the different objectives, such as their relative benefits. This expression scheme allows the resource manager to determine how to allocate its limited resources and to avoid wasting resources, for example, by doing a high-quality presentation of low-quality results.

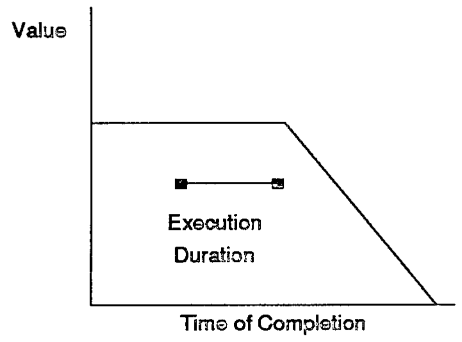
5. Objective Functions

An objective describes the importance to the system of producing a given result according to a given service specification. An objective has two aspects: (1) the degree or level to which some service is provided (e.g., the precision of a result), and (2) the relative benefit to the system of obtaining a given level of the service. The objective can be quantified by being expressed as a function relating the benefit to the level of service; for example, we could express a precision objective in terms of a precision-benefit function, where benefit = f(degree of precision).

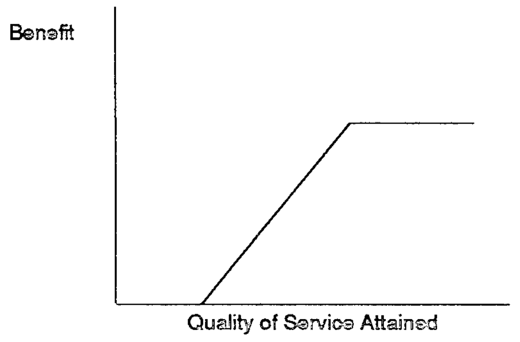
This abstraction is similar to the time-value function used in the Alpha operating system, in which the value of a computation is related to the time at which it is completed. We extend and generalize the Alpha abstraction to allow the specification of arbitrary objectives, not just timelines, and to include resources other than time—for example, memory or disk space. We generally use the term *benefit* rather than *value* when discussing objectives, since "value" is already used in many other contexts.

Figure 2(a) shows a graphical representation of a typical Alpha time-value function. The time-value (or time-benefit) function specifies the benefit to the system of completing a computation at various times. The time-value function expresses both the *importance* of completing a computation (as indicated by the value) and the *urgency* (as indicated by the time constraint). For reference, an indication of the expected duration of execution is also included, to allow a determination to be made of the time that a computation must be started in order to finish by a specified time.

Figure 2(b) shows a generic quality-of-service (QOS)-benefit function (which we also refer to as an objective function). The generic benefit function specifies the benefit to a user or to the whole system as a function of the quality of service (for example, precision or correctness) attained during a computation. In order to fully specify an objective function, it is necessary to define a metric that quantifies the level of service that is provided; it is useful if the metric is related to a quantity that can readily be measured or observed. For example, a metric for precision relating to a numerical calculation might be the number of decimal places in the result. It is also necessary to quantify



(a) Alpha Time-Value Function



(b) Generic QOS-Benefit Function

Figure 2. Time-Value Function and Generic QOS-Benefit Function

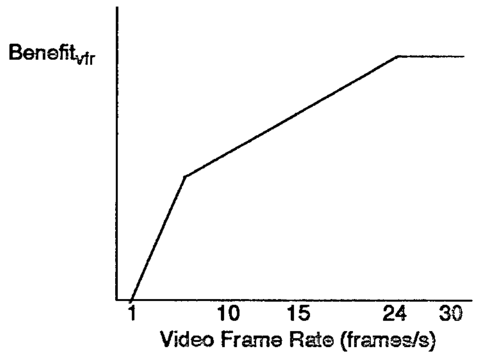
the relative benefit of achieving different values of the level-of-service metric, e.g., the relative benefit of three decimal places of precision versus four decimal places. The assignment of relative benefits is subjective and highly application and mission dependent, and may require experimentation. For some applications, the benefit increases as the precision is increased; other applications might not be able to take advantage of more precise data (for example, an application might be executing on a computer that has a low-resolution display), and the benefit would remain constant no matter how much the precision increased.

In a multimedia conferencing application, the recipient of a briefing has objectives regarding how the briefing information will be communicated and presented. For example, the benefit to a user of a stream of audio/video information is a function of the video resolution, the image size, the video frame rate, the time delay between the generation and display of the information, the audio fidelity, the degree of audio/video synchronization, and so forth. The recipient's objectives include human factors considerations relating to assimilation of video and audio information. For display of the presenter's face, the recipient may accept low frame rates, low resolution, and gray scale. However, a high frame rate, high resolution, and wide color range are desirable for prerecorded video clips of reconnaissance flights. The recipient may prefer as high a level of audio fidelity as possible, but may also be willing to accept telephone-quality audio; and may be able to tolerate several seconds of delay in the overall presentation, but may prefer that audio and video be synchronized to within a tenth of a second.

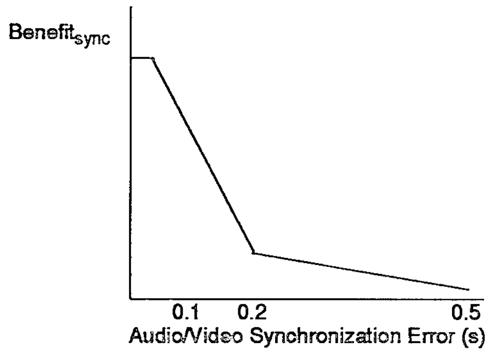
Figure 3 shows two sample objective functions for multimedia streams. In Figure 3(a), the benefit increases as the frame rate increases, up to a plateau after which the user perceives no improvement. In Figure 3(b), the benefit decreases as the time difference between the corresponding audio and video streams increases.

The benefits associated with given objectives can vary with the mission of the system, and in response to conditions. For example, some objectives are different during combat and noncombat periods; the changes in these objectives can be triggered by a change in the operational mode of the system. Objectives can also change dynamically with the state of the system; for example, in a missile defense system, as a missile gets closer the importance of dealing with the missile increases, relative to the importance of other activities.

By quantifying the level of service for a given objective and then assigning relative benefits to various levels, we can construct an objective function for any objective we wish to define. It is necessary to relate the benefit values for one objective to the benefit values for another objective. Using



(a) Frame Rate Benefit Function



(b) Synchronization Benefit Function

Figure 3. Sample Objective Functions for Multimedia

methods to compare benefits across objectives, a resource manager can use generic objective functions to make tradeoffs among objectives, without needing to understand the semantics of particular objectives. The use of generic objectives means that we do not have to define a fixed set of objectives that apply to all applications, but instead have the flexibility to define new objectives as appropriate.

When control decisions are made, the various system management objectives must be considered together. One approach is to try to combine the various objectives for each activity into a single composite objective that can be plugged into the resource management algorithm. Such an approach is likely to be computationally intractable when several objectives are involved, except, possibly, when it is used to configure the system. Another approach is to fix certain of the objectives on the basis of requirements and the environment, and use the remaining objectives for scheduling decisions. For example, all the objectives other than timeliness might be fixed (by the selection of operating modes that provide appropriate levels of each) and the scheduling would then be done in such a way as to maximize the timeliness objective. The order in which the objective functions are evaluated and the objectives are fixed must be carefully chosen, since the order can affect the outcome.

6. Characteristics of Execution Techniques

Based on the objectives expressed by the recipient (and to a lesser extent, the presenter), and on the availability of system resources, tradeoffs must be made and appropriate execution techniques must be selected. For example, by using appropriate compression techniques, it may be possible to reduce the use of communication resources at the expense of using more CPU resources (assuming no special hardware support). Using lower frame rates and lower video resolutions may free up CPU and communication resources to provide higher-quality audio. Using additional memory and disk buffers may improve audio/video synchronization, while increasing delays. The characteristics of the various execution techniques (in terms of resources used and effect on objectives) must be represented in such a way that they can be used by the system resource manager when selecting effective execution plans.

The important characteristics of execution techniques related to system resource management are (1) the resources required by a technique, (2) the benefits (in terms of meeting of objectives) provided to the user by a technique and (3) the dependencies among techniques. For example, providing one second of video of a given size, resolution, and frame rate at the receiving site, using a given encoding technique, may (1) require a certain amount of processing time and a certain amount of communication time, (2) meet the user's needs to a given extent, and (3) require that a certain communication protocol be used.

We have investigated various execution techniques that can produce results satisfying the objectives related to multimedia conferencing to varying degrees, using various resources. For example, there are several categories of compression and encoding algorithms, including frequency-, prediction-, and importance-oriented categories. Each category includes several related techniques, such as subband or transform coding in the frequency-oriented category. Which technique to select in order to maximize the benefit depends on the available resources (possibly including special-purpose hardware) and the user objectives (e.g., loss of information tolerated). We have also identified other techniques for selecting and maintaining frame rates, improving audio/video synchronization, reducing jitter, and supporting varying degrees of error correction.

When choosing among techniques, the resource manager must know how well each technique meets the current objectives, how many resources are consumed by each technique, and when changes in techniques or parameters can occur. Objectives such as delay, video resolution, and audio quality are all affected by the encoding technique used. Figure 4 illustrates approximate tradeoffs between

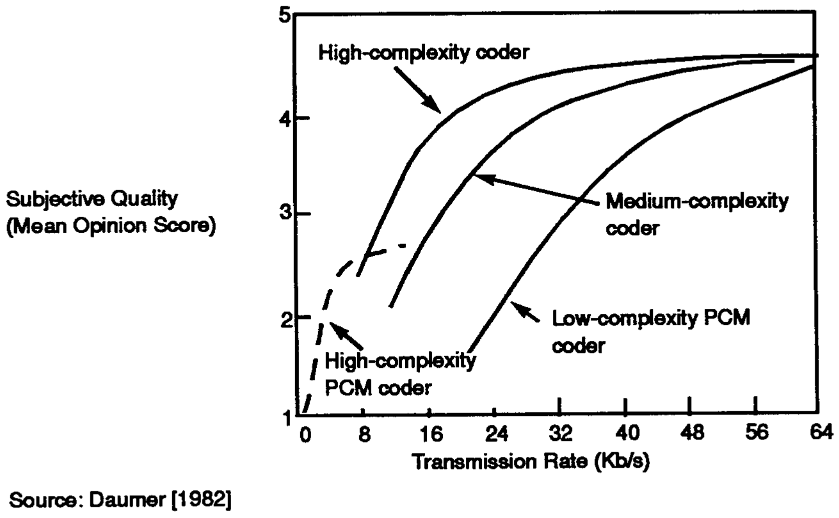


Figure 4. Tradeoffs for Audio Compression

quality (as expressed by a mean opinion score determined via experimentation), required processing resources (correlated with the delay objective), and transmission rates for various audio compression techniques [Daumer, 1982]. As expected, a low-complexity pulse code modulation (PCM) coder must transmit more data than a high-complexity coder to achieve the same subjective quality. A high-complexity vocoder, which takes advantage of the characteristics of human speech, can achieve the greatest compression factor but is limited to low-quality voice reproduction.

Similarly, there are tradeoffs among compression factors, quality, and resource usage for different video compression techniques. Video compression typically can achieve 1 bit/pixel (from the uncompressed value of 24 bits/pixel) with little perceivable loss of quality, which would reduce the data rate from the National Television Standard Committee (NTSC) value of 221 Mb/s to 9.216 Mb/s. Some algorithms are able to reduce video data rates to 1.5 Mb/s, although quality is lost. With further loss of quality (such as in video teleconferencing, i.e., CCITT H.261), data rates can be as low as 64 Kb/s or multiples thereof, with increasing quality up to about 1.5 Mb/s. Another example is the DVI system for compressed video, which uses lowered image and color resolution to achieve a factor of about 13 in data reduction and then applies a compression technique to yield an additional factor of about 15, resulting in about 8 MB/min of video image [Ripley, 1989].

Several compression and encoding techniques for audio and video separate the data conceptually into layers, such that the lowest layer provides at least the minimal acceptable quality (e.g., of resolution) and each successive layer improves upon the quality of the lower ones. Typically, the higher layers depend on the lower ones, and are worthless without them. This encoding scheme is known as hierarchical encoding [Shacham, 1992]. Hierarchical encoding allows a system to adapt quickly to varying environments; if it is not possible to process all the data, the system can simply drop the data in the highest layers (and therefore the data with the lowest relative importance). In a resource-rich environment, all layers are processed and high-resolution images are displayed, whereas in a resource-poor environment, only the lowest layers are processed and low-resolution images are displayed.

7. Scheduling Approach

As discussed earlier, a two-level approach to resource management and scheduling is appropriate for the multimedia conferencing application. A set of cooperating high-level decision makers make decisions involving medium- to long-range tradeoffs among activities and their objectives, and select the activities that will be permitted to run. Decentralized low-level schedulers associated with each resource make real-time scheduling decisions among competing threads within the selected activities.

The high-level decision maker considers the user objective functions, resource constraints, characteristics of candidate techniques, and system status information. The characteristics of the candidate techniques include the resources used by each technique, the degree to which each technique permits the objectives to be met, and the resources required to transition between techniques. On the basis of this information, the decision maker chooses the activities (e.g., multimedia conferences or periodic tasks) to execute; the nominal amount of resources to devote to each activity; and the techniques and parameters to use when executing these activities, such that the objectives can be met to the optimal degree consistent with satisfying the resource constraints. On the basis of these decisions, the decision maker then generates the information that is required by the low-level real-time schedulers, namely (1) the expected job lengths (execution times) for the threads within each activity, based on the selected techniques and parameters, and (2) the time-benefit functions for the threads, based partly on a composition of higher-level objective functions.

The high-level decision maker monitors the resource usage and the level of objectives that are being met. If the measured values differ significantly from the goal values previously specified by the decision maker, the decision maker takes compensating actions such as revising resource allocations, techniques, and parameters.

Given the combinatorial growth of the high-level scheduling problem (due to the number of alternative techniques, parameters, resources, activities, and so forth), the use of exhaustive or even statistical techniques is not feasible in real-time and dynamic environments. Various AI techniques have proven their ability to reduce the dimensions of large search spaces, such as occur in the SRM problem domain. The SRM decision maker may use different AI techniques for initial scheduling and revised scheduling, namely knowledge-based scheduling (which uses goals and constraints) and rule-based expert systems, respectively. Specifically, we are investigating the use of the capacity scheduling technique [Sycara et al., 1991], which has been proven successful in resource-constrained scheduling. Note that while the extensively adaptable application described here may use such techniques, other applications with fewer modes of adaptability could use simple decision makers based on tables generated from simulations or experimentation. The current prototype implementation uses simple heuristics rather than more powerful AI techniques.

Decentralized low-level resource schedulers are responsible for scheduling the threads within each activity (e.g., a thread that processes a frame within a briefing activity). Low-level scheduling decisions must have relatively low overhead and must be made quickly. The resource schedulers are associated with physical resources such as processors, disks, memory buffers, and communication channels, as well as with logical resources such as databases. Although decentralized, resource schedulers use integrated policies when determining their schedules.

The low-level schedulers use resource status information, control abstractions provided by the high-level decision makers (time-benefit functions and job-length functions), and additional information, such as dependencies, that is maintained by the application. Sample scheduling algorithms that use information similar to this include LBES [Locke, Jensen, and Tokuda, 1985], CBS [Peha and Tobagi, 1991], and DASA [Clark, 1990]. To adapt to short-term congestion, the low-level schedulers can drop the least important layers of hierarchically encoded multimedia information, enabling the multimedia application to gracefully degrade quality.

Integrated control among low-level schedulers requires that the schedulers consistently interpret abstractions (e.g., time-benefit functions, execution lengths, dependencies, and security levels) associated with the threads. The schedulers do not necessarily have to use identical policies, but they must work toward a common goal (unlike, for example, systems with processor schedulers that minimize missed deadlines, but disk and communication schedulers that maximize throughput).

8. Proof-of-Concept Implementation

We implemented a simple remote video presentation application in which video from a video camera or video cassette recorder (VCR) is captured and digitized at a computer workstation, transferred across a local area network to a destination workstation, and displayed on the monitor of the destination workstation [Davis and Siravara, 1993]. The key concept demonstrated is the adaptation of the distributed application to limited communication resources according to preferences specified by the user.

The architecture used for the implementation is shown in Figure 5. The implementation is done on Sun Microsystems, Inc. (Sun) SPARC workstations,* connected via an Ethernet local-area network, and executing version 4.1.3 of the SunOS (UNIX) operating system. An OpenWindows server process on the source workstation manages interactions with the video camera (via a Parallax video capture/compression board) and acts as a sending process. A controlling/receiving process executes on the destination workstation. An OpenWindows process on the destination workstation manages interactions with the display (via a Parallax board).

A video camera or VCR generates an analog video signal that is digitized and compressed by a Parallax board in the source workstation. The sending process and the Parallax board perform various operations on the digital video stream, such as modifying the frame rate, frame size, or Q (quality) factor. The resulting digital video stream is transmitted over the network to the destination workstation via TCP/IP. Additional processing is optionally done at the destination, and the video stream is then passed to a Parallax board, which uncompresses each frame and displays it in a window.

The controlling process has a user interface that allows the user to specify frame rate, frame size, and Q factor preferences, and to control operations such as starting and stopping the video capture and display. A high-level decision maker combines user preference information and status information from the receiving process to determine the optimal values for the frame rate, frame size, and Q factor. The selected values are communicated to the sending process, where they affect the operations performed by the Parallax board.

The user interface for specifying a typical benefit function (the benefit as a function of the video frame rate) is shown in

* All product names mentioned in this document are the trademarks of their respective holders.

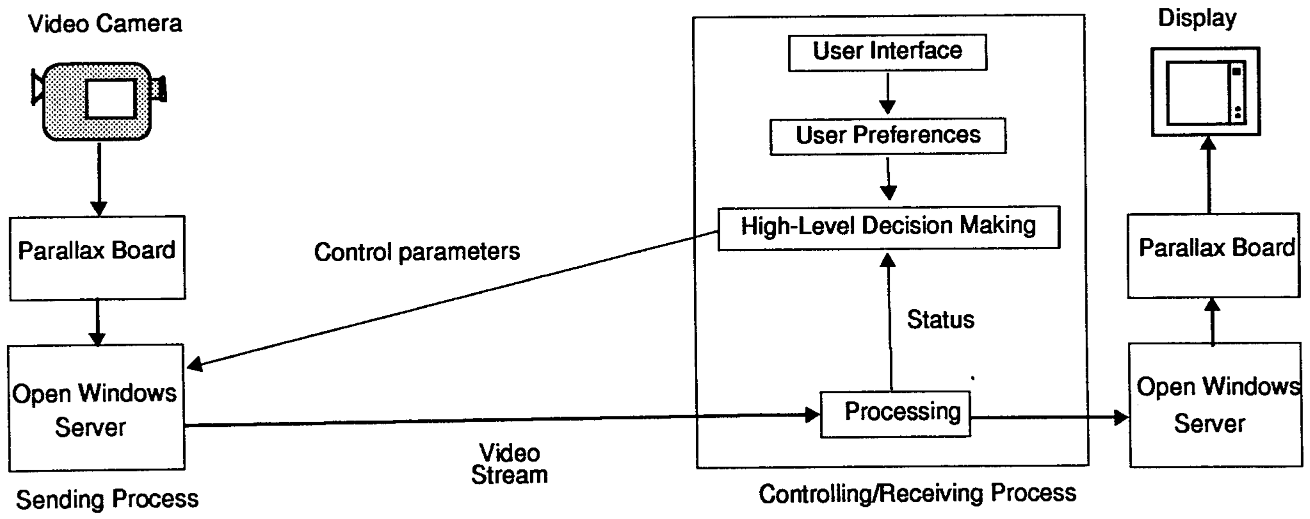


Figure 5. Implementation Architecture of SRM Multimedia Demonstration

Figure 6. The user draws the desired shape, via a computer mouse. The horizontal axis displays frame rates of 0 to 30 frames per second. The vertical axis displays the benefit on an arbitrary scale of 0 to 100.

The high-level decision maker considers the three objective functions and the measured average throughput when determining which values of frame rate, frame size, and Q factor are best. Because the bandwidth usage is proportional to the frame rate and the frame size, it is easy to calculate the change in required bandwidth that will result from a given change in frame rate or frame size. However, there is no simple relationship between changes in the Q factor and changes in the required bandwidth; for a given Q factor, the amount of compression attained (and therefore the required bandwidth) depends on the complexity of the contents of the frame. The decision maker must therefore determine the effect of changes in the Q factor by monitoring changes in bandwidth as the Q factor is varied.

Using a simple heuristic technique, the decision maker determines the combination of frame rate, frame size, and Q factor that produces the highest total benefit, given resource constraints. The heuristic initially assumes that the maximum frame rate, frame size, and quality can be attained. If measurements indicate that these results are not being attained, the heuristic determines which parameter can be decreased with the least loss of benefit. The appropriate parameter is modified, and the results are computed or measured, as appropriate. The process is repeated until the system is able to sustain the quality of service specified by the parameters.

The implementation is currently in the prototype stage, but preliminary results show that the system is able to adapt to available resources according to the preferences expressed by the user.

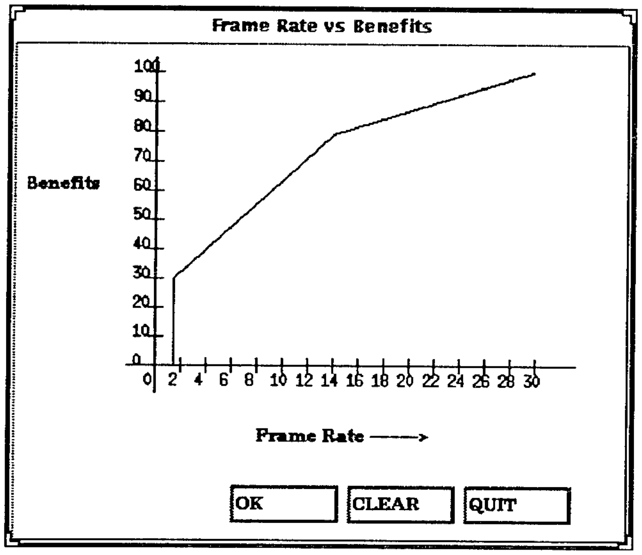


Figure 6. Specifying Frame Rate Benefit Function

9. Conclusion

We have developed a model for resource management in distributed soft real-time systems. The model considers user objectives, resource constraints, and adaptable execution techniques. We have developed the model in the context of a multimedia conferencing application in a command and control environment. We have implemented a prototype distributed multimedia display application that demonstrates key aspects of the model, including adaptation to a changing execution environment.

References

Clark, R.K. 1990. *Scheduling Dependent Real-Time Activities*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University.

Davis, M.B. 1994. *System Resource Management for Distributed Real-Time Systems*, Final Technical Report, ITAD-2655-FR-94-003, SRI International, Menlo Park, California.

Davis, M.B., and N. Siravara. 1993. *System Resource Management Demonstration Software*, Technical Report—Software Documentation, ITAD-2655-TR-93-434, SRI International, Menlo Park, California.

Daumer, W.R. 1982. "Subjective Evaluation of Several Efficient Speech Coders," *IEEE Trans. Communications*, pp. 655-662 (April).

Locke, C.D., E.D. Jensen, and H. Tokuda. 1985. "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proceedings IEEE Real-Time Systems Symposium* (December).

Northcutt, J.D. and R.K. Clark. 1988. *The Alpha Operating System: Programming Model*, Archons Project Technical Report 88021, Department of Computer Science, Carnegie Mellon University (February).

Peha, J.M., and F.A. Tobagi. 1991. "A Cost-Based Scheduling Algorithm to Support Integrated Services," *Proceedings IEEE INFOCOM '91*, Miami, Florida (March).

Ripley, G.D. 1989. "DVI-A Digital Multimedia Technology," *Communications of the ACM*, Vol. 32, No. 7 (July).

Sycara, K.P., S.F. Roth, N. Sadeh, and M.S. Fox. 1991. "Resource Allocation in Distributed Factory Scheduling," *IEEE Expert*, pp. 29-40 (February).

Shacham, N. 1992. "Multipoint Communication by Hierarchically Encoded Data," *Proceedings IEEE INFOCOM '92*, pp. 2107-2114, Florence, Italy (May).

Appendix C

RELATED WORK IN LOW-LEVEL INTEGRATED CONTROL

1 RELATED WORK IN LOW-LEVEL INTEGRATED CONTROL

This appendix is derived from the SRM Interim Technical Report of March 1993 [Downing and Davis 1993]. In it we summarize existing work related to low-level integrated control, and recommend areas where additional work would be useful.

Operating systems typically are designed with a particular objective in mind. For example, real-time systems may try to maximize the performance of the highest-priority activity, or to make all deadlines hard, or to maximize the benefit achieved by completing activities. Other systems try to maximize throughput, or promote fairness among competing activities. To best achieve any of these system objectives, it is important to use all the system resources in an integrated manner.

Unfortunately, many systems today do not have integrated resource management. In real-time systems, it is not unusual for the scheduler to use a real-time scheduling algorithm to determine which thread to execute next, while a disk manager or a communication manager in the same system uses a simple first-come-first-serve (FCFS) queue. Such conflicting policies prevent the system from meeting its objectives. For example, suppose a static priority scheduler starts a thread with priority 9 that quickly proceeds to asynchronously send 10 megabytes to disk. Soon thereafter, a thread is created with priority 10. Since this thread has the highest priority, the scheduler executes it; almost immediately the thread requests a synchronous read of one page from the disk. Since the disk manager uses a simple FCFS policy, the higher-priority task with very little information to read must wait a significant amount of time for the writes queued for the lower-priority thread to finish. Obviously, the policies used by the disk manager and scheduler to choose between activities should have been better integrated; they both could have used schemes based upon static priority scheduling. While this example shows only two threads with extremes in resource requirements, the same effect can be observed if there are many threads with similar resource requirements. Furthermore, similar examples can be easily constructed for many other scheduling algorithms (e.g., earliest deadline first) and other resource conflicts (e.g., "thrashing" caused by activities competing for memory).

Choosing a set of integrated policies for physical resource managers would normally be done when the system is being designed and would be nonadaptive. However, determining which are the optimum set of policies and the degree to which they should be integrated to produce the best overall results is difficult. For example, having the scheduler and the disk manager both use the same static-priority algorithm may not produce the best results in the case where the disk manager, for any kind of reads and writes, must choose between a read of a single page for a thread of priority 8 and a read of hundreds of pages for a thread of priority 10, while the CPU is idle.

Low-level integrated control would enable the managers of system resources used by multiple computational activities to cooperate among themselves in order to meet common goals, and would enable the computational activities to adjust the ways in which they use system resources. The manager of each resource can control which activities use the resource, but multiple managers must cooperate, for the activities to complete satisfactorily. Similarly, each activity controls which of multiple ways (using different amounts of resources) it uses to perform its function; but the activities need to cooperate with the resource managers, for the resources of the system as a whole to be used efficiently.

We separate the problem of integrated control into two categories:

- Coordinated control across resources such as processors, storage, and communication. Control policies must be chosen that can work together effectively for the different resources. Integrated control across nodes in a distributed system can be considered a variation of this category.
- Control based on multiple objectives for each activity, such as performance and precision. Resource constraints force trade-offs to be made regarding which objectives can be met.

In Subsections 1.1 and 1.2, we provide a review of past research on integrated control of resource schedulers. We identify the abstractions that are currently being used; any proposed abstractions should be a superset of the given abstractions. In Subsection 1.3, we identify future work.

1.1 INTEGRATED CONTROL ACROSS RESOURCES

Much of the research to date in integrated control across resources has been performed for real-time databases. Real-time databases use a rich set of resources and typically utilize soft, rather than hard, real-time control techniques; for these reasons they provide a good example for us to examine. In this section, we discuss integrated control across resources in the context of real-time database systems, with emphasis on concurrent transactions that compete for resources. In a more generic (nondatabase) example, we would be concerned with tasks or subtasks rather than transactions.

1.1.1 Work to Date

In Table 1 we list several resources that are controlled in real-time database systems, and a small sampling of the techniques used to manage each resource. In addition to the standard CPU, storage, and communication resources that we have discussed in previous SRM reports, we include memory buffer and shared data resources (the latter affects scheduling via concurrency control policies); we also list a small sampling of various techniques for managing overload.

We now briefly discuss each resource and its associated techniques.

CPU Scheduling. Many algorithms exist for scheduling the CPU resource for real-time transactions (or alternatively, for assigning priorities to transactions). These algorithms include the following:

- Earliest Deadline First (EDF)—The transaction with the earliest deadline is executed first.
- Static Priority (SP)—Each transaction is assigned a fixed priority that does not vary over time (which might be considered the importance or criticality of the transaction); the transaction with the highest priority is executed first.
- Least Laxity (LL)—The transaction with the least laxity (i.e., the least delay in starting such that it can still meet its deadline) is executed first.
- Locke's Best Effort Scheduler (LBES)—Unlike other scheduling techniques, LBES computes an entire schedule based on known jobs, rather than just determining the next job to be executed. LBES uses a two-pass algorithm to add potential jobs with

Table 1. System Resources and Techniques Used to Control Them

SHARED RESOURCE	REPRESENTATIVE TECHNIQUES
CPU	Earliest Deadline First, Static Priority, Least Laxity, Locke's Best Effort Scheduling, Dependent Activity Scheduling, Cost-Based Scheduling, Adaptive Earliest Deadline First
Buffers	Least Recently Used, Least Frequently Used, Cost-Based Dropping, Priority Least Recently Used
I/O (Storage)	First Come First Served, Shortest Seek Time, Elevator Scan, Earliest Deadline First, Highest Priority First, Feasible Deadline, Highest Priority Group First
Shared Data	Optimistic Concurrency Control (Broadcast, Wait), Two-Phase Locking (High Priority, Wait, Wait Promote, Conditional Restart), Timestamp, Multiversion
Communication	Earliest Deadline First, Least Laxity, Cost-Based Scheduling, Priority Token Bank
Overload Management	All Eligible, Not Tardy, Feasible Deadlines, Last Come First Dropped, Static Priority Dropping

the highest value densities (value accrued per unit of remaining computation time) added to a schedule, in EDF order, as long as the jobs can feasibly make their deadline.

- Dependent Activity Scheduling (DAS)—DAS considers potential value density (which includes the effect of dependencies such as precedence constraints and resource requirements) and deadline when determining a schedule. It constructs a schedule by considering activities in order of their value density, and inserting them into a partial schedule that is ordered by deadline. If the addition of the activity fails to produce a feasible schedule, the activity is dropped. Like LBES, DAS constructs a complete schedule of known activities rather than just determining the next activity to execute.
- Cost-Based Scheduling (CBS)—This algorithm was originally developed for communication scheduling and is now being adapted for CPU scheduling. The priority of a transaction depends on (1) the cost of not executing the transaction (as a function of time); (2) the expected execution time for the transaction; and (3) the anticipated delay in starting the execution. In high-level terms, the priority is a weighted average of the slopes of the cost function (which is similar to a time-benefit function).

- Adaptive EDF—The priority depends not only on the deadline, but also on the (static) importance of the transaction. The algorithm tries to maximize the total value of in-time transactions. It does not require job-length estimates, which may be hard to obtain in nondeterministic database systems.

In addition to deadline and laxity, other attributes that can be used to determine the priority of a transaction include importance (criticality), job length, amount of unfinished work, amount of work already invested, and arrival time.

Buffer Management. Buffer management allocates memory space among concurrently executing transactions, using admission control and buffer replacement algorithms. Buffer management policies include the following:

- Least Recently Used (LRU)—When it is necessary to reuse a buffer, the buffer that has not been used for the longest time is chosen.
- Least Frequently Used—The buffer that has been used the least frequently during a recent time interval is chosen for replacement.
- Cost-Based Dropping (CBD)—This algorithm was originally developed for managing communication buffers, but can be adapted to manage other buffer pools. The technique is analogous to CBS. In fact, a study was performed that investigated the combined affects of CBS and CBD, and reached the interesting conclusion that CBS with CBD is not much better than CBS with static-priority dropping.
- Priority LRU—Buffers are grouped into priority classes, based on the priorities of the transactions that used them. Both the priority and recency of use are considered when a buffer is chosen for replacement.

I/O Scheduling. The following algorithms are used for scheduling read and write operations to disk storage systems. Disk storage systems have the property that the major part of the latency is due to seeking between tracks. Sometimes, the assumption is made that only reads must be scheduled in real time, and that writes can be flushed to the disk at leisure.

- First Come First Served (FCFS)—Requests are processed in the order in which they are received.
- Shortest Seek Time—The request with the shortest seek time (based on the current position of the disk head and its direction of motion) is processed first.
- Elevator Scan—The disk head moves back and forth from end to end, servicing whatever requests are on its way, and changes direction whenever there are no more requests ahead in the direction it is moving.
- Earliest Deadline First (EDF)—The request with the earliest deadline is serviced first.
- Highest Priority First (HPF)—The request with the highest priority (as determined from the priority of the associated transaction) is serviced first.
- Feasible Deadline Scan—The disk head is sent in the direction of the highest-priority request, but feasible requests that can be serviced on the way are also serviced. (This is a combination of the elevator scan and HPF algorithms.)
- Highest Priority Group First (HPGF)—The requests are grouped into a small number of priority levels. The requests are scheduled in order of priority group; within each group, the elevator algorithm is used.

Shared Data. Concurrency control techniques, which determine which transactions can gain access to shared data and which transactions must abort or wait, affect scheduling because they remove activities from the run queue and place them on the wait queue. Database concurrency control techniques include the following:

- Optimistic Concurrency Control (OCC)—A transaction execution has three phases: a read phase, a validation phase, and a (possibly empty) write phase. If a conflict is detected during the validation phase, one or more transactions are aborted. In forward validation, the set of items read by the transaction is compared with items written by recently committed transactions. In backward validation, the set of items to be written by the transaction is compared with the items read by other currently active transactions. The following are some of the variations of OCC (backward validation):
  - OCC-Broadcasting Commit—The validating transaction always commits; all conflicting transactions are aborted. This is the traditional OCC approach.
  - OCC-Wait. If there is a conflicting transaction with higher priority than the one attempting to commit, the one attempting to commit waits.
- Two-Phase Locking (2-PL)—A transaction execution has two phases: a lock acquisition phase and a lock release phase. A transaction must obtain a lock before accessing a shared resource. The following variations of two-phase locking policies determine which transaction gets the lock when there is contention.
  - High Priority—The transaction with the highest priority gets the lock. If a lower-priority transaction was holding the lock, that transaction is aborted. In a variation of this algorithm (High Priority Without Cyclic Restart), the lower-priority transaction is not aborted if its priority immediately after being aborted would be greater than that of the current requester.
  - Wait—The requesting transaction must wait until the holder of the lock releases it, regardless of the relative priorities.
  - Wait Promote—The requesting transaction must wait. If the requesting transaction has a priority higher than that of the lock holder, the priority of the lock holder is raised to be the same as that of the requester.
  - Conditional Restart—This is the same as High Priority without Cyclic Restart, except that a lower-priority lock holder is allowed to continue running (with its priority promoted to that of the requester) if the remaining execution time for the lock holder is less than the slack time for the requester.
- Timestamp—Transactions are assigned timestamps when they enter the system, and are serialized in timestamp order. Any read or write operation that would invalidate the ordering causes the requesting transaction to abort. There is some question whether timestamp-based concurrency control is a feasible approach for real-time scheduling [Graham 1992].
- Multiversion Concurrency Control—Transactions are assigned timestamps when they enter the system. Transactions are allowed to read out-of-date versions of data items (in particular, they read the version with the most recent write timestamp that

is less than the timestamp of the transaction). A write operation is rejected if it invalidates a result previously returned by a read operation (in particular, if the next higher timestamp for the data item is associated with a read operation).

Communication Scheduling. Communication scheduling is necessary in distributed systems in which transactions communicate across nodes. Many scheduling algorithms for communication are analogous to the ones used for CPU scheduling, and include the following:

- Earliest Deadline First (EDF)—The request with the earliest deadline is serviced first.
- Least Laxity (LL)—The request with the least laxity is executed first.
- Cost Based Scheduling (CBS)—The priority of a request depends on the cost of not transmitting the packet (as a function of time), the length of the packet, and the anticipated delay.
- Priority Token Bank (PTB)—This algorithm attempts to allocate communication resources to multiple data streams in such a way that each stream's minimum performance requirements can be met. The algorithm uses a combination of stream admission control and packet scheduling. Packets to be transmitted are assigned to classes based on performance objectives. A counter associated with each class (which is decremented when a packet is transmitted and is incremented periodically) assures fairness among the classes.

Overload Management. If the system becomes overloaded (which is indicated if it begins to miss some of its deadlines), decisions must be made regarding which transactions (if any) to drop/abort. Some policies for dropping transactions are analogous to policies for scheduling transactions. Policies for overload management include the following:

- All Eligible—No transactions are dropped or aborted. (An assumption is made that there are sufficient resources to process all transactions eventually.)
- Not Tardy—Transactions that have missed their deadlines are dropped or aborted.
- Feasible Deadlines—Transactions whose estimated remaining execution time extends beyond their deadlines are aborted.
- Last Come First Dropped (LCFD)—The last transaction to arrive is dropped.
- Static Priority Dropping (SPD)—The transaction with the lowest static priority is dropped.

1.1.2 Sample Results

Combinations of many of the algorithms discussed above have been analyzed or simulated. The results indicate that certain algorithms work well together, while others do not. In addition, the best algorithm or combination of algorithms depends on the system load, the application, the resource capabilities, the costs of transaction restart, and other factors. While a few results can be stated generally (for example, EDF performed very poorly in overload situations, and OCC performed better than 2-PL and timestamp concurrency control for most real-time database systems), it is better to review the related documentation to understand the associated assumptions and conditions.

In general, the more information the algorithms used, the better the results. For example, FCFS used the least amount of information, and consistently performed the worst of the algorithms. With a little more information associated with a job, such as a deadline or a static priority, significantly better results are possible. Slightly better results can be achieved if job length is known. Further improvements are possible with the use of time-value functions and estimates of transaction restart costs. While additional information improved results, it often diminished returns. Obtaining additional information can be difficult or expensive; for example, queries to database systems typically have nondeterministic execution times.

Consistent policies must be used to manage resources. For example, FCFS with EDF can conflict, causing priority inversion. Unfortunately, real-time systems often use this combination, because many commercial communication or I/O systems do not use priorities or other real-time information. Even those algorithms that use the same information may have conflicting policies. For example, a system using High Priority to resolve shared data conflicts, where priority is defined by Least Laxity, could result in cyclic scheduling where the aborted job aborts the job that aborted it.

1.2 INTEGRATION OF OBJECTIVES

Several scheduling algorithms allow trade-offs among multiple (typically, two) objectives, in contrast to the traditional scheduling approach, which considers a single objective such as fairness (FCFS), static priority, or a deadline. In Table 2 we list several algorithms that consider two or more competing objectives.

Table 2. Scheduling Algorithms That Consider Two or More Objectives

OBJECTIVES	REPRESENTATIVE ALGORITHMS
Timeliness and importance	Locke's Best Effort Scheduling, Cost-Based Scheduling, Dependent Activity Scheduling
Timeliness and precision	Liu et al. [1991], Epsilon Serializability, Chaos, COBASE
Timeliness, importance, and precision	Moiin and Smith [1992], SRM's OR-Dependency Approach
Recency and priority	Priority-Least-Recently-Used Buffer Scheduling
Meeting deadlines and response time	Highest-Priority-Group First I/O Scheduling
Timeliness and covert channel bandwidth	Secure Alpha

We now briefly describe each algorithm and how the trade-offs can be specified.

Timeliness and Importance. Several algorithms use both timeliness (based on such measures as deadline and expected execution time) and importance or criticality (often expressed as "value" or its opposite, "cost") in determining schedules. The algorithms described here use time-value or time-cost functions to express both importance and timeliness objectives.

- LBES—LBES has been described in Subsection 1.1.1. It uses time-benefit functions and job lengths as its abstractions.
- CBS—CBS has been described in Subsection 1.1.1. It considers cost functions (which are similar to time-benefit functions), anticipated delay, and expected execution times to determine the highest priority task to execute.

- DAS—DAS has been described in Subsection 1.1.1. It uses time-benefit functions, precedence constraints, and job lengths as its abstractions.

Timeliness and Precision. Several researchers have considered both timeliness and precision in their scheduling algorithms. In some cases, the importance of a task (as expressed in a value function) has also been included. These algorithms typically consider computations in which the degree of precision varies with execution time, because of the periodic checkpointing of partial results that become more accurate as the execution progresses; or different modes of operation that give different precisions (and require different execution times); or splitting the execution into subtasks that build upon each other to increase precision.

- Liu et al. [1991]—This algorithm considers tasks that can be broken into mandatory tasks and optional subtasks; there is a partial benefit from doing only the mandatory subtasks, and additional benefit from doing the optional tasks. The algorithm uses a variation of EDF to schedule mandatory tasks first and then as many optional subtasks as possible. The algorithm computes a complete schedule of known tasks, rather than just the highest-priority task.
- Epsilon Serializability—This technique trades off timeliness and accuracy (consistency) by allowing the reading of out-of-date data for certain types of queries. Knowledge about the semantics of the data is used to bound the degree of inconsistency.
- Concurrent Hierarchical Adaptable Object System (CHAOS)—This hard real-time system modifies the modes of operation (trading off less precise but faster modes for more precise but slower ones) in response to changing loads or other events.
- COBASE—If a query results in a null response, the query is relaxed, and approximate, summary, or related results are returned instead. This system sacrifices precision (or accuracy) in order to provide timeliness and partial functionality.

Timeliness, Importance, and Precision. The algorithms that consider these three objectives more closely address the problems of the SRM project, because many high-level objectives can be mapped into these three low-level objectives.

- Moiin and Smith [1992]— This algorithm considers the case of tasks that are capable of generating intermediate results with varying degrees of precision, and in which the overall value can be considered a function of both time and precision. Optimal and heuristic solutions for various categories of time-value and precision-value functions are analyzed.
- SRM's OR-Dependency Approach—If there are several ways in which a result could be generated (with different precisions), the scheduler considers all of them. As soon as one of the alternatives is executed, the others are dropped. This approach uses time-benefit functions, job lengths, and OR dependencies.

Recency and Priority. As described in Subsection 1.1.1, Priority LRU (PLRU) buffer replacement uses a single parameter to control the tradeoff between recency and priority.

1.3 RECOMMENDED FUTURE WORK ON LOW-LEVEL INTEGRATED CONTROL

Low-level integrated control involves the research and development of real-time scheduling policies and abstractions for resources such as communication, disks, CPUs, and shared data. Analysis, simulations, and implementations should be used to determine which scheduling policies complement each other, and if, when, and how these policies can be enhanced to consider more complex abstractions (e.g., to support tasks of variable precision).

Our investigation of low-level integrated control across resources has revealed several areas that have not yet been sufficiently researched. First, the work to date has emphasized databases with nondeterministic query lengths. On the other hand, scheduling approaches that use time-benefit functions for nondatabase activities of deterministic lengths have been partially studied only twice: (1) the integration of CPU scheduling and conditional-restart conflict resolution using DAS, and (2) the integration of CBS scheduling and buffer management techniques. In addition, the integrated control of communication with other resources has not been considered. For example, how would CBS scheduling for the CPU with priority token bank scheduling for the communication channels perform, compared to LBES with EDF?

Even real-time control of single resources needs additional research. For example, predictable communication over multiple hops is an open research area, as are I/O scheduling techniques that use more enriched information (e.g., time-value functions). Alternate techniques of using time-benefit functions, such as CBS and LBES, have not been sufficiently compared. In addition, security abstractions that allow trade-offs between covert channels and performance require more investigation.

Some scheduling approaches have considered trade-offs among particular objectives. The approaches that trade off precision, importance, and timeliness are the most general, since many high-level objectives can be mapped into these low-level objectives. There are alternative abstractions for supporting precision: precision-value functions, OR dependencies, and precedence constraints. The advantages and disadvantages of these abstractions for precisions must be further investigated, for example, by using simulations to compare (1) CBS with OR dependencies of alternate precisions; (2) LBES adapted to support alternate precisions using an approach like that of Liu et al. [1991]; and (3) the approaches proposed by Moiin and Smith [1992].

Nonheuristic approaches that deal with multiple objectives are likely to be far too computationally complex to be of practical use in real-time systems. More research on heuristic techniques is needed.

Most of these areas of investigation would probably best be addressed through a wide range of simulations. Additional insight into low-level integrated control should be obtained through actual prototype implementation on a real operating system, such as Alpha OS.

2 REFERENCES

Downing, A.R., and M. Davis. 1993. "System Resource Management for Distributed Real-Time Systems: Sample Architecture," Technical Report, ITAD-2655-TR-93-59 (March).

Graham, M.H. 1992. "Issues in Real-Time Data Management," *The Journal of Real-Time Systems*, Vol. 4, pp. 185-202.

Liu, J.W.S., K. Lin, W. Shih, A.C. Yu, J. Chung, and W. Zao. 1991. "Algorithms for Scheduling Imprecise Computations," *IEEE Computer* (May).

Moiin, H., and P.M.M. Smith. 1992. "Better Late Than Never," submitted for publication, University of California, Santa Barbara, California.

Rome Laboratory

Customer Satisfaction Survey

RL-TR-________

Please complete this survey, and mail to RL/IMPS,
26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.
Thank You

Organization Name:______________________________________(Optional)

Organization POC: ______________________________________(Optional)

Address:____

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful 1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No______

If yes, please identify the area(s), and comment on what aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

☆U.S. GOVERNMENT PRINTING OFFICE: 1995-710-126-20058